J Supercomput
DOI 10.1007/s11227-011-0557-9

Compositional specification of real time embedded systems by priority time Petri Nets

Adel Mahfoudhi · Yessine Hadj Kacem · Walid Karamti · Mohamed Abid

© Springer Science+Business Media, LLC 2011

Abstract An important key challenge in Embedded Real Time Systems (ERTS) analysis is to provide a seamless scheduling strategy. Formal methods for checking the temporal characteristics and timing constraints at a high abstraction level have proven to be useful for making the development process reliable. In this paper, we present a Petri Net modeling formalism and an analysis technique which supports not only systems scheduling analysis but also the compositional specification of real time systems. The proposed Priority Time Petri Net gives determinism aspect to the model and accelerates its execution. Indeed, a compositional specification of a PTPN for complex application and multiprocessor architecture that solves the problem of hierarchy is presented.

Keywords ERTS · Scheduling analysis · PTPN · Compositional specification

1 Introduction

An important key challenge in Embedded Real Time Systems (ERTS) analysis is to provide a seamless scheduling strategy. Formal methods for checking the temporal characteristics and the timing constraints at a high abstraction level have proven to be useful for making reliable the development process. In fact, the specification and

A. Mahfoudhi (✉) · Y. Hadj Kacem · W. Karamti · M. Abid
CES Laboratory, University of Sfax, ENIS Soukra km 3,5, B.P. 1173-3000 Sfax, Tunisia
e-mail: adel.mahfoudhi@fss.rnu.tn

Y. Hadj Kacem
e-mail: yessine.hadjkacem@ceslab.org

W. Karamti
e-mail: walid.karamti@ceslab.org

M. Abid
e-mail: mohamed.abid@enis.rnu.tn

Published online: 02 February 2011

formal verification techniques constitute an important and recent field of research, whose purpose is to be more trustworthy.

The model checking category of formal method produces an example showing why an imposed constraint is not satisfactory. Thus, Petri Nets (PN) have been successfully used in ERTS specification. Petri Nets also present an adequate model checking thanks to their great expressivity dynamic vision and executable aspect. Consequently, various PN extensions and generalizations and numerous supporting computer tools have been developed to further increase their modeling opportunities, though falling short of ERTS verification. However, the application of Petri Nets to a scheduling problem is far from being trivial; determinism aspect is undertaken neither in regular Petri Nets nor in their extension for real time system verification. For schedulability analysis operations [21], verifying whether a schedule of a task execution meets the imposed timing constraints, is more challenging. This is due to the distribution of and complex interactions between the multiple execution resources of the target hardware architecture. In fact, it has been proven that when it is used for modeling the distribution of a parallel application onto heterogeneous computing resources, the specification power of Petri Nets reaches its limitations. In particular, the following problems were identified:

- Need for supporting systems scheduling analysis, considering periodicity, fixed priority, preemption, multiprocessor architecture and intertask relations
- Favoring determinism aspect for the model and accelerating its execution
- Necessity of simple scheduling problem presentation: while modeling complex real time tasks, a hierarchical approach is necessary in order to allow the designer to separate issues of local modeling from those of global modeling.

In this paper, we present a Petri Net modeling formalism and a technique of analysis that support both systems scheduling analysis and the compositional specification of real time systems (RTS). The proposed Priority Time Petri Net (PTPN) gives determinism aspect to the model and accelerates its execution. Indeed, the compositional specification of a PTPN for complex application and multiprocessor architecture solves the problem of hierarchy while modeling complex systems. Besides, the developed tool of our support infrastructure is founded on the Eclipse platform, in the form of a plug-in capable of exchanging data with Eclipse Features.

The remainder of this article is organized as follows. Section 2 provides a brief discussion about related work. As an introduction to the technical sections, basic concepts on Petri Nets are overviewed in Sect. 3. Then Sect. 4 introduces the Priority Time Petri Nets highlighting computational and analysis models. Next, Sect. 5 presents how our extended formalism support is used for specifying the schedulability analysis problem. Afterward, the local composition, explained in Sect. 6, yields the main PTPN components. While the interconnection between PTPN components is illustrated in Sect. 7, their analysis is described in Sect. 8. As for Sect. 9, it introduces the supporting tool and depicts experiments. Finally, the proposed approach is briefly outlined and future perspectives are given in the last section.

2 Related work

The adoption of model checking by means of PN extensions has been vastly investigated. In [19], the authors use T-Timed stochastic Petri Nets to model real time tasks. They stress the independent periodic tasks in order to support Rate Monotonic (RM) strategy [13]. For them, 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. Besides, 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 encounter difficulties to interpret the graph.

Roux [20] 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 a calculation of a more reduced state space than the traditional state graph suggested by [2] and the difference bound matrix. This work is extended in [11] to support the tasks with variable time execution. Another improvement [12] consists in extending the STPN to manage the dynamic priorities and support variant scheduling policies.

In the same line of thought, Berthomieu et al. [3] introduced the notion of priorities within the temporal Petri nets framework via the PrTPN (Priority Time Petri Nets) extension. His proposal is based on the constraint that *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 Net (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 the authors does not take into account the periodicity of real time tasks.

Pailler and Choquet-Geniet, in [16] propose a methodology for scheduling analysis from accessibility graph. They suggest to extract a scheduling graph that contains all valid sequences. The optimal sequence presents a simple activity. However, this approach is restricted to the offline scheduling.

The literature also illustrates other research work carried out to address the separation problem of local PN blocs from global modeled network. For example, the approach presented in [23] for scheduling analysis specification is based on the separation between timing and behavior properties. In the proposal, the compositional timing analysis aims at easily analyzing the structured Time Petri Nets. But, the structured Petri Nets elements have to be decomposed into a number of subsequences in order to test the system schedulability regardless of the direct model analysis construction.

3 Preliminaries

In this section, background concepts that are useful to understand our proposal are provided. In what follows, some definitions on Petri Nets are briefly recalled. Then the hierarchical approach in PN is explained through the concept of Petri Nets in object. Finally, this section ends with a brief discussion.

3.1 Background for Petri Nets and Time Petri Nets

Petri Net [17] is a mathematical formalism that allows the specification of the behavior of real time interactive systems. Concurrent real time concepts such as timing constraints, shared resource and synchronization are taken into account through two main Petri Nets extensions: Time Petri Nets [14] and Timed Petri Nets [18].

Definition 1 A Petri Nets is a 4-tuple [15], $R = \{P, T, B, F\}$, where:

(1) $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places $n > 0$;
(2) $T = \{t_1, t_2, \ldots, t_m\}$ 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 and defined by: $M : P \mapsto \mathbb{N}$.

Definition 2 A marked Petri Net is a couple $\text{PN} = \langle R, M_0 \rangle$, where R is a Petri Nets and M_0 is the initial marking.

Definition 3 A vector Ft_s is associated with each marking M which presents the transitions to fire. 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 presented as follows:

$$M' = M + F(P,t) - B(P,t)$$

In the rest of this section, 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 expressivity, the regular PN makes it possible to describe parallelism, dependency, and semaphores, which present important properties of ERTS. In spite of this significant ability, regular PNs are not able to model the temporal evolution of ERTS. That is why the regular PN is extended to Time Petri Net.

