Petri Nets Modeling for the Schedulability Analysis of Industrial Real Time Systems

Alessandro Fantechi and Stefano Pepi
DINFO, University of Florence, Via S. Marta 3, Firenze, Italy

Keywords: Petri Nets, Timed Petri Nets, Coloured Petri Nets, Real Time Systems, Scheduling Algorithm, Modeling, Formal Verification, Railway Signalling.

Abstract: In the experience of a railway signaling manufacturer, schedulability analysis takes an important portion of the time dedicated to configure a complex, generic, real-time application into a specifically customized signalling embedded application. We report on an approach aimed at substituting possibly unreliable and costly empirical measures with rigorous analysis. The analysis is done resorting to modeling the scheduling algorithms by Petri Nets. We have compared two types of Petri Nets: Timed Petri Nets (TPN) and Coloured Petri Nets (CPN), supported by open source tools, respectively *TINA* and *CPN Tools 4.0* concluding that the latter are more suited for the dealt problem.

1 INTRODUCTION

Real-Time Systems (RTS) are those computer-based systems where correct operation does not only depend on the correctness of the results obtained, but also on the time at which the results are produced (Stankovic, 1988).

The interest for real-time systems is motivated by many applications that require that computations satisfy given time constraints, in domains such as automotive, avionics, communications, railway signalling etc.

The most important property of a RTS is *predictability*. Predictability is the ability to determine in advance if the computation will be completed within the time constraints required. Predictability depends on several factors, ranging from the architectural characteristics of the physical machine, to the mechanisms of the core, up to the programming language. Predictability can be measured as the percentage of processes for which the constrains are guaranteed.

In this article we report the experience made in collaboration with our industrial partner, a railway signalling manufacturing company, in the implementation of a generic real-time platform based on a proprietary microkernel Real Time Operating System; in particular we present a method for schedulability analysis.

With the recent expansion of markets to Asia and Africa, the company has experienced a growing need for a versatile system that can be configurable for each different application. The transition from a traditional "main loop"-based system to a general purpose platform has allowed low-cost configuration, simply by changing the application inside and the hardware to interact with. With the same Hw/Sw platform both *ground* and *on-board* systems can be built, either for urban (like metro) or main line applications, meeting the signaling regulations of different countries.

Experience has however shown that guaranteeing predictability for the different customizations of the platform takes a considerable portion of the customization effort, if based only on testing every time the newly customized software on the platform.

We have therefore considered the possibility of building a generic model of the scheduling algorithms employed in the platform that is going to be instantiated on the temporal constraints and tasks numbers of the different specific applications (that is, customizations), in order to support the validation of predictability by means of proper model simulation tools. Basing on the wide literature about modeling real-time systems with Petri Nets (see, for example, (Barreto et al., 2004; Berthomieu and Diaz, 1991; Felder et al., 1994; Grolleau and Choquet-Geniet, 2002)) and on the availability of related tools, we have chosen to experiment two Petri Nets dialects for the modeling of the scheduling algorithms, in order to predict schedulability of the set of tasks governing a new specific ap-

Fantechi, A. and Pepi, S.
Petri Nets Modeling for the Schedulability Analysis of Industrial Real Time Systems.
DOI: 10.5220/0005841700050013
In *Proceedings of the International Workshop on domAin specific Model-based AppRoaches to vErificaTion and validaTiOn (AMARETTO 2016)*, pages 5-13
ISBN: 978-989-758-166-3

plication. Both Timed Petri Nets (TPN) and Coloured Petri Nets (CPN) have been evaluated for this purpose, together with their support tools, favouring at the end the adoption of Coloured Petri Nets.

Due to the limited time available to conduct the experiments, in order to satisfy stringent temporal requirements from our industrial partner, we have chosen not to investigate other temporal modeling formalisms, such as timed automata (Alur, 1992). The results obtained by these experiments were however judged sufficiently satisfactory to consider the adoption of the technique inside the development process of our industrial partner.

