A Petri Net Extension for Schedulability Analysis of Real Time Embedded Systems

Yessine Hadj Kacem, Walid Karamti , Adel Mahfoudhi, and Mohamed Abid
Computer Embedded Systems (CES),University of Sfax
National School of Engineers (ENIS), BP 1173, Sfax, 3038, Tunisia

Abstract— *Petri Net can be used to analyze and verify real time systems. However, scheduling problems are not totally covered using this formalism. Traditionally, the verification of these systems is limited to the performance and behaviour analyses.*
This paper presents a scheduling analysis method for real time systems at an early stage. The proposed formalism called Priority Timed Petri Net (PTPN) extends Time Petri Nets in order to find a feasible schedule that satisfies timing constraints. Computational and analysis model were given while explaining the proposed model-checking.

Keywords: real time systems, verification, Time Petri Nets, scheduling analysis, PTPN

1. Introduction

Real time systems (RTS) are time constrained systems that are in a continuous interaction with their external environment. Such systems have become increasingly complex, especially with their distributed aspects (parallel operation on various machines), where the need for using reliable techniques of scheduling emerges. Since function or bug may cause economic and human catastrophes. Thus, the anomaly absence at simulation stage cannot confirm the non-appearance of bugs. So, schedulability [21] is a key challenge in the analysis of distributed RTS.
Therefore, to protect such systems from problems and failure, it is necessary to implement formal techniques intended to make reliable the development process of the real-time applications, from their design to checking. This allows designers to accurately validate systems, and check the required properties of their behaviour.
The specification and formal verification techniques could reduce the problem impact. They aim at being a reliable approach and they constitute an important and recent field of research. The objective of the formal specification is starting from an abstract modeling to express the system properties by taking into account a set of constraints and therefore, to check that the system specification satisfies these properties. Generally, the choice of the suitable formal method depends on the characteristics of the system and the properties to be checked. We distinguish two classes from prove methods: deductive methods and models-based methods.
Deductive methods consist in interpreting the problem of verification like a theorem; its major limitation is that the user interacting with the prover must be able to guide him. For example, the method B proposed in [15] is used in order to validate opted models. The authors start with an abstract specification from the scheduler and then use successive refinments to account for the time characteristics of the automats. They check the hierarchy of refinements by proving the various generated obligations. Formal model-based methods describe system behaviour and explore different system states. These techniques are Simulation, feasibility test and Model checking. Simulation is based on the simulation of the task progress during one system period, wich assures that all the authorities of tasks respect their deadline. It can treat scheduling policies or task models which are difficult to analyze mathematically. Calculating response times applies a recurrence formula which calculates the deadlines of execution on an interval that contains the greatest response times of the considered task which is met. Feasibility test which applies a formula decides the feasibility of task scheduling according to which characteristics appear to be the simplest method to implement with a low calculation complexity. Contrary to the simulation and test of feasibility, the model-checking provides examples showing why an imposed constraint is not satisfied. Petri Nets (PN) presents an adequate model-checker due to its greatly expressive dynamic vision and its executable aspect. However, the application of Petri Nets to a scheduling problem is far from trivial; determinism aspect is tackled neither in regular Petri Net, nor in its extension for RTS verification. There are many recent research efforts on real time systems verification using PN. The adoption of model checking by means of PN extensions has been vastly investigated.
In [18], the authors use T-Timed stochastic Petri Nets to model real time tasks. The authors stress the independent periodic tasks in order to support Rate Monotonic strategy [12]. Priorities are integrated using the inhibitor arcs which connect the waiting places of each task with transitions that call the processor modeled by a place. The temporal faults, considered as a task behavior, are presented in the Petri Nets task model. However, if the tasks to be modeled are too complex then the size of the model will be enormous and the added inhibitor arcs face difficulties to interpret the graph.
Roux [19] presented the Scheduling Timed Petri Nets (STPN) to analyze the schedulability of the independent

tasks on a multiprocessor architecture. Each processor is modeled by a place; thus the concept of priority is introduced by the inhibitor arcs. The motivation behind this approach was mainly to propose calculation of a state space that is more reduced than the traditional state graph suggested by [2] and the difference bound matrix. The authors extend their work in [10] to support the tasks with variable time execution. Another improvement [11] consists in extending the STPN to manage the dynamic priorities and support variant scheduling policies.

Berthomieu, in the same line, introduced the notion of priorities within the temporal Petri nets framework [3] via the PrTPN (Priority Time petri Nets) extension. His proposal is based on the following constraint: a transition will be fired only when it has the highest priority compared to the concurrent and enabled transitions. To introduce the priorities, the author also uses inhibiting arcs as a new component which connects the simultaneous transitions. The problem arises when the network is of an enormous size and several transitions are dependent. The model will then be difficult to be presented and interpreted.