Definition 6 Time Petri Nets associate a static time interval with the transition firing. A Time Petri Net is a 3-tuple *TPN*:
$TPN = \{\text{PN}, T_{\min}, T_{\max}\}$, 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 with the system.

Definition 7 Each transition in a *TPN* is accompanied with 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 is valid.

Definition 8 A transition is valid when it is firable and the local clock has met the time interval:

$$t \text{ is valid } \Leftrightarrow t \in Ft_s \wedge Hl(t) \in [T_{\min}, T_{\max}]$$

Definition 9 Only valid transitions will be fired. To simplify their selections, the vector Ft_s is filtered and the temporal filter is described by:

$$t \text{ is not valid } \Longrightarrow \begin{cases} Ft_s(t) \leftarrow 0 \\ \textit{Set Increment } Hl(t) \end{cases}$$

Hence, the marking after filtering is: $M' = M + F(P,t) - B(P,t) \Longrightarrow t \in Ft_s$.

3.2 Petri Nets in objects

A major tendency of integration between objects and PN consists in using the networks to describe the internal behavior of the objects. In this approach, the internal state of the object is modeled by tokens in the places, and the execution of a method by the object is modeled by the net's transitions. So, the net structure specifies the availability of a method according to the internal state of the object, and indicates the possible sequences of methods execution by the object. The interest of Petri Nets is to describe the intrinsically competing objects capable of executing several methods at the same time. Furthermore, certain transitions of the net can remain "hidden" or protected inside an object, and therefore model the internal and spontaneous behavior of an object by contrast to the services it offers to its environment.

The fundamental concern of such approach is to allow the use of concepts stemming from the objects approach (classification, encapsulation) to describe the system structure, instead of using a purely hierarchical structuring. In the "Petri Nets in objects" paradigm, a system is described as a set of objects which communicate the behavior of each object being described in terms of Petri Nets. Mostly, these approaches are class-based, which allows the association of a PN with a class of objects rather than with an individual object.

3.3 Discussion

As already mentioned, regular PNs are not able to support the temporal evolution of ERTS. Among the multitude of the existing extensions of Petri Nets for ERTS modeling, the following can be cited:

- Timed Petri Nets in which time can be assigned to places or transitions. In fact, P-Timed Petri Nets and T-Timed Petri Nets 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 from the input place at an indeterminate time and added to the output place after the duration d_i. Timed Petri Nets can lead to a well-performing analysis method but could not cover the scheduling problem which requires 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 Net is quite accurate for the task behavior characterization, there are some issues in the cases of the real time 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 an interval of firing the transitions which present an event. This could bring the passage of a task state towards another during an undeterminable time. Indeed, in our study, the 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 the set of all 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 overcome the indeterminism problem.

As mentioned in Sect. 2, the existing research work suffers from two major problems. The first one concerns the indefinite date of transitions firing. Indeed, the extensions of the PN dedicated to the analysis of RTS scheduling are based on the temporal or delayed PN. For example, the STPN [20], the extension of the temporal PN, adds constraints on the crossing of the transitions to verify the respect for the firing interval. Besides, the check of these constraints is a new dimension added to the problem of scheduling analysis. The determinist crossing of the valid transitions presents our first contribution for the modeling of the RTS.

In order to solve the problems of conflict of the valid transitions, the existing work has integrated the priorities within the PN but the way to do so is the second problem. Indeed, most of the extensions such as PrTPN [3] and STPN [20] make appeal to additional components such as the inhibitory arcs to specify the priorities. In fact, the modeling of a complex RTS with the PN generally produces complex models. Besides, the addition of the other constituents gradually increases their complications.

The mastery of the increasing complexity of the PN models of scheduling analysis is our second contribution. Firstly, we present a new way of integrating the priorities into networks without using additional components. Secondly, we propose a new modeling of the RTS based on components which mask PN models.

4 Priority Time Petri Net: PTPN

PTPN is proposed to overcome the firable transition problems. Only the transition that has the highest priority is fired with the extended formalism. The PTPN semantics permits to support the scheduling strategy with a fixed priority such as Rate Monotonic (RM). In what follows, the basic definitions of the proposed formalism are stressed. Then its firing semantic which accelerates the simulation of firing transition vector is explained.

4.1 PTPN elements

Definition 10 PTPN is a 3-tuple defined by $PTPN = \langle R, T_f, P_r \rangle$ where:

(1) R is a regular Petri Net
(2) $T_f : T \mapsto \mathbb{Q}^+$ is the firing time 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 rows and P columns,

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

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

Definition 11 The marked PTPN is the couple $MPTPN = \langle PTPN, M_0 \rangle$:

(1) $PTPN$ is a PTPN network
(2) M_0 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 preserved.

Definition 12 A transition is valid if it is firable and it respects it firing date.

$$\forall t \in T \text{ is valid} \Longleftrightarrow t \in Ft_s \wedge Hl(t) = T_f(t)$$

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, then this vector will present only independent transitions.

Definition 13 Two transitions are independent if the firing of the first does not influence that of the second. Let $t_1, t_2 \in T$ with $B(P,t_1) \leq M$ and $B(P,t_2) \leq M$

$$t_1 \text{ and } t_2 \text{ are independent} \Leftrightarrow \begin{cases} M' = M + F(P,t_1)B(P,t_2) \\ \text{and} \\ B(P,t_2) \leq M' \end{cases}$$

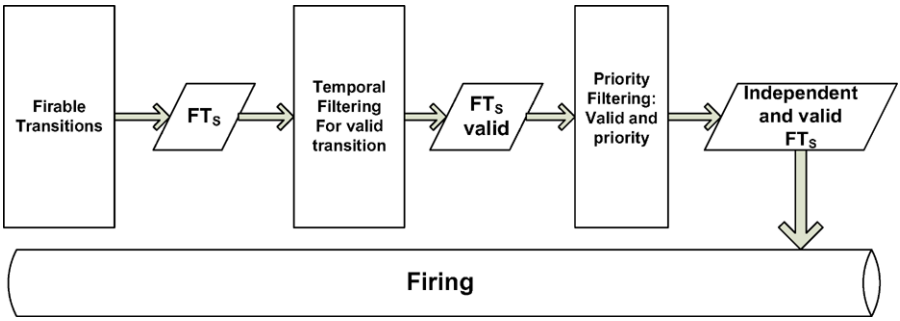

Fig. 1 PTPN Firing Machine

t_1 influence t_2 if t_2 is not enabled by the new marking M'. To accelerate the firing of transitions with PTPN, we propose a method that makes it possible to fire a vector of independent transitions.

After describing the necessary PTPN definitions, we present essential concepts of the method for merging the transition firing and the PTPN state space in order to compute a feasible system schedule.

Definition 14 To fire a marking, a PTPN Firing Machine (PFM) is offered by our extended formalism, whose mechanism is described in Fig. 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 the firing time.

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 for 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 place is selected from the vector $Prod(p_i)$.

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

- $i \in [0, n]$
- $j \in [0, m]$
- 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 $i \in [0, n]$

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

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 input marking and M' is the output marking. To simplify notations, M' is said to be accessible via M through Ft_s.