2 PRE-RUNTIME SCHEDULING IN SAFETY RELATED RT APPLICATIONS

A real-time process is characterized by a fixed time limit, which is called deadline. A result produced after its deadline is not only late, but can be harmful to the environment in which the system operates. Depending on the consequences of a missed deadline, real-time processes are divided into two types:

- *Soft Real-time:* if producing the results after its deadline has still some utility for the system, although causing a performance degradation, that is, the violation of the deadline does not affect the proper functioning of the system;
- *Hard Real-time:* if producing the results after its deadline may cause catastrophic consequences on the system under control.

To meet this requirement, scheduling plays an important role. Depending on the assumption done on the processes and on the type of hardware architecture that supports the application, the scheduling algorithms for real-time systems can be classified according to the following orthogonal characteristics:

- *Uniprocessor* vs. *Multiprocessor*
- *Preemptive* vs. *No preemptive*
- *Static* vs. *Dynamic*
- *Pre-runtime* vs. *Runtime*
- *Best-Effort* vs. *Guaranteed*

For what concerns the fourth characteristic, in pre-runtime scheduling all decisions are taken before the process activation on the basis of information known a priori. The schedule is stored in a table which will be integrated into a run-time kernel. The kernel has one component called *dispatcher* which takes tasks from the table and loads them onto the processing elements, according to specified timing constraints. The *Runtime* category represents instead those algorithms in which the scheduling decisions are made at runtime on all current active processes. The ordering of tasks is then recalculated for each new activation.

The choice between pre-runtime or run-time scheduling has been done in our case in favour of the former thanks to the better possibility to demonstrate predictability, which is a must in a safety-critical environment. That is, with pre-runtime scheduling it is possible to exhibit to an assessor the analyses conducted on the considered set of tasks in order to establish that tasks do not miss their deadline, while with runtime scheduling evidences provided simply by running tests can be not convincing about their coverage of all possible cases.

Indeed, the present paper aims to show a method to strengthen the analysis on pre-runtime scheduling to a high level of confidence. In particular, we present a method that can be used to verify the pre-runtime schedulability of a task set that contains only periodic tasks with time and priority constrains.

The motivations for this approach come also from the high variability of installations of the same signalling system at different locations or controlling different stations or lines. Indeed, in railway signalling systems, a distinction is often done between *generic applications* and *specific applications* (see, e.g., the CENELEC EN50128 (Cenelec, 1996) guidelines): generic software is software which can be used for a variety of installations purely by the provision of application-specific data and/or algorithms. A specific application is defined as a generic application plus configuration data, or plus specific algorithms, that instantiate the generic application for a specific purpose.

While the platform is part of a generic application, and hence it is validated once for all, for each specific application the satisfaction of real-time constraints must be verified from scratch.

Indeed, quite often in everyday work it is necessary to revise the schedule of some systems, and all this is routinely done in an empirical way. It is clear that each application has a different way to interact with the platform and especially with its resources, such as, for example, input/output drivers for different hardware. It is for this reason that the schedule of real-time tasks should be revised at any new specific application.

The adopted empirical approach includes actions to be taken when configuring the platform for a new specific application, such as: get a new schedule configuration offline and test it on the target. It rarely happens

that the first test is successful.

The estimated effort required for the identification and testing of a new configuration of scheduling can be summarized with the following parameters:

- **Offline Identification Time:** time needed in order to design the new schedule, it is usually about 30 minutes.
- **Flashing Time:** the time needed to load the scheduling on the target, 15 minutes.
- **Startup Time:** start-up time of the platform, 1.5 minutes.
- **Running Time:** time during which the system must run without exhibiting timing problems, 30 minutes / 1 hour.
- **Attempts:** average number of attempts to get the scheduling, 3.

Summing all the times shown above we get that for each test scheduling, the whole process easily reaches 8 hours, which means an entire working day. This process can be automated by a tool that, given a task set and a number of constraints, is able to produce a feasible scheduling. This would mean a huge saving in terms of man hours used to refine the scheduling. Moreover, an empirical evaluation of schedulability of a given dataset does not guarantee that the deadlines are met in any case, putting in danger the overall safety of the system. Using a rigorous approach to the analysis of the schedulability will improve hence the conformance, of a specific application, to safety guidelines.