As a notable model checker, the Extended Place-timed Petri Nets (EPdPN)[5] is worth mentioning. EPdPN is based on P-Timed Petri Net and introduces two priorities to solve the problem of transitions' conflict. Nevertheless, the contribution of authors does not take into account the periodicity of real time tasks.

This paper presents a Time Petri Net based approach for systems scheduling analysis, considering periodicity, fixed priority, pre-emption, multiprocessor architecture and inter-task relations (e.g., precedence and mutual exclusion). The proposed Priority Time Petri Net gives determinism aspect to the model and to accelerate its execution. Besides, the developed tool of our support infrastructure is founded on the Eclipse platform, in the form of a plug-in capable to exchange data with Eclipse Features.

The paper is organized as follows: the description of the proposed Priority Time Petri Net is given in the next session highlighting formal definitions. Section 3 explains the extended Petri Nets building blocks for the schedulability analysis. Section 4 introduces the supporting tool and depicts an experiment. Finally, the conclusion and future research directions will be discussed.

2. Computational and Analysis Models

Petri nets [16] is a mathematical formalism, which allows the specification of the behaviour of real time interactive systems. Concurrent real time concepts such as timing constraints, shared resource and synchronisation are taken into account through two main Petri Nets extensions: Time Petri Nets [13] and Timed Petri Nets [17]. In what follows some definitions on Petri Nets are presented. Next, some basic concepts on regualr PN are briefly recalled. Then, the extension of the proposed formalism is stressed. Finally, the model analysis of the extended PN is described.

2.1 Background for Petri Nets and Time Petri Nets

Definition 1

A Petri Net is a 4-tuple [14], R = { P, T, B, F}, where:
(1) P = {p_1, p_2, ..., p_n} is a finite set of places $n > 0$;
(2) T = {t_1, t_2, ..., t_m} is a finite set of transition $m > 0$;
(3) $B:\ (P \times T) \mapsto \mathbb{N}$ is the backward incidence function;
(4) $F:\ (P \times T) \mapsto \mathbb{N}$ is the forward incidence function;
Each system state is represented by a marking M of the net, it is defined by : $M : P \mapsto \mathbb{N}$

Definition 2

A marked Petri Net is a couple $PN =< R, M_0 >$, where R is a Petri Net and M_0 is the initial marking.

Definition 3

A vector Ft_s is associated with each marking M which presents the fired transitions.
Ft_s is defined by: $FT_s:\ T \rightarrow \begin{cases} 1 \\ 0 \end{cases}$
$\forall t \in T$ and $B(P,t) \leq M \Leftrightarrow t \in FT_s \Leftrightarrow FT_s(t) = 1$

Definition 4

The firing of a transition is described by :
Let M be a marking, $\forall t \in FT_s$, then the firing of the transition t is said:
$M' = M + F(P,t) - B(P,t)$
In the rest of this paper, we use the following notation : $M \xrightarrow{t} M'$

Definition 5

M' is reachable from a marking M iff: $\exists t \in T,\ M \xrightarrow{t} M'$
Thanks to its power of expressivities, the regular PN make it possible to describe parallelism, dependency and semaphores, which present important properties of RTS. In spite of this significant ability, regular PNs are not able to model the temporal evolution of RTS. That is why the regular PN is extended to Time Petri Net.

Definition 6

Time Petri Net associate a static time interval with the transition firing. A Time Petri Net is a 3-tuple TPN:
TPN = { PN, T_{min}, T_{max}}, where:
(1) PN is a regular Petri Net;
(2) $T_{min}\ : T \mapsto \mathbb{Q}^+\ ,\ T_{min}(t_i)$ is the earliest firing time mapping;
(3) $T_{max}\ : T \mapsto \mathbb{Q}^+ \cup \{\infty\},\ T_{max}(t_i)$ is the latest firing time mapping;
A global clock is coupled to the system.

Definition 7

Each transition in a TPN is accompanied by a local clock. All local clocks are grouped in a vector called (Hl), with $Hl\ : T \mapsto \mathbb{Q}^+$.
The clock transition is activated as soon as the transition is enabled and remains activated until it will be valid.