Algorithm 1 State graph construction

```
repeat
    if valid transition exists then
        Firing Fts and state construction and New marking
    else
        Set increment clock
    end if
until marking ≺ B
```

4.2 PTPN state space

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

The construction of PTPN state space 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 tick, an iterative process is triggered. A marking is sent to the PFM. Next, the machine brings back 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 present the new entry point of the PFM. This iteration is continued while there are firable and valid transitions.

5 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.

5.1 Task creation and activation with PTPN

We will use the net shown in Fig. 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 Fig. 2, there are two tokens; one in the place "Uncreated" and another in the place "Disabled." 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 place "Disabled" 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 transition "Uncreated" 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 transition "Activation" is enabled, the firing time is 0 and consequently it is validated. "Activated" 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."

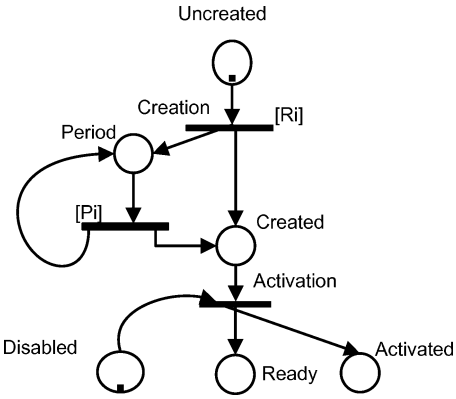

Fig. 2 Creation and activation of a task with PTPN

5.2 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 toward the execution state as well as that of the processor toward 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 preemptive 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, otherwise it will be anticipated. The modeling with PTPN similarly inserts 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 modeled 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

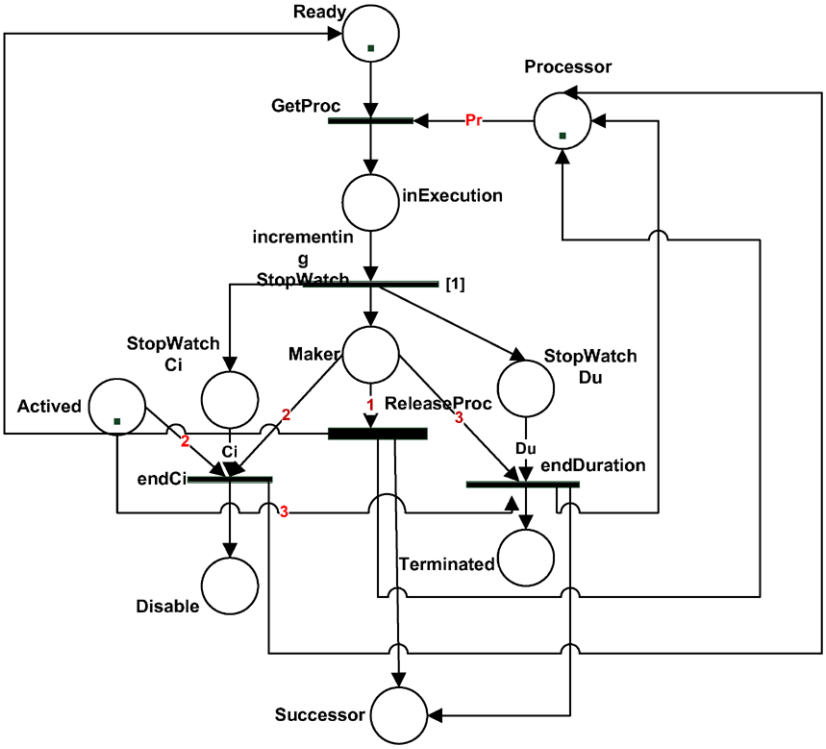

Fig. 3 Executing a task with PTPN

PTPN, we need a shared place between both transitions. Figure 3 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 preemption.

A task in the course of execution on a processor can be preempted by a more prioritized one. With PTPN, the common task releases the processor after each unit of time by firing the transition "ReleaseProc" even if it is still the most prioritized. The firing of the transition engenders the passage of the current task towards the state ready for the execution. The place "Ready" becomes marked, which leads to the state where all the active tasks, the most prioritized of which is then going to occupy the processor, are ready for the execution.

Two scenarios are possible. The current task, still the most prioritized, is going to continue its execution. Should the opposite occur, the other more prioritized one occupies the processor and the common task would be preempted.

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 Fig. 4, there are four places ("Period," "Created," "Activated," "Deadline"), two transitions ("Tperiod," "Tdeadline") and $T_f(0, 0)$. The transition "Tperiod" is valid for each P_i time unit. The firing consumes the token from the place "Period" and produces two: One in the place "Created" and the other in the

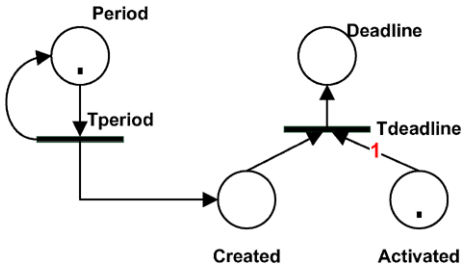

Fig. 4 Meeting deadlines with PTPN

place "Period." This means that the task is ready to wake up and the new period is loaded. If the task has already been activated, the transition "Tdeadline" is valid. The firing of the transition "Tdeadline" provides a token to the place "Deadline" and disables the task execution.

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

6 Local PTPN components for scheduling analysis specification

As highlighted before, an RTS consists of a set of real-time tasks and a set of processors. The formal definition of an ERTS is presented by Definition 18.

Definition 18 The ERTS Ω is defined by the 5-uplet $\Omega = \{TK,\ R_s,\ Proc,\ alloc,\ Prio\}$, with:

- TK: is a finite set of real-time tasks with each task determined by the following parameters:
  - R_i: the date of the first activation
  - P_i: the period associated with the task
  - C_i: the execution period of the task for the P_i period
  - D_u: the life cycle of the task (the duration of total execution). For each task, two invariants must be respected: $C_i \leq D_u$ and $R_i \leq P_i$
- $Proc$: a finite set of processors
- R_s : $TK \mapsto \{TK\} \cup \{\oslash\}$, a function which initializes precedence relations between tasks
- $Alloc$: $TK \mapsto Proc$, a function which allocates a task to a processor
- $Prio$: $TK \times Proc \mapsto \mathbb{N}$, a function which allocates priorities to the tasks according to the processor. If the task is not allotted to a processor, the priority will be null, that is to say:
  $\forall TK_i \in TK \wedge P \in Proc,\ Alloc(TK_i) = P \Rightarrow \forall P_j \in Proc \backslash \{P\},\ Prio(P_j) = 0.$

It should be born in mind that we are interested in the multiprocessor scheduling by clustering, where each task is assigned to a single processor. In other words, a task can

take place only on a single resource of execution. So, the problem of multiprocessor scheduling is reduced to monoprocessor subsolutions. That is why the application of the policies of monoprocessor scheduling is justified. Our concern is in the RTS with a fixed priority where the RM strategy is an optimal policy. From the moment a task is created, a priority is allocated to it and the priority remains fixed until its end.