3 TASKSET AND CONSTRAINTS SPECIFICATION

In our system the application is decomposed into a set of tasks $\tau_i : i = 1,...n$ and for this paper we only consider periodic tasks, and we assume that non-periodic tasks are carried out by a periodic server, or processed in the background (Buttazzo, 2011). The temporal model mostly used in real-time scheduling theory is an extension of the model of Liu and Layland (Liu and Layland, 1973) where each task τ_i is characterized by the following parameters:

- R_i: first release time of τ_i;
- C_i: run time of τ_i, which is its worst case execution time (WCET);
- D_i: relative deadline of τ_i, the mamimum time elapsed between the release of an instance of τ_i and its completion;
- P_i: release period of τ_i.

In the following we use as a running example the case of a real signalling application, an interlocking system. An *interlocking* system is the safety-critical system that controls the movement of trains in a station and between adjacent stations. The interlocking monitors the status of the objects in the railway yard (e.g., points, switches, track circuits) and allows or denies the routing of trains in accordance with the railway safety and operational regulations that are generic for the region or country where the interlocking is located. The instantiation of these rules on a station topology is stored in the part of the system named control table that is specific for the station where the system resides. We refer to (Fantechi, 2013) for a review on how interlocking functionality is formally modeled. In this context, we are interested to focus on the characteristics of the task set of this application, consisting of 7 threads which have the following goal:

- T_1 is in charge of operating on the Ethernet channel;
- T_2 is one of the most important thread and it is in charge of the safety of the system;
- T_3 implements a protocol stack for the receipt and transmission of messages;
- T_4 is in charge of copying the value received in the input of the Business Logic and preparing the output for the trasmission.
- T_5 is the application thread that contains the logic of the system.
- T_6 is a diagnostic thread;
- T_7 is a USB driver used for logging data in a key.

The scheduler operates by dividing processor time into epochs. Within each epoch, every task can execute up to its time slice. In this case, the scheduler has two epochs of 100 milliseconds and the taskset have the following constraints:

- The total time of scheduling cycle is 200 milliseconds.
- Each epoch needs to last exactly 100 milliseconds.
- The first execution of T_3 in the first and second epoch must terminate within 95 milliseconds.
- The second execution of T_3 in the first and second epoch must terminate within 95 milliseconds.
- The second execution of T_3 in the first and second epoch must execute at least 65 milliseconds after the first one.
- T_4 in the first epoch must terminate within 90 milliseconds and in the second epoch in 140 milliseconds.

- The sum of times for T_5 must be of at least 90 milliseconds.

The taskset used in our example is defined in the Tables 1 and 2 with the relative scheduling order and parameters.

Table 1: TaskSet in first epoch.

Epoch1	R_i	C_i	D_i
T_1	0	6	6
T_2	6	5	11
T_3	11	16	27
T_2	27	5	32
T_3	32	4	36
T_4	36	24	60
T_5	60	40	100

Table 2: TaskSet in second epoch.

Epoch2	R_i	C_i	D_i
T_1	0	6	6
T_2	6	5	11
T_3	11	16	27
T_2	27	5	32
T_3	32	4	36
T_5	36	40	76
T_4	76	8	84
T_6	84	10	94
T_7	94	6	100

These assumptions are the basis on which a model of the scheduling algorithm can be built. We resorted to the use of Petri Nets, that result quite intuitive in the modeling of scheduling algorithms (Berthomieu and Diaz, 1991; Felder et al., 1994; Leveson and Stolzy, 1987; Tsai et al., 1995). In order to represent time, we have investigated the use of bothTimed Petri Nets (TPN) (Ramchandani, 1974) and Coloured Petri Nets (CPN) (Jensen, 1987). In the following part of the article we illustrate the two kinds of models by means of the running example, giving a comparison between the two modeling approaches.

4 PROPOSED METHOD

A Petri Net (Petri, 1962; Murata, 1989; Peterson, 1981) is a mathematical representation of a distributed discrete system. As a modeling language, it describes the structure of a distributed system as a bipartite graph with annotations. A Petri Net consists of places, transitions and directed arcs. There may be arcs between places and transitions but not between places and places or transitions and transitions.
The places can hold a certain number of tokens and the distribution of tokens on all the places of the network it's named marking. Transitions act on input tokens according to a rule, that is named *firing rule*.
A transition is enabled if you can fire it, that is, if there are tokens in every input place. When a transition fires, it consumes tokens from its input places and places a token in each of its output places.