Definition 8

A transition is valid when it is firable and the local clock has met the time interval:
t is valid $\Leftrightarrow t \in FT_s \wedge Hl(t) \in [T_{min}, T_{max}]$

Definition 9

Only valid transitions will be fired. To simplify their selections, the function Ft_s is filtered and the temporal filter is described by: t is not valid $\Longrightarrow \begin{cases} FT_s(t) \leftarrow 0 \\ Set\ Increment\ Hl(t) \end{cases}$
Hence, the marking after filtring is: $M' = M + F(P,t) - B(P,t) \Longrightarrow t \in FT_s$

2.2 Discussion about the indeterminism

As already mentioned, regular PNs are not able to support the temporal evolution of RTS. Among the multitude of the existing extensions of Petri Nets for RTS modeling, we can cite:

- Timed Petri Nets in which time can be assigned to places or transitions. In fact, P-Timed Petri Nets and T-Timed Petri are two subclasses of this PN extension. While firing a transition with a duration d_i in Transition Timed Petri Nets, the required tokens are removed form the input place at an indeterminate date and they are added to the output place after the duration d_i. Timed Petri Nets can lead to a well performance analysis method but could not cover the scheduling problem wich require determinism aspect.
- Previously presented Time Petri Nets: They offer a suitable solution able to describe the different states of a Task and their related events.

While the Time Petri Nets is quite accurate for the task behavior characterisation, there are some issues in the cases of the RTS task with fixed priorities. It is worth noting that the Time Petri Nets lead to a good presentation of all system tasks and events, but they intercalate a firing interval of the transitions which presents an event. This could bring about the passage of a task state towards another during an undeterminable date. Indeed, in our study, tasks with the highest priority are executed. However, a Time Petri Net handles the tasks in an equitable way (no transition priority) that causes the problem of transition conflict. The states graph of all the nodes does not obey to a strategy of real time scheduling. Thus if we attribute the notion of priority to transitions, the new formalization of Time Petri Nets can go beyond the indeterminism problem. In particular, T_{min} is equal to T_{max} to avoid indeterminate time.

2.3 Priority Time Petri Net: PTPN

Definition 10

PTPN is a 3-tuple defined by $PTPN =< R, T_f, P_r >$ where:
(1) R is a regular Petri Net
(2) $T_f : T \mapsto \mathbb{Q}^+$ is the firing date of a transition
(3) $P_r : (T \times P) \mapsto \mathbb{N}$ is the priority of a transition according to a place

The relation between P_r and B is defined as:
$\forall t \in T \wedge \forall p \in P$, if $B(p,t) = 0 \Leftrightarrow P_r(t,p) = 0$
Let $p \in P$ and the set $T_p = \{\forall t \in T \backslash B(p,t) > 0\}$
For a concrete presentation of B, we consider it as a matrix of $|T|$ lines and $|P|$ columns,

$$P_r = \begin{bmatrix} P_r(t_0,p_0) & P_r(t_0,p_n) \\ & \\ P_r(t_m,p_0) & P_r(t_m,p_n) \end{bmatrix}$$

$\forall p \in P$ and $\forall t \in T,\ P_r[p,t] \rightarrow \begin{cases} \mathbb{N}^*,\ if\ t \in T_p \\ 0 \end{cases}$

Definition 11

The marked PTPN is the couple $MPTPN =< PTPN, M_0 >$:
(1) RTP is a PTPN network
(2) M_o is the initial marking of the net
PTPN is an extension of Time Petri Nets; so the definitions of local and global clock and firable transitions Ft_s are conserved.

Definition 12

A transition is valid if it is firable and it respects it firing date.
$\forall t \in T\ is\ valid \Longleftrightarrow t \in\ FT_s \wedge Hl(t) = T_f(t)$
The marking strategy is based on priorities: the transition having the highest priority will be fired. When such policy is applied to a valid Ft_s vector which presents only independent transitions.

Definition 13

Two transitions are independent if the firing of the first does not influence that of the second.To accelerate the firing of transitions with PTPN, we propose a method which makes it possible to fire a vector of independent transitions.

2.4 Marking evolution in PTPN

Definition 14

To fire a marking, a PTPN Firing Machine (PFM) is offered by our extended formalism. The mechanism is described in Figure 1. The PFM input is a PTPN marking. For each entry, the machine triggers four events considered as jobs: During the first job, PFM determines the vector of firable transitions Ft_s which presents the input of the second job The second job consists in isolating non-valid transitions from Ft_s which respect firing time.