In order to apply the detailed structuring in the previous section, we start with the modeling of the behavior of the first object "Task."

6.1 Task behavior encapsulation

The PN paradigm in objects necessitates the encapsulation of the various behaviors of the object in a centenary called PN component. We propose a PTPN constituent called "Tc" to encapsulate the behavior of a real-time task. It is characterized by input and output places which assure the communication with the environment. Figure 5 describes the graphic presentation of the constituent "Tc."

Formally, "Tc" is defined as follows:

Definition 19 Let a real-time task be $T_i \in TK$ of Ω with the following characteristics $(Ri,\ Pi,\ Ci,\ Du)$, then the corresponding component "Tc" is described by the following triplet: $Tc = \langle PTPN,\ PI,\ PO \rangle$, where:

- The main characteristics of PTPN are:
  - $P : \{P_{\text{uncreated}},\ P_{\text{period}},\ P_{\text{deadline}},\ P_{\text{created}},\ P_{\text{PreviosRessorce}},\ P_{\text{activated}},\ P_{\text{disabled}},\ P_{\text{ready}},\ P_{\text{GetProc}},\ P_{\text{inexecution}},\ P_{\text{maker}},\ P_{\text{stopwatchCi}},\ P_{\text{stopwatchDu}},\ P_{\text{successor}},\ P_{\text{terminated}},\ P_{\text{Release}}\}$
  - $T : \{T_{\text{creation}},\ T_{\text{period}},\ T_{\text{deadline}},\ T_{\text{activation}},\ T_{\text{Execution}},\ T_{\text{incrementing}},\ T_{\text{releaseProc}},\ T_{\text{endCi}},\ T_{\text{endDuration}}\}$
  - $T_f(T_{\text{creation}}) = RI;$
    $T_f(T_{\text{period}}) = P_i;$
    $T_f(T_{\text{incrementing}}) = 1;$
    $\forall t \in T \backslash \{T_{\text{creation}},\ T_{\text{period}},\ T_{\text{incrementing}}\} \Leftrightarrow T_f(t) = 0$
- $PI = \{P_{\text{uncreated}}, P_{\text{PreviosRessorce}}, P_{\text{GetProc}}\}$, is a finite set of places defining the input interfaces for the Tc component.
- $PO = \{P_{\text{deadline}}, P_{\text{terminated}}, P_{\text{successor}}, P_{\text{Release}}\}$ is the finite set of output places of the "Tc" component. The places $P_{\text{terminated}}$ and P_{deadline} describe the scheduling of the task: either well organized or committing a temporary mistake. The places $P_{\text{Ready}},\ P_{\text{GetProc}},$ and P_{Release} are reserved for modeling the occupation and liberation of the processor. The places $P_{\text{PreviousResource}}$ and $P_{\text{successor}}$ present the communication between tasks.

The "Tc" component is the PTPN model, and then it disposes of the characteristics matrices of $PTPN\ (B, F, Pr)$. In order to simplify the presentation of matrices (B and F), we adopt the organization of the places and transitions presented in the P and T sets. We point out the following invariants:

- $\forall t \in T \wedge p \in PO \Rightarrow B(p,t) = 0$
- $\forall t \in T \wedge p \in PI \Rightarrow F(p,t) = 0$
- $\forall t \in T \wedge p \in P,\ B(p,t) = 0 \Leftrightarrow F(t,p) = 0.$

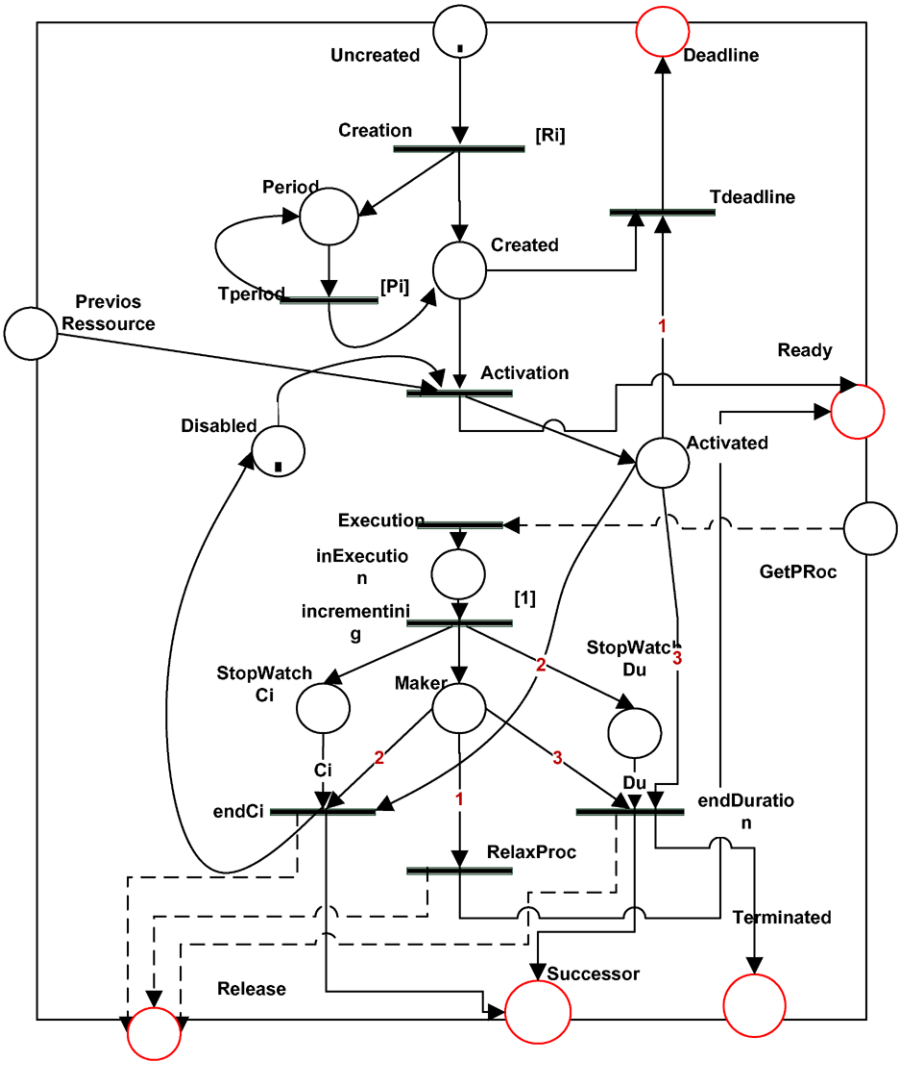

Fig. 5 PTPN component for describing a task

6.2 PTPN processor component

The second most important ERTS component is the execution resource. In our study, we shed the light on the processor resources in order to execute the system tasks. With PTPN, each resource is modeled by a simple place. The marking of the place describes the state of the resource. The presence of a mark indicates that the resource is free and ready to execute a task as shown in Fig. 6. The absence of the mark indicates that the resource is occupied by a system task Ω. The number of the processor resources is described by the Proc set of the system.