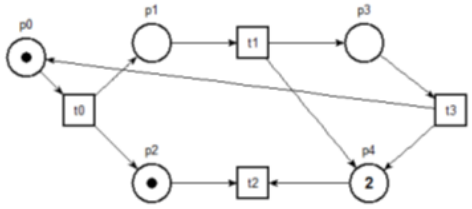

Figure 1: Representation of an ordinary Petri Net.

In Figure 1 is an example of an ordinary Petri Net. The execution of Petri Nets is not deterministic, that is, if there are more transitions enabled at the same time any of them can fire. Since taking a transition is not predictable in advance, Petri Nets are well suited for modeling the concurrent behavior of distributed systems.
Formally we can define a Petri Net as a tuple $PN = (P,T,F,W,M_0)$ where:

- P is a finite set of *places*;
- T is a finite set of *transition*;
- $F \subseteq (PxT) \cup (TxP)$ is a set of *arches*;
- $W : F \to \mathbb{N}$ represents the weight of the flow relation F.
- $M_0 : P \to \mathbb{N}$ is the initial marking vector, which represents the initial state of system.
- $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

4.1 TPN

A Timed Petri Net is a Petri Net extended with time. In Timed Petri Nets, the transitions fire in "real-time", i.e., there is a (deterministic or random) firing time associated with each transition, the tokens are removed from input places at the beginning of firing, and are deposited into output places when the firing terminates. Formally we can define a Timed Petri Net (Ramchandani, 1974) as a tuple $TPN = (PN,I)$ where:

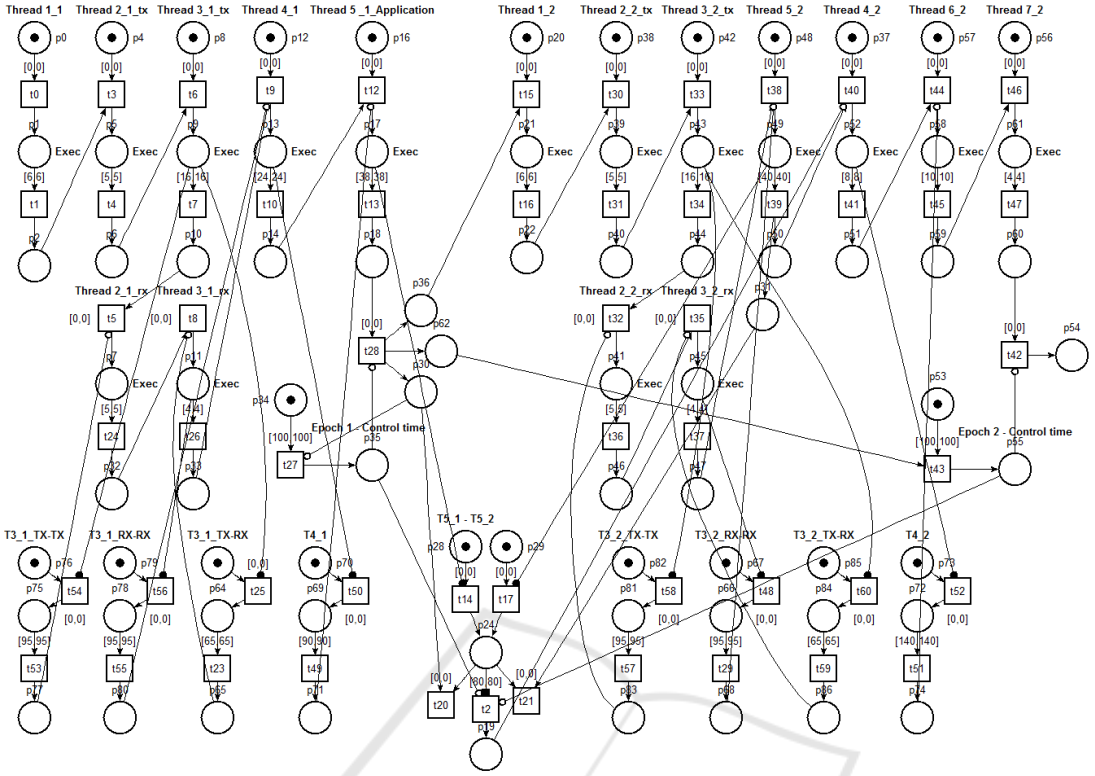

Figure 2: Timed Petri Net model for a fixed scheduler.