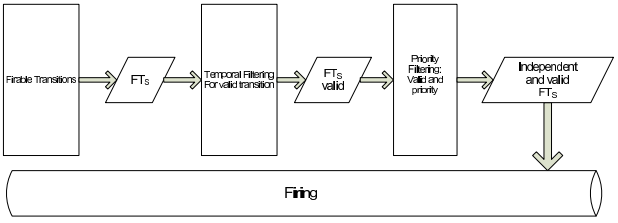

Figure 1: PTPN Firing Machine

Definition 15

Ft_s temporal filtering is defined as:
$\forall t \in Ft_s \; and \; Hl(t_i) \neq T_f(i) \Longrightarrow Ft_s(ti) \leftarrow 0$
After this filtering, we can say:
$\forall t \in Ft_s \Longleftrightarrow t$ is valid.
The objective of the third job is to ensure that the vector contains only independent transitions. So, PFM applies a priority filtering.

Definition 16

Filtering must take into account all places, if two transitions are in competition of place sharing, then the transition which has the highest priority will be fired. In fact, the product of the Ft_s vector and the matrix Pr is calculated in order to select a concurrent and valid transition for each place; it is saved in the vector $Prod$ as follows:

$$Prod: \; P \mapsto Ft_s \times Pr$$

$$Prod(P_i) = \begin{bmatrix} Ft_s(t_0) \\ - - - \\ \\ \\ - - - \\ Ft_s(t_m) \end{bmatrix} \times \begin{bmatrix} Pr(t_0, p_i) \\ - - - \\ \\ \\ - - - \\ Pr(t_m, p_i) \end{bmatrix}, \text{ where } i \in [0, n]$$

Next, the transition having the highest priority per palce is selected from the vector $Prod(p_i)$.

$$\begin{bmatrix} Ft_s(t_0) \times Pr(t_0, p_i) \\ - - - - - - \\ \\ \\ - - - - - - \\ Ft_s(t_m) \times Pr(t_m, p_i) \end{bmatrix} \overset{Max()}{\rightarrow} [Prod(p_i)[t_j]], \text{ where:}$$

- $i \in [0, n]$
- $j \in [0, m]$
- Max is a function that returns the transition with the highest priority

Now, each vector $Prod()$ of each place is transformed into a binary vector through the following function:
For $i \in [0, n]$

$$Bin :\rightarrow \begin{cases} \forall t_j \in Ft_s \backslash Max(Prod(p_i)) \Longrightarrow Prod(p_i)[t_j] \leftarrow 1 \\ \forall t_j \in Ft_s \wedge (t_j = Max(Prod(p_i))) \Longrightarrow Prod(p_i)[t_j] \leftarrow 0 \end{cases}$$

In that case, the vector Ft_s is updated by subtracting the vector Pr from each place p_i of the Ft_s vector:
$Ft_s \leftarrow Ft_s - Prod(p_i)$

$$Ft_s \leftarrow \begin{bmatrix} Ft_s(t_0) \\ - - - \\ \\ \\ - - - \\ Ft_s(t_m) \end{bmatrix} - \begin{bmatrix} Prod(p_i)[t_0] \\ - - - \\ \\ \\ - - - \\ Prod(p_i)[t_m]] \end{bmatrix}$$

Definition 17

The firing of Ft_s is defined as:
$M' = M + \sum_{t \in Ft_s} (F(P,t) - B(P,t))$
The PFM accelerates the PTPN evolution by firing the whole vector since all Ft_s transitions are independent.
We note $M \overset{Ft_s}{\rightarrow} M'$, where M is an entered marking and M' is the ouput marking. To simplify notations, M' is said to be accessible via M through Ft_s.

Algorithm 1 State graph construction

```
1: for each clock tic do
2:   for each marking do
3:     if firable transition exists then
4:       if valid transition exists then
5:         state construction and New marking
6:       else
7:         Set increment clock
8:       end if
9:     end if
10:  end for
11: end for
```

2.5 PTPN states graph

In PTPN, a state is a node composed of (M, tmp), where M is a marking and tmp is the time for firing the marking M. The graph of states is a set of nodes connected with arcs having Ft_s weight. The connection arcs will be established only when two markings of the nodes are reachable via a Ft_s vector.

The construction of PTPN states graph corresponds to the execution algorithm shown in Algorithm 1. This process starts with the creation of an initial state$(M_0, 0)$. Then, for each clock tic, an iterative process is triggered. A marking is sent to the PFM. Next, the machine returns a new marking that allows the creation of a new node (M, tmp) with an arc having the weight of the Ft_s vector. The new generated marking presents the new entry point of the PFM. This iteration is continued while there are firable and valid transitions.

3. Model Construction

This section depicts how PTPN is used to model real time tasks and particularly their periodicity, priority, dependency and distribution on distributed architecture.