Fig. 6 Processor component

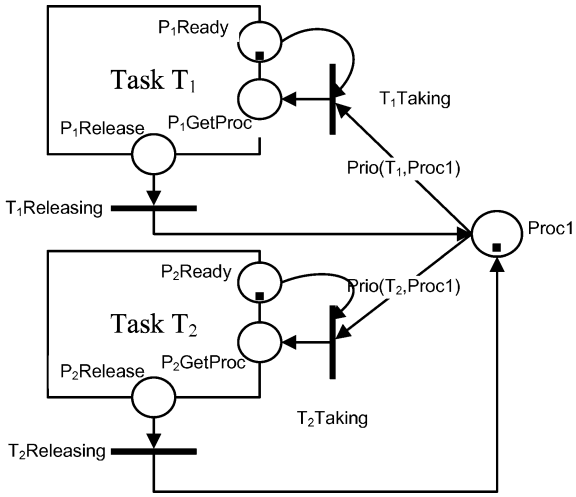

Fig. 7 Communication Task/Processor

7 Modeling of the communications between PTPN components

The scheduling analysis modeling requires that the modeling of the behavior of the system is described through the communication between the various components of an RTS. We distinguish two major types of communication: the Task/Processor communication and the Task/task communication.

7.1 Modeling of the communication Task/Processor

The analysis of scheduling is applied for a distribution of the tasks according to processors. The distribution is a description of the processors allocations by all the tasks. It is defined by the *Alloc* function of the system Ω. The communication Task/Processor respects the proposed allocation. It is presented through both events: the occupation and liberation of the processor resources. During the modeling, both events are symbolized by two transitions "Taking" and "Releasing" associated with each task.

Figure 7 describes the allocation of a processor "Proc1" to two tasks "Task T1" and "Task T2." The occupation events of "Proc1" by "T1" and "T2" are modeled consecutively with the transitions "T1Taking", "T2Taking." Both tasks are ready to execution. Such a state is described by the presence of a token in the output place "PiReady" for each: "Task T1" and "Task T2." The place "Proc1" contains a token, both occupation transitions are enabled.

The place "Proc1" is a common resource for both transitions "P1Taking" and "P2Taking." The priorities of these two transitions with regard to "Proc1" are the

task priorities "Task T1" and "Task T2" pertaining to the processor. The task priorities are defined by the Prio function of Ω. The priorities with PTPN are placed on the input arcs to the occupation transition. The firing of the most prior transition allows the consumption of the token from the place "Proc1" and the production of a token in the input place "PGetProc."

No sooner does the output place "PRelease" indicate the presence of a token, then the transition "TRelease" is enabled. The firing allows the liberation of the processor by putting a token in the place "Proc1."

7.2 Communication between tasks with PTPN components

Most of ERTS frequently require data transmission between the tasks. Precedence constraint is dealt with to indicate any communication between tasks. We suppose that the size of the buffer used for the communication is infinite. This constraint allows the input task to send the information as soon as it finishes all or a part of its activity without the risk of waiting for the buffer liberation or information destruction. As a consequence, the emitting task is not going to exceed its term because of the communication. Thus, a condition necessary for the scheduling can be revealing. Two tasks which precede the output task must have the necessary information before its term.

In the present study, the periods of the input and output tasks are not identical: it is the case of the constraints of generalized precedence [16]. We adopt an RM modification in order to organize this type of tasks. Such a modification consists in allocating more elevated priorities to input tasks than output ones. For the independent tasks, the priorities are attributed according to the ordinary RM. The relations of precedence are described by the R_S function defined in the system Ω. The start set of R_s describes the input tasks, and the end set describes the output ones. The function R_s^{-1} which is the opposite of R_s is the set of the output toward the input tasks. We highlight the following invariants for any $t \in TK$:

– If T_i is an input task, then $R_s(T_i) \neq \oslash$
– If T_i is an output task, then $R_s^{-1}(T_i) \neq \oslash$.

As mentioned before, each component "Tc" has an input place "PreviousResource" and an output place "Successor" preserved for the communication with the other tasks. When the input place is marked, then the component receives all the necessary information and becomes ready to be activated. The marking of the output place indicates that the constituent finishes a part of or all the activity and becomes ready to emit the information to the receiving components. The sending and receiving events are modeled by associated transitions "Send" and "Receive" for each constituents "Tc." Figure 8 describes the PTPN modeling with one emitting component "TaskE" and two receiving components "TaskC1" and "TaskC2." The transition "TESend" is the transfer transition associated with transmitting components. The output place "Psuccessor" of the emitting component dispose of a token, then "TESend" is sensitized. The firing allows the placement of a token in the places "TE2TC1," "TE2TC2," and "WakeTE". The marking of "WakeTi" indicates that the task Ti has finished an activity and becomes ready to finish the following. The place "Ti2Tj" designates the

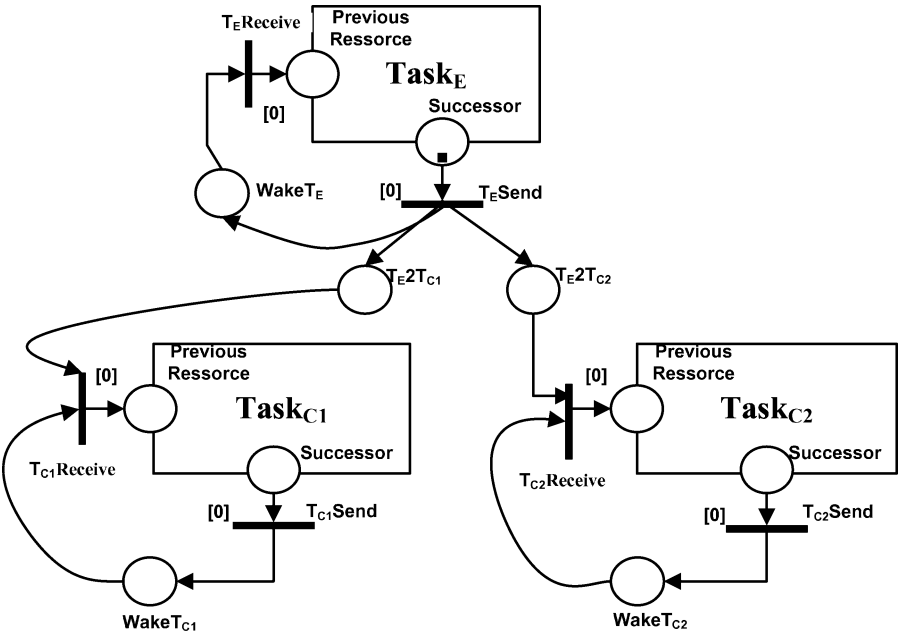

Fig. 8 Communication between Task components

information to be sent by the input task Ti to the output Tj. The transition of the reception of information "Receive" is valid only when the corresponding place "Wake" and the places of the received information are marked. For the input tasks, the validation of the transition "Receive" necessitates the marking of the place "Wake." We have supposed that the duration of the sending and that of receiving are negligible and that is why the transitions do not take time for firing.