- PN is a standard Petri Net;
- $I : T \to \mathbb{N} \times \mathbb{N}$ is a function that maps each transition to a bounded static interval
- $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

In Figure 2 is reported the model generated with the tool TINA (Berthomieu and Vernadat, 2006) for a fixed scheduler (Grolleau and Choquet-Geniet, 2002; Barreto et al., 2004; Aalst, 1996). As we can see the representation with TPN is a little bit chaotic and representing larger sets of tasks could be very difficult. Looking at the model we can underline some diagram parts which are used for the verification of constraints (Tsai et al., 1995):

- **Check for the Total Time**
  The network used to control the time of each epoch consists of two transitions and respectively five and three places. Taking into consideration the network a) in Figure 3, the transition t27 counts the total time available for the execution in the epoch. When the time available ends, the token content in place p34 is moved in place p35 and inhibits the passage of the token, from the last thread running, in places p36, p62 and p30.
  If this happens it means that the execution time in this epoch is not the one expected. If, however, the performance ends too soon the transition t28 will not be inhibited by p35 place and pass tokens in places p36, p62 and p30, decreeing the passage of the positive control and the start of the second epoch.

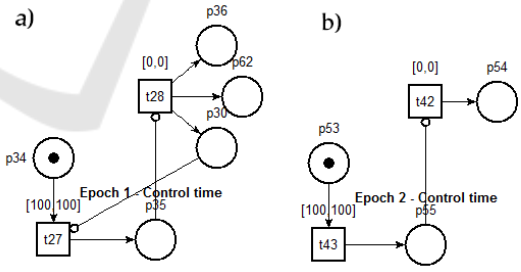

Figure 3: Diagram of epoch control block.

- **Check Constrains on T_3**
  The network in Figure 4 models the various controls on the execution times for T_3. For example the last block checks that between the first execution of T_3 in the first epoch and the second execution in the second epoch, at least 65 milliseconds have expired. The transition t25 is enabled when the task is running and, if the task completes before the time set in the transition t23, scheduling can continue. Otherwise, if the task does not complete within the specified time, the inhibitor arc

that starts from p65 does not allow the scheduler to continue.

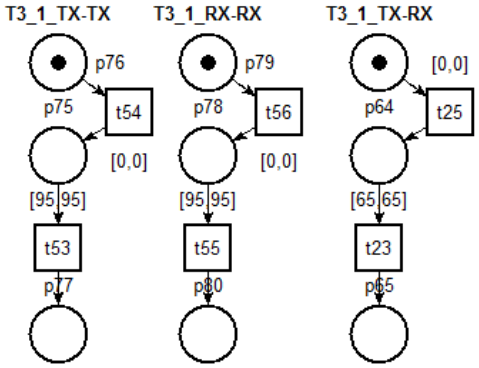

Figure 4: Diagram of block for the verification of constraints on task T_3.

- **Checking the Scheduled Time between Two Epochs**
  The network in Figure 5 monitors the execution time of a task between the two epochs. The transitions t14 and t17 are enabled when the task is run in both the first and the second period. This starts the timer of transition t2. If the task completes before the time set in the transition, the scheduling can continue. Otherwise, if the task does not complete within the specified time, the inhibitor arc that starts from p19 does not allow the scheduler to continue.

With this model and TINA tool we are able to understand if the hypothesized schedule is correct, and in fact if all checks are satisfied we can found a token in p54, otherwise the tool stops the simulation on the place that generated the error.

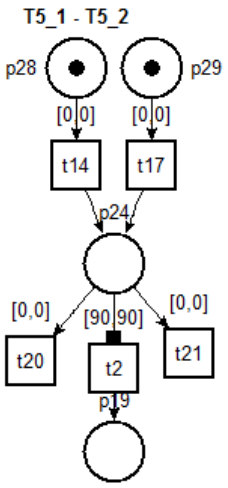

Figure 5: Check block for the scheduled time between two epoch for task T_5.