3.1 Task creation and activation with PTPN

We will use the net shown in Figure 2 to illustrate the creation and activation of task with PTPN. There are six places ("Uncreated", "Period", "Created", "Disabled", "Ready" and "Activated") and three transitions ("Creation", "TPeriod" and "Activation"). The vector T_f is defined by (R_i, P_i, 0). In the state shown in Figure 2, there are two tokens; one in the "Uncreated" place and another in the "Disabled" place. The tokens in the place "Uncreated" represents an uncreated Task which is ready to be created at the time R_i. The token in the "Disabled" place indicates that the task is disabled. If the task is activating a job, then there are no tokens in "Disabled" place and there is one token in the "Activated" one. At time 0, the "Creating" transition is enabled and the firing time is R_i. So at the time R_i, it will be validated. The firing consumes the token from "Uncreated" and produces two tokens: one is for "Period" and another for "Created". Now the "Activation" transition is enabled, the firing time is 0 and consequently it is validated. "Activation" fires at time

"Ri" and consumes one token from "Disabled" and another from "Created". The firing produces one token in "Ready" and one in "Activation".

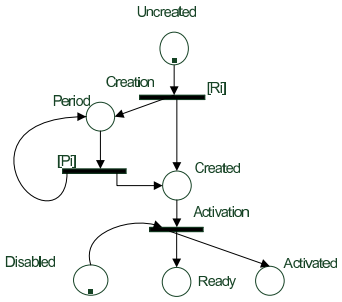

Figure 2: Creation and activation of a task with PTPN

3.2 Dependency between tasks

The dependency modeling is based on task graph which describes the precedence between tasks. Let T_i be a producer task of the consumer job T_j, T_h. T_j and T_h depend on T_i; so T_i is called the predecessor of T_j and T_h (the successors). The execution of the task T_i has to be completed before the execution of the task T_j and T_h may start. Figure 3 depicts how the relation between tasks is presented. There is a "successor" place for each task to describe the end of execution. When "successor" is marked, the transition "Send" is valid. It is attached only to predecessor tasks. In Figure 3, "Sends" is linked to the place "Successor" of T_i. The firing produces two tokens: one for the place $T_i 2 T_j$ and another one for the place $T_i 2 T_h$. Now, both of T_j and T_h can be activated and complete its execution.
It should be noted that priorities are attributed according to Rate Monotonic Policy. Regarding dependent tasks, the predecessor task has a higher priority than the successor one according to rectified Rate Monotonic policy.

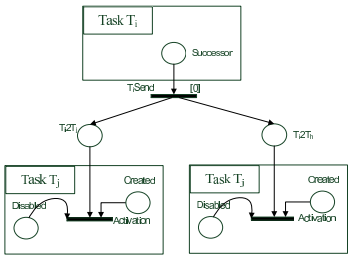

Figure 3: Representation of tasks dependency with PTPN

3.3 Task Execution

According to the adapted organization strategy, a task can occupy the free processor only when it has the highest priority. The event of activity allows the passage of a task towards the execution state as well as that of the processor towards the busy state.
Indeed, the modeling of this event with PTPN, consists in creating two entry arcs towards the "GetProc" activity transition. The first arc arises from the "Ready" place and the second comes from the "Processor" place which is shared by the various "GetProc" transitions of the tasks. The outputting arcs of this place have to carry the priorities associated with the suitable tasks. When a token appears in the "Ready" place, the task is set to occupy the processor. The associated "GetProc" transition is valid only when each one of the "Ready" and "Processor" places presents a token. The firing allows the consumption of both tokens and the production of the other one in the "inExecution" place. The new marking describes the activity of the processor of the current task. Of course, we are interested in premptive systems, whose notion is managed as follows: each task occupies the processor for a single unit of time. If the task remains the most primary, then it occupies it again, other wise it will be preempted. The modeling with PTPN inserts similarly a firing date (1) on the "incrementing StopWatch" transition. The firing allows incrementing the chronometers which calculate the time of the processor use per task. Each task has two stopwatches: the first one to compute the 'Ci' units and the second to count the 'du' units. They are modelled by two places "StopWatchCi" and "StopWatchDu". For any "inExecution" firing, there is a production of tokens in both stopwatch places. As soon as the "StopWatchCi" place indicates the presence of Ci tokens, the transition "endCi" is validated. Its firing allows the liberation of the processor as well as the blocking of the task up to the new wake. The transition "endDu" is valid when the "StopWatchDu" place indicates the presence of Du tokens. The firing allows the liberation of the processor and the deactivation of the task (the task did not appear in the line of the processor). In particular cases where Du is a multiple of Ci, the transitions "endCi" and "endDu" are valid. The task has to pass towards the "Terminated" state to indicate that it is well organized, then the transition "endDu" is more principal than "endCi". To integrate both priorities associated with these two transitions with PTPN, we need a shared place between both transitions. Figure 4 presents a "Maker" place which plays the role of a judge for both events. "Maker" allows the firing of a "ReleaseProc" transition responsible for the liberation of the processor for the pre-emption.
An important issue in real time system verification is to check if each job has met its deadline. Therefore, we identify two verification levels: the periodicity and the deadline. As shown in Figure 5, there are four places ("Period", "Created", "Activated", "Deadline"), two transitions ("Tperiod", "Tdeadline") and $Tf(P_i, 0)$. The transition "Tperiod" is valid for each P_i time unit. The firing consumes the token from the place "Period" and produces two others: One in the place "Created" and the other in the place "Period". This means that the task is ready to wake up and the new period