8 Computational model of PTPN components

The graphic modeling of a real-time system has become an easy activity thanks to the structuring. The new modeling considers only the behavior of the system while the internal behavior of every component does not emerge. The structuring also appears in the matrix presentation of the model. Each PTPN matrix must be well organized respecting the new constituents which are created. In the following section, we propose an approach of construction and organization of three matrices of the PTPN: B, F, and Pr. The strategy of construction is based on the following four stages:

1. Determine the size of each matrix
2. Fill in the components (Tc, Processor) in the matrices
3. Initialize the matrices by the allocations Task/Processor
4. Initialize the matrices by the precedence between tasks.

The matrix structuring allows the clear legibility of the model matrices. Therefore, the reverse passage from the mathematical representation of the model to the graphic one has become easier.

8.1 The dimensions of the matrices

The dimensions of the PTPN matrices depend on the number of tasks, and the relations of precedence between them. The Backward and Forward matrices have the same dimension: "nbPlaces, nbTrans." The matrix of priority Pr is the transposed dimension of the Pre: "nbTrans, nbPlaces" with:

- nbPlaces: the number of places that constitutes PTPN model of Ω
- nbTrans: the number of model transitions.

The number of places is determined by the following rule:
$nbPlaces = [nbPlaces_{Tc} \times size(TK)] + size(Proc) + nbRes$, with:

- $nbPlaces_{Tc}$: the number of the places of a single component "Tc"
- $size(TK)$: the number of the system tasks Ω
- $size(Proc)$: the number of the processors of Ω
- $nbRes$: the number of the places that present the transferred resources of the input to output tasks and the resources "Wake." It is determined as follows: $nbRes = size(TK_i) + \sum_{i=0}^{size(TK)} size(R_s(TK_i))$, with:
  - $size(TK_i)$ designates the sum of the resources "Wake" which is equal to the task number.
  - $size(R_s(TK_i))$ is the sum of the dimensions of the arrival set of R_S for each input task which designates the number of the transfer places.

The number of the model transitions is the sum of: the transitions of all the tasks, the transitions of communication Task/Processor and transitions of communication Task/Task. It is determined by:
$nbTrans = [nbTrans_{TC} \times size(TK)] + nbTrans_{TP} + nbTrans_{TT}$, with

- $nbTrans_{TC}$: the number of the transitions of a component "Tc"
- $nbTrans_{TT}$: the number of the transfer transitions between the components. Each component is associated with a couple of transitions: send and receive, where the following rule: $nbTrans_{TT} = 2 \times size(TK)$
- $nbTrans_{TP}$: the number of the transitions responsible for the occupation and liberation of the processors and tasks. $nbTrans_{TP} = 2 \times size(TK)$.

8.2 Filling of the matrices by PTPN components

For the RTS modeling with component-based PTPN, we offer the standard structure for the matrices. The B and F matrices are described by the following format:

- On the lines:
  - the places of the B and F matrices of each component "Tc"
  - the processors component
  - The transfer resources

– On the columns:
  – the transitions B and F matrices of each "Tc"
  – The transitions of the occupation and liberation of processors. The transitions which are associated with the task of TK_1 are placed in front of that of TK_2, etc.

The Pr matrix is the transposed format of the Pre matrix; the lines of Pre are the columns of the Pr. Following the formats of each matrix, we fill in the matrices of each component "Tc" in the section reserved to the system matrices.

8.3 Initialization of the communications Tasks/Processors

The initialization of the allocations of the processors by the tasks in the matrices consists in initializing the events of occupation and liberation. We propose the rules responsible for this initialization for each matrix. $\forall TK_i \in TK,\ \forall Proc_j \in Proc$,

– the matrix B is expressed by:
  if $Alloc(TK_i) = Proc_j$
  $$\Leftrightarrow \begin{cases} B(Proc_j, (T_{\text{Taking}})_i) = 1 \\ B(TK_i(P_{\text{Ready}}), (T_{\text{Taking}})_i) = 1, \quad \text{with} \\ B(TK_i(P_{\text{Release}}), (T_{\text{Releasing}})_i) = 1 \end{cases}$$
  – $(T_{\text{Taking}})_i$: the transition of the occupation associated with the component "Tci" which describes the task TK_i
  – $TK_i(P_{\text{Ready}})$: the output place of "Tci," responsible for the processor demand
  – $(T_{\text{Releasing}})_i$: the transition of liberation associated with the component "Tci"
  – $TK_i(P_{\text{Release}})$: the output place of "Tci" used to launch the liberation of the processor
– the F matrix is illustrated by:
  if $Alloc(TK_i) = Proc_j$
  $$\Leftrightarrow \begin{cases} F(TK_i(P_{\text{GetProc}}), (T_{\text{Releasing}})_i) = 1 \\ F(Proc_j, (T_{\text{Releasing}})_i) = 1 \end{cases}$$
  with $TK_i(P_{\text{GetProc}})$ the input place of "Tci" which describes the processor resource occupied by the task TK_i
– While identifying the Pr matrix, we initialize that of the priority system of the tasks running on processors as they are described by the $Prio$ function of Ω
  if $Alloc(TK_i) = Proc_j$
  $$\Leftrightarrow \begin{cases} Pr((T_{\text{Taking}})_i, Proc_j) = Prio(TK_i, Proc_j) \\ Pr((T_{\text{Taking}})_i, TK_i(P_{\text{Ready}})) = 1 \\ Pr((T_{\text{Releasing}})_i, TK_i(P_{\text{Release}})) = 1 \end{cases}$$
  For both "PReady" and "PReleasing" input places, they do not have several shared transitions. It is enough to allocate the default priority "1" to them.

8.4 Initializing the matrices by the precedence between tasks

The communication between tasks is initialized in the matrices with the same method illustrated in the previous subsection. $\forall TK_i,\ TK_j \in TK$

– the matrix B is expressed by:

if $R_s(TK_i) = TK_j$

$$\Leftrightarrow \begin{cases} B(TK_i(P_{\text{Successor}}), (T_{\text{send}})_i) = 1 \\ B((P_{\text{Wake}})_i, (T_{\text{Receive}})_i) = 1, \quad \text{with} \\ B(T_i 2T_j, (T_{\text{Receive}})_j) = 1 \end{cases}$$

- $(T_{\text{send}})_i$: the transition for sending information to the receiving task
- $TK_i(P_{\text{Successor}})$: the output place of "Tci," responsible for the communication with tasks
- $(T_{\text{Receive}})_j$: the transition associated with "Tcj" that receives information from the sender component
- $T_i 2T_j$: the place that models the information sent by "Tci" to "Tcj" component.
- $(P_{\text{Wake}})_i$: the place associated with "Tci" component to indicate that the task is ready for achieving execution time during the new period

– the F matrix is illustrated by:
if $R_s(TK_i) = TK_j$

$$\Leftrightarrow \begin{cases} F(T_i 2T_j, (T_{\text{Send}})_i) = 1 \\ F(TK_j(P_{\text{PreviousResource}}, (T_{\text{Receive}})_j)) = 1 \end{cases}$$

with $TK_j(P_{\text{PreviousResource}})$ is the input place of the receiver component "Tci" which describes the presence of all necessary resources for the wake-up

– Pr is identified as:
$if\ Alloc(TK_i) = Proc_j$

$$\Leftrightarrow \begin{cases} Pr((T_{\text{send}})_i, TK_i(P_{\text{Successor}})) = 1 \\ Pr((T_{\text{Receive}})_i, (P_{\text{Wake}})_i) = 1 \\ Pr((T_{\text{Receive}})_j, T_i 2T_j) = 1. \end{cases}$$

9 Tools and experiments

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

9.1 Tooling support

The implemented tool takes the form of a Petri Net editor and an executer model of the modeled net. Indeed, the developed editor for our PTPN relies on the GMF founded on Eclipse Modeling Framework (EMF). Besides, the definition of the PTPN Meta Model represents the starting point of the editor's generation process. As shown in Fig. 9, the regular Petri Nets Meta Model is extended with the *priority* attribute 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) [8]. The validation doubles through the verification during and after constructing the 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.