4.2 CPN

An ordinary PN has no types and no modules, only one kind of tokens and the net is flat. With CPNs it is possible, instead, to use data types and complex data manipulation. In fact each token has attached a data value called the *token colour*, which defines the range of values that the attributes can assume and the operations applicable in the same way of a variable type in any programming language. The types can be basic types or structured types, the latter defined by the user. The token colours can be investigated and modified by the occurring transitions.
With CPNs it is possible to build a hierarchical description and for this reason a large model can be obtained by combining a set of submodels.
In Figure 6 the model of the running example generated with CPN tools 4.0 (Jensen et al., 2007) is reported. As we can see the representation with CPN is more concise than the one seen with TPN, for example in one place we can represent all the tasks of the set. The tasks are represented as a list of objects and each one is represented by a token that have two attributes: a string that contains the name and one integer that represent the WCET C_i of the task.

The verification of the constraints on the execution time of the thread are realized through some functions listed on the transitions. On the first and second transitions named ”Put”, for example, we can find respectively the functions called *[verifyTh3 ()]* and *[verify2Th3 ()]*. These two functions implement the constraint that between the two executions of T_3 cannot elapse less than 65 milliseconds. The functions are defined as follows:

```
fun verifyTh3((n,t)::l) =
  if  n="Thread3" andalso
      intTime() > 65
  then false else true
fun verify2Th3(((n,t)::l)) =
  if  n="Thread3" andalso
      (intTime()-100) > 65
  then false else true
```

The function checks if the token in input to the transition represents the task 3, and verifies that the current simulation time (obtained with the function *intTime()*) is less than 65 units. If the constraint is not respected, the transition is not enabled.
On the transition ”End” we can find a function named *[verifyTime()]* that checks all the other constraints (the function is similar to the one above) except that the one on T_5 that is represented by function *[verifyTh5ctime()]* placed as guard on the same transition. The modeling of this last constraint, specific for task T_5, requires to save in a variable the timing at which

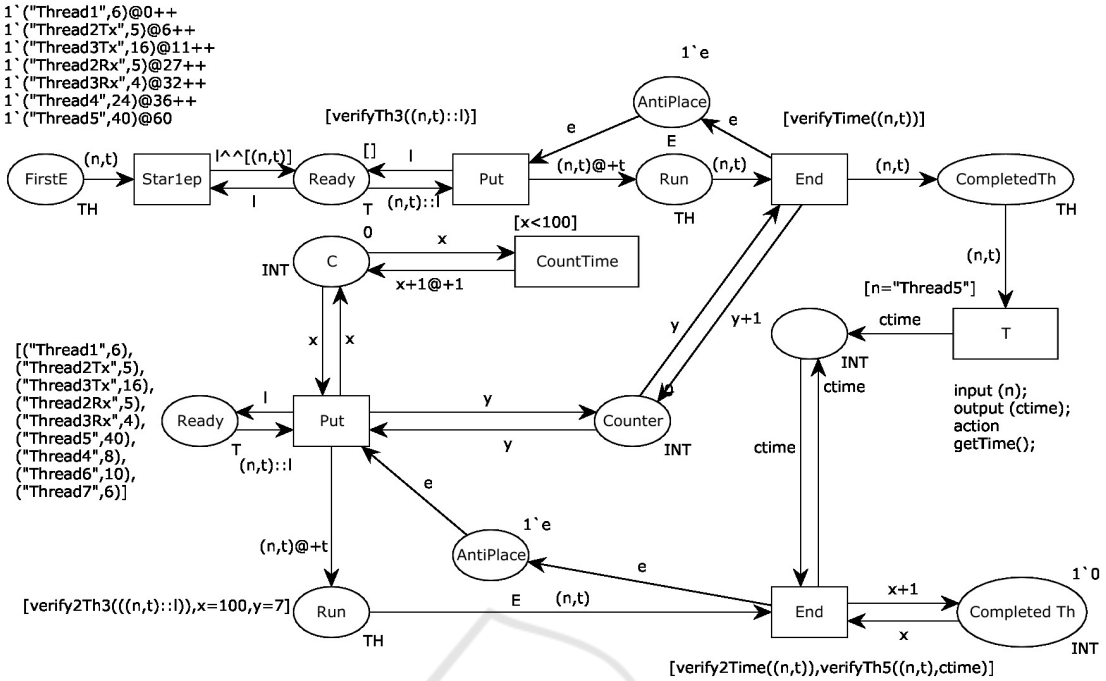

Figure 6: Coloured Petri Net model for a fixed scheduler.