is loaded. If the task is already activated, the transition "Tdeadline" is valid. The firing of the transition "Tdeadline" provides a token to the place "Deadline" and disables the task execution.

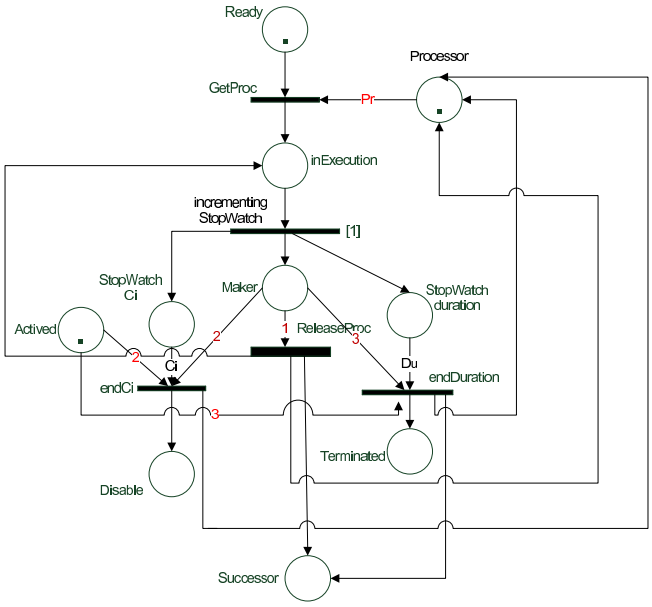

Figure 4: Executing a Task with PTPN

According to the described models presented previously, the scheduling analysis of a real-time system modelled with PTPN produces a model whose size is very important. The great complexity of the RTS makes it difficult to understand and manage. In order to solve the problems of organisation and interpretation of the models described in PTPN, we resort to the structuring method of the Petri nets in the next session.

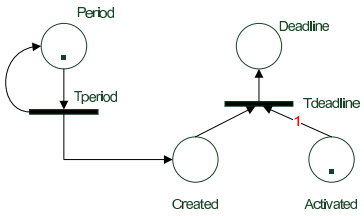

Figure 5: Meeting deadlines with PTPN

4. Tool and Experiment

This section introduces PPTNS (Priority Time Petri Nets for Scheduling analysis), the tool that we have implemented to concretize our proposed formalism. The implementation uses the Graphical Modeling Framework (GMF). We subsequently present an example that illustrates how PPTNS is used to test the schedulability of a set of tasks.

4.1 Tool

The implemented tool takes the form of a Petri Net editor and a simulator of the modelled net. The developed editor for our PTPN relies on the GMF founded on Eclipse Modeling Framework (EMF). The definition of the PTPN Meta Model represents the starting point of the editor's generation process. As shown in Figure 6, the regular Petri Nets Meta Model is extended with the attribute $priority$ linked to the $InputArc$ entity. Thus, the production of an editor plug-in allows the interactive edition of the PTPN (create drag, drop, grab or delete a component). The created models are checked through a set of constraints expressed with the Object Constraint Language (OCL) [7]. The validation doubles through the verification at times and after constructing the model.