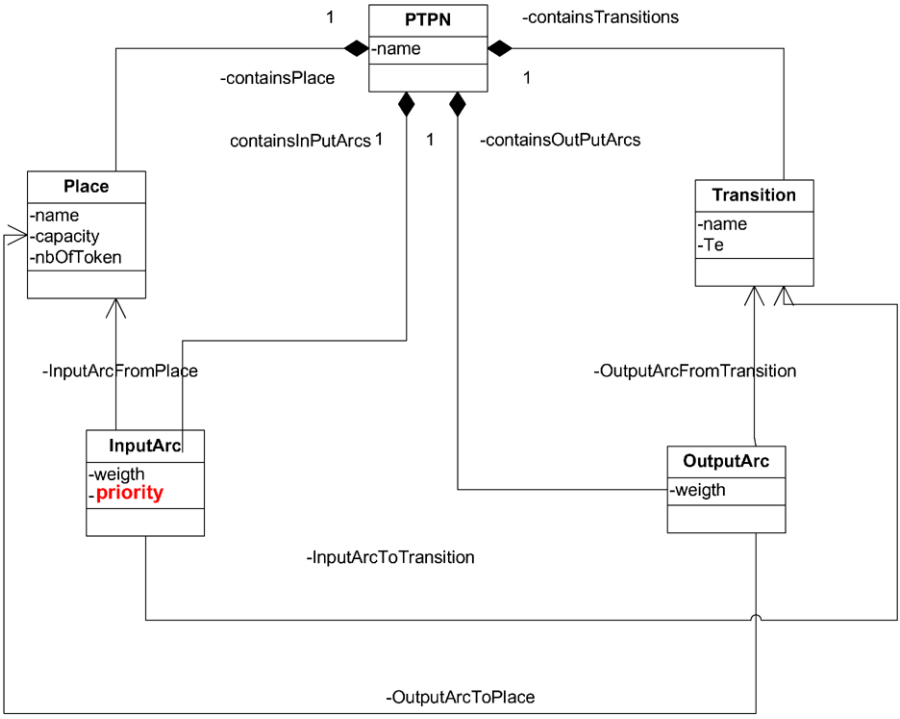

Fig. 9 PTPN Meta Model

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 executer. Due to the structure of the editor output, the properties of the modeled 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. Despite the representation of PTPN elements provided by the editor, the palette is equipped with PTPN components in order to facilitate the illustration of complex tasks and computing resources. So, it is sufficient for the developer to select the structured PTPN class from the palette with the communication means.

Compared to the existing Time Petri Nets simulators such as ROMEO [7] and TINA [4], the impetus of our tool are the integration of the priority concept and its structured input/output files and Petri components which guarantee interaction with the existing PN simulators and Eclipse features.

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

– Contrary to Cheddar tools [22], Mast [10], 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.

Table 1 Tasks' characteristics of the experiment

Id_Task	R_i	C_i	deadline	Period	Duration
T1	1	1	3	3	3
T2	0	2	5	5	4
T3	0	2	6	6	2
T4	0	2	5	5	4

– Pertaining to other extensions presented in Sect. 2, PTPN offers a strategy which accelerates the marking and avoids the combinatorial explosion in front of a large number of states.

9.2 Case study

In the present section, we introduce the technique of how PTPN Components solve the scheduling analysis problems of the real-time systems. It is through a case study that our PTPN extension and the PTPNS tool are brought to the light.

In fact, we present a generic experiment which consists of a nonschedulable system. In the latter, we establish the way how the PTPNS supplies a description of the temporal fault to help the designers in refining the partitioning space Sw/Hw. It should be born in mind that due to space limitation, we present a pedagogical experiment.

The case study deals with four tasks running on two processors. Using Definition 18, the specifications of the task characteristics (Table 1) as well as the allocation of the processors by the tasks are described as follows.

$TK = \{T1,\ T2,\ T3,\ T4\}$
$Proc = \{P1,\ P2\}$
$R_s(T1) = \{T3\} \quad Alloc(T1) = P1 \quad Prio(T1, P1) = 6$
$R_s(T2) = \oslash \quad Alloc(T2) = P1 \quad Prio(T2, P1) = 5$
$R_s(T3) = \{T4\} \quad Alloc(T3) = P2 \quad Prio(T3, P2) = 4$
$R_s(T4) = \oslash \quad Alloc(T4) = P2 \quad Prio(T4, P2) = 2$

The first stage consists in modeling each component of Ω with the PTPNS tool. The result is a PTPN model composed of a set of PTPN components: Task and Processor, and the relation between them (Fig. 10). As for the second step, it consists in placing the marks on PTPN constituents: a mark in each constituent "Processor: P1, P2," a mark in the places "Uncreated" of the components "T1, T2, T3, T4" and a mark in the places "PreviousResource" of the components "T1, T2, T3,".

The following stage is the transformation of the model from the graphic shape towards its arithmetical shape (Sect. 8). This action assured by the PTPNS tool is essential for the execution of the model.

Let us recall that the strength of our PTPN extension is its capacity to determine step by step the valid scheduling sequence. In fact, the execution of the model with PTPNS allows the construction of the valid sequence. If the construction is well established, then the system is well scheduled, otherwise when the sequence encounters a marking, then the system is considered as non-schedulable. Moreover, the scheduling sequence supplied by PTPNS is optimal because PTPN supports the optimal RM strategy.