the token of the T_5 exits from the "Run" place in the first epoch. This has been achieved through the transition "T" with the pattern *input, output, action* where we take a variable in input (variable n) and by the action (getTime() function) we generate an output (variable ctime). This transition is enabled only for T_5 as we can see from the guard on the arch. So the variable that we have obtained could be used in the function:

```
fun verifyTh5 ((n, t), ctime)=
  if  n= "Thread5" andalso
      (intTime () - ctime) >= 90
  {then false else true}
```

Similarly to the modeling done with TPN, also the simulation of the CPN model by means of the CPN Tools 4.0 stops if one of the constraint is not satisfied, so the user is able to understand where the problem is located.

4.3 Comparison between TPN and CPN

The experiments have allowed a comparison between the two Petri Nets dialects, in particular enlightening the following points:
The TPN model is difficult to read, and the addition of further tasks would result in a huge increase of the places and transitions number, making it more and more unreadable. This increase is due to the fact that:

- Any place can hold a single token and the execution of a thread must be reproduced a number of times equal to the number of modeled processes; indeed in TPNs it is not possible to express an attribute that differentiates the identity of a token.
- Time management for each thread is left to time constraints on the transitions themselves.
- It is not possible to create aggregate objects: a FIFO queue, for example, can be realized only through checks by inhibitors arcs with a number of places that depends on how many threads should be modeled (the number of places to represent the queue is equal to n^2 where n is the number of threads).

The CPNs instead can represent a queue using a single place that contains a token of type list. Time management is left to the token using an integer and a timestamp. This allows to represent a large number of tasks by simply adding tokens to the initial marking, leaving the structure of the model unaffected.
The time constraints can be grouped in auxiliary functions, thus simplifying size and readability of the model. In TPNs, instead, for each thread it is necessary to model a constraint through places and transitions.
With TINA and TPNs the control of time during simulation allows to easily understand the global state of the system. The time is increased at any time unit time. In CPNTools and CPNs if there are no transitions enabled at the current time, the simulated time count is increased in one step up to the time at which

at least one transition is activated.
CPNs have however resulted to be more advantageous in terms of time spent in model design or in changes, mainly for two reasons:

1. constraints can be simply modeled by a guard on the transition, expressed by a function written in pseudo code, which is easier to express;
2. to populate the model with new tasks it is not necessary to draw new graphic elements but just add an entry to the related place;

We have experienced that the time spent in CPN modeling is at the end more than half that spent in TPN modeling.

5 CONCLUSIONS

We have applied the two modeling options sketched above to different scheduling algorithms and different sets of tasks as well. The quite straightforward conclusion is that the CPN modeling is more advantageous in terms of size and readability of the model, and in terms of adaptability of the model to different task sets.
It is indeed easier with CPN to instantiate the same model, for the same scheduling algorithms, on a different set of tasks, and this is what is important in the daily application of this modeling framework. Since essentially only the characteristics of the taskset need to be changed for a new, or modified, specific application, the overall time to analyse a new taskset, summing up the time to produce a model of the schedule of a new specific application, to run a simulation and to analyse the simulation is about two hours with a TPN modeling and about one hour with the CPN modeling. Anyway, this time compares with the much longer time (eight hours) needed by the empirical approach previously used, and therefore is convenient in both cases.
Even if some rework is needed due to a negative response of the simulation, the information returned by the simulation helps understanding where the problem lies, indicating the solution to the problem. Usually one rework cycle is at most needed, so the overall cost is anyway reduced.

For this reason we have not considered convenient to investigate solutions based on counterexample generated by a model checker (Guillaume Gardey, 2005), able to provide automatically the taskset parameters satisfying the scheduling requirements.