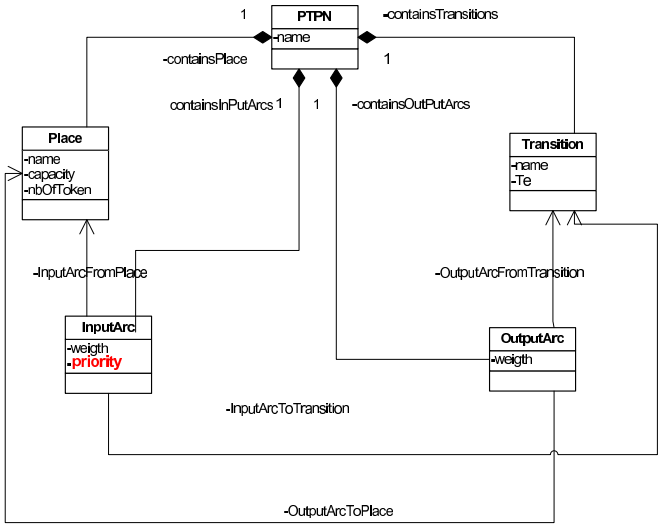

Figure 6: PTPN Meta Model

It is obvious that the created model is built around a drawing composed of places, transitions and arcs. In fact, we need to easily extract the existing data from the editor. Fortunately, the created model can also be serialized to generate an XML (Extensible Markup Language) or XMI (XML Metadata Interchange) file. The generated file conforms to the PTPN Meta model and presents the entry port point of the Simulator. Due to the structure of the editor output, the properties of the modelled net are easily interpreted.
The verification framework is sufficiently flexible and expressive to support module inclusion and extension. The use of the editor tool makes it easier and faster to create PTPN models. Compared to existing Time Petri Nets simulators such as ROMEO [6] and TINA [4], the impetus of our tool is its structured input and output files which guarantees interaction with the existing PN simulators and Eclipse features.

Table 1: Experiment for scheduling anlysis

Id_Task	R_i	C_i	deadline	Period	Duration	Priority	Partition
1	1	1	3	3	3	6	P1
2	0	2	5	5	4	5	P1
3	0	2	6	6	2	4	P2
4	0	2	5	5	4	2	P2

Table 2: Simulation Results

Step	Time	Ft_s	Step	Time	Ft_s
0	0	Creating t_2 , t_3, t_4	10	3	incrementingStopwatch t_2 t_3
1	0	Activating t_2	11	3	end$C_i t_2$, ReleaseProc t_3
2	0	Executing t_2	12	3	Executingt_3
3	1	Creating t_1, incrementingStopwatch t_2	13	4	incrementingStopwatch t_3, period t_1
4	1	Activating t_1, ReleaseProc t_2	14	4	end$C_i t_3$,Activating t_1
5	1	Executing t_1	15	4	Executingt_1, Activating t_4
6	2	incrementingStopwatcht_1	16	4	Executingt_4
7	2	end$C_i t_1$	17	5	incrementingStopwatch t_1,t_4, period t_2,t_4
8	2	Executing t_2 , Activating t_3	18	5	end$C_i t_1$, Deadline t_4
9	2	Executing t_3	19	5	

If we are to situate our extension with regard to the existing tools, we note the following distinctions:

- Contrary to Cheddar tools [22], Mast [9], Times [1], which cannot cover all the possible states of the system, PTPN starts from an initial state to succeed in determining the error source if it occurs.
- Pertaining to other extensions presented in Section 2, PTPN offers a strategy which accelerates the marking and avoids the combinatorial explosion in front of a large number of states.

4.2 Case study

In this section, we show how PTPN can be used to deal with the problems of schedulability analysis of real time system, with reference to the case of four tasks running on two processors. It should be noted that due to space limtation, we present a simple experiment. The dependency between tasks in this experiment is shown in Figure 7. The main characteristics of the taken tasks are illustrated in Table 1. We use the following acronyms to simplify notations:

- Id_Task is the identifier of a task
- R_i is the date of the first activation
- C_i is the execution time of a task
- P_i is a processor

After creating the model with PTPN editor, the result of the simulation is shown in Table 2. Each step describes a node of the previously explained state graph. The column time presents different overall clock ticks. The simulation begins at the instant 0, the initial marking vector M_0 is the input of the PFM. This machine is responsible for determining the valid and the highest priority transitions in the column Ft_s. For the marking M_0, PFM fills the Ft_s vector with the event "Creating" for the three tasks : t_2, t_3, t_3.

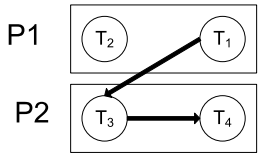

Figure 7: Tasks dependency in the experiment

The "Step 17" describes the presence of transitions "Period t_4" and "increntingStopwatch t_4". After firing them and during the step 18, the PFM detects two new transitions: "deadline t_4" and "ReleaseProc t_4", as shown in Table 2. The new deadline is activated when the task has not completed its execution. The firing of "deadline t_4" interrupts the simulation. The simulator PTPN confirms that the system is not schedulable because t_4 does not meet its deadline. This result can help the designer not only to run t_4 on another computing resource but also to avoid the new partitioning of all the other tasks.