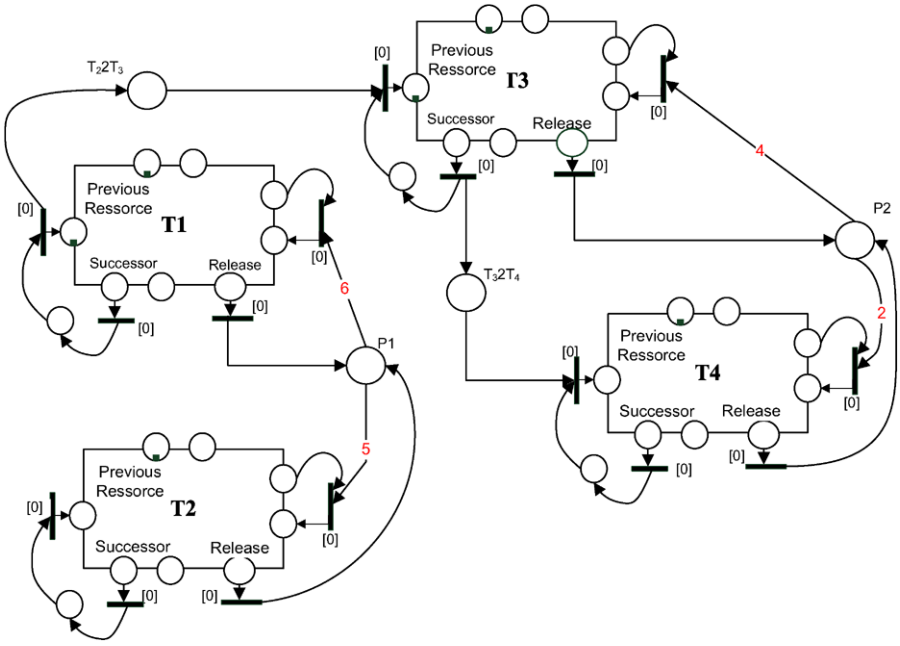

Fig. 10 PTPN components in the case study

For a better presentation, we have detailed the execution of the model (Fig. 10) in Table 2. Three columns are represented; the first "step" is the number of the steps in the scheduling, which corresponds to a state in the states graph explained previously. The first step starts with "step=0." The second column is "time"; it is the number of ticks of the global clock of the PTPN model. The last column "Ft_s" represents the high-priority and valid transitions in each step of scheduling.

The execution process of the model starts at 0 with the initial marking M_0. These two parameters present the necessary entries to launch the PFM machine. Such machine determines the high-priority and valid transitions in the column Ft_s: Transition-name (component-name). It is the case of "step0," in which the column Ft_s presents "Creating(T1, T2, T3, T4)." The simultaneous firing of this set of events gives birth to a new marking and a new step "step2."

What is worthy to note is that step "step28" presents two events "Period(T4)" and "IncrementingStopWatch(T4)." This indicates that the period T4 is provoked while the work of the last period is not achieved. The firing and the passage to the step "step29" brings about a new marking which replies to the event "deadline(T4)." The firing of this event implies the blocking of the execution, signaling a temporal fault (deadline).

The PTPNS tool is not sufficient to indicate a temporal fault but also supplies the combination cause of this fault to the designer. As in the present study, the PTPNS indicates that the partition (T3, T4) on P2 is a combination to be neglected in the future iterations of the partitioning.

Table 2 Model execution related to the case study

Step	Time	Ft_s	Step	Time	Ft_s
0	0	Creating(T2, T3, T4)	15	2	Executing(T3)
1	0	Activating(T2)	16	3	Incrementing StopWatch(T2, T3)
2	0	TakingProc(T2)	17	3	ReleaseProc(T3), endCi(T2)
3	0	Executing(T2)	18	3	Releasing(T2, T3), send(T2)
4	1	Incrementing StopWatch(T2), Creating(T1)	19	3	TakingProc(T3), Receive(T2)
5	1	ReleaseProc(T2), Activating(T1)	20	3	Executing(T3)
6	1	Releasing(T2)	21	4	Incrementing StopWatch(T3), Period(T1)
7	1	TakingProc(T1)	22	4	endCi(T3), Activating(T1)
8	1	Executing(T1)	23	4	Releasing(T3), send(T3), TakingProc(T1)
9	2	Incrementing StopWatch(T1)	24	4	Receive(T4), Execution(T1)
10	2	endCi(T1)	25	4	Activating(T4)
11	2	Releasing(T1), send(T1)	26	4	TakingProc(T4),
12	2	TakingProc(T2), Receive(T3,T1)	27	4	Executing(T4)
13	2	Executing(T2), Activating(T3)	28	5	Incrementing StopWatch(T1, T4), Period(T2, T4)
14	2	TakingProc(T3)	29	5	endCi(T1), Activating(T2), ReleaseProc(T4), deadline(T4)

10 Conclusion

Priority Time Petri Nets represent a powerful formalism for the scheduling analysis of real-time systems running on multiple processors, including periodic, dependant, and preemptive tasks with determinist strategy. The originality of the PTPN semantics is the attribution of a priority to transitions in order to avoid conflicts. Besides, the use of PTPN, as it is, leads to huge and sophisticated nets to be analyzed. That is why, we have realized a hierarchical composition allowing the considerable reduction of the size and complexity of the nets and facilitating their analysis. Rather than presenting a solution for the confusion problems, PTPN Firing Machine accelerates the PTPN evolution by applying temporal and priority filtering. In a regular PN, the firing of a transition requires identifying the new vector of fireable transitions. However, with PFM, this vector is established only after the firing of the old one. It is also guided by the priorities. Consequently, starting from a marking M, the PFM simultaneously fires valid transitions having the highest priority and returns the new marking M'.

In future research, we are interested in including performance analysis in the verification of ERTS and planning the integration of tasks with variable execution times. Indeed, we intend to incorporate PTPN in a Model Driven Engineering (MDE) process [6]. The transformation from annotated analysis models to a formal model allows the validation of the 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) [9] into PTPN

model to analyse system schedulability. The PTPNS tools taking the form of a plug-in could easily exchange data with UML Eclipse editors.

References

1. Amnell T, Fersman E, Mokrushin L, Pettersson P, Wang Yi (2002) 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, London, UK. Springer, Berlin, pp 460–464
2. Berthomieu B, Diaz M (1991) Modeling and verification of time dependent systems using time Petri Nets. IEEE Trans Softw Eng 17(3):259–273
3. Berthomieu B, Peres F, Vernadat F (2006) Bridging the gap between timed automata and bounded time Petri Nets. In: FORMATS, pp 82–97
4. Berthomieu B, Vernadat F (2006) Time Petri Nets analysis with tina. In: QEST, pp 123–124
5. Chen L, Shao Z, Fan G, Ma H (2008) A Petri Net based method for analyzing schedulability of distributed real-time embedded systems. J Comput 3(12). doi:10.4304/jcp.3.12.35-42
6. Douglas CS (2006) Model-driven engineering. IEEE Comput 39(2):25–31. doi:10.1109/MC.2006.58
7. Gardey G, Lime D, Magnin M, Roux OH (2005) Romeo: a tool for analyzing time Petri Nets. In: CAV, pp 418–423
8. Object Management Group (2003) UML 2.0 OCL specification. OMG adopted specification ptc/03-10-14. Object Management Group, October
9. Object OMG Management Group (2008) A UML profile for MARTE: modeling and analysis of real-time embedded systems, Beta 2, ptc/2008-06-09. Object Management Group, June
10. Gonzalez Harbour M, Gutierrez Garciia JJ, Palencia Gutierrez JC, Drake Moyano JM (2001) Mast: modeling and analysis suite for real time applications. In: Euromicro conference on real-time systems. p 0125
11. Lime D, Roux OH (2004) 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, Washington, DC, USA. IEEE Computer Society, Los Alamitos, pp 187–196
12. Lime D, Roux OH (2009) Formal verification of real-time systems with preemptive scheduling. Real-Time Syst 41(2):118–151
13. Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61
14. Merlin PM (1974) A study of the recoverability of computing systems. PhD Thesis, Univ California, Irvine. Available from Ann Arbor: Univ Microfilms