The low cost of the simulation based solution has an obvious positive impact on the costs of the process of instantiating a generic application to a new specific application for marketing a new product or variant.
This process is currently under experimentation in our partner company, with the aim of introducing it in the routine customization process. An help for this introduction could come from providing tools to support an easier instantiation of the generic models into specific ones, so that the use of CPN is transparent to the final user who only sees the simulation results. This objective requires also a facility to explain the reasons of a negative response without showing the underlying CPN model. This is considered as future work.

ACKNOWLEDGEMENTS

We wish to thank Marco Bartolozzi, Daniele Marchetti and Luca Santi for their contribution to the conducted modeling experiments.

REFERENCES

van der Aalst, W. M. P. (1996). Petri Net based scheduling. In Operations-Research-Spektrum vol.18 (219-229). Springer-Verlag.

Alur, Rajeev and Dill, David (1992). The theory of timed automata. In 17th Real-Time: Theory in Practice, Lecture Notes in Computer Science vol.600 (45-73). Springer.

Barreto, R., Cavalcante, S., and Maciel, P. (2004). A time Petri Net approach for finding preruntime schedules in embedded hard real-time systems. In Distributed Computing Systems Workshops (846-851), 2004. Proceedings. IEEE.

Berthomieu, B. and Vernadat, F. (2006). Time Petri Nets analysis with TINA. In Quantitative Evaluation of Systems (123-124), 2006. IEEE.

Berthomieu, B., Diaz, M. (1991) Modeling and Verification of Time dependent system using Time Petri Nets. IEEE Transactions on Software Engineering, vol. 17 , no. 3 (259-273). IEEE.

Buttazzo, G. (2011). Hard Real-Time Computing System. Springer, New York, 3rd edition.

Cenelec (1996). Cenelec en 50128. In Railway Applications - Communications, Signalling and Processing Systems - Software for Railway Control and Protection Systems.

CPNTools (2015). <http://http://cpntools.org/>.

Fantechi, A. (2013) Twenty-Five Years of Formal Methods and Railways: What Next? SEFM Workshops 2013, Lecture Notes in Computer Science vol.8368 (167-183), Springer.

Felder, M., Mandrioli, D., Morzenti, A. (1994) Proving Properties of Real-Time Systems through Logical Specifications and Petri Net Models, IEEE Transactions on Software Engineering, vol.20, no.2 (127-141)

Grolleau, E., Choquet-Geniet, A. (2002). Off-line computation of real-time schedules using Petri Nets. In Discrete Event Dynamic Systems, vol.12, no.3 (311-333). Springer.

Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier (H.) Roux (2005). Romo: A tool for analyzing time Petri nets. In 17th International Conference on Computer Aided Verification (CAV05), Lecture Notes in Computer Science vol.3576 (418-423), Edinburgh, Scotland. Springer.

Jensen, K. (1987). Coloured Petri Nets. In Petri Nets: Central Models and Their Properties. Lecture Notes in Computer Science, vol.254 (248-299), Springer.

Jensen, K., Kristensen, L. M., and Wells, L. (2007). Coloured Petri Nets and CPN tools for modelling and validation of concurrent systems. In International Journal on Software Tools for Technology Transfer, vol.9, no.3 (213-254) Springer-Verlag.

Leveson, N. G., Stolzy, J. L. (1987) Safety Analysis using Petri Nets. IEEE Transactions on Software Engineering, vol.13, no.3

Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. In Journal of the ACM, vol.20, no.1 (46-61). ACM.

Murata, T. (1989). Petri Nets: Properties, analysis and applications. In Proceedings of the IEEE, vol.77, no.4 (541-580). IEEE.

Peterson, J. L. (1981). Petri Net Theory and the Modeling of Systems. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Petri, C. A. (1962). Kommunikation mit automaten. In PhD thesis. Universitat Hamburg.

Ramchandani, C. (1974). Analysis of asynchronous concurrent systems by Timed Petri Nets. Massachusetts Institute of Technology.

Stankovic, J. (1988). Misconceptions About Real-Time Computing. IEEE Computer, vol.21 (10-19). IEEE.

TINA (2015). <http://http://http://projects.laas.fr/tina//>.

Tsai, J., Jennhwa Yang, S., and Chang, Y.-H. (1995). Timing constraint Petri Nets and their application to schedulability analysis of real-time system specifications. In IEEE Transactions on Software Engineering, vol. 21, no. 1 (32-49). IEEE.