5. Conclusion

Priority Time Petri Nets supports the scheduling analysis of real-time systems running on multiple processors, including periodic, dependant, and pre-emptive tasks with determinist strategy. The salient trait of novelty in the semantics of PTPN consists in the attribution of a priority

for transitions to isolate conflicts. Rather than presenting a solution for the problems of confusion, PTPN Firing Machine accelerates the PTPN evolution by applying temporal and priority filtering. In regular PN, the firing of a transition requires identifying the new vector of firable transitions. However, with PFM, this vector is established only after the firing of old one. It is also guided by the priorities. Consequently, starting from a marking M, the PFM fires simultaneously valid transitions having highest priority and returns the new marking M'.

In future research, we are interested in including performance analysis in the verification of RTS and planning to integrate PTPN design patterns ranging building blocs for an easy model construction and interpretation. Indeed, we intend to incorporate PTPN in a Model Driven Engineering (MDE) process [20]. The transformation from annotated analysis models to a formal model allows the validation specific properties. In particular, we are interested in translating Unified Modeling Language (UML) diagrams annotated with the profile Modeling and Analysis of Real-Time Embedded systems (MARTE) [8] into PTPN model to analyse system schedulability. The SPTPN simulator taking the form of a plug-in could easily exchange data with UML Eclipse editors.

References

[1] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times - a tool for modelling and implementation of embedded systems. In *TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 460–464, London, UK, 2002. Springer-Verlag.

[2] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. Softw. Eng.*, 17(3):259–273, 1991.

[3] B. Berthomieu, F. Peres, and F. Vernadat. Bridging the gap between timed automata and bounded time petri nets. In *FORMATS*, pages 82–97, 2006.

[4] B. Berthomieu and F. Vernadat. Time petri nets analysis with tina. In *QEST*, pages 123–124, 2006.

[5] L. Chen, Z. Shao, G. Fan, and H. Ma. A petri net based method for analyzing schedulability of distributed real-time embedded systems. *Journal of Computers*, 3(12), 2008.

[6] G. Gardey, D. Lime, M. Magnin, and O. H. Roux. Romeo: A tool for analyzing time petri nets. In *CAV*, pages 418–423, 2005.

[7] O. M. Group. UML 2.0 OCL Specification. OMG Adopted Specification ptc/03-10-14. Object Management Group, October 2003.

[8] O. O. M. Group. A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2, ptc/2008-06-09. Object Management Group, June 2008.

[9] M. G. Harbour, J. J. G. Garciia, J. C. P. Gutierrez, and J. M. D. Moyano. Mast: Modeling and analysis suite for real time applications. *Real-Time Systems, Euromicro Conference on*, 0:0125, 2001.

[10] D. Lime and O. H. . Roux. A translation based method for the timed analysis of scheduling extended time petri nets. In *RTSS '04: Proceedings of the 25th IEEE International Real-Time Systems Symposium*, pages 187–196, Washington, DC, USA, 2004. IEEE Computer Society.

[11] D. Lime and O. H. Roux. Formal verification of real-time systems with preemptive scheduling. *Real-Time Syst.*, 41(2):118–151, 2009.

[12] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[13] P. M. Merlin. *A Study of the Recoverability of Computing Systems*. Irvine: Univ. California, PhD Thesis, 1974. available from Ann Arbor: Univ Microfilms, No. 75–11026.

[14] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[15] O. Nasr, J.-P. Bodeveix, M. Filali, and M. Rached. Verification of a scheduler in b through a timed automata specification. In *Annual ACM Symposium on Applied Computing (SAC), Dijon, 23/04/06-27/04/06*, pages 1800–1801, http://www.acm.org/, avril 2006. ACM.

[16] C. A. Petri. Fundamentals of a theory of asynchronous information flow. In *IFIP Congress*, pages 386–390, 1962.

[17] C. Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. Technical report, Cambridge, MA, USA, 1974.

[18] P.-H. Robert and G. Juanole. Modélisation et vérification de politiques d'ordonnancement de tâches temps-réel. In *8ème Colloque Francophone sur l'Ingénierie des Protocoles-CFIP'2000*, pages 167–182, 17-20 octobre 2000.

[19] O. H. Roux and A.-M. Déplanche. A t-time Petri net extension for real time-task scheduling modeling. *European Journal of Automation (JESA)*, 36(7):973–987, 2002.

[20] D. C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2), February 2006.

[21] L. Sha, T. Abdelzaher, K.-E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems Journal*, 28(2/3):101–155, 2004.

[22] F. Singhoff, J. Legrand, L. t. Nana, and L. Marcé. Cheddar : a flexible real time scheduling framework. *ACM Ada Letters journal, 24(4):1-8, ACM Press, ISSN :1094-3641*, Nov. 2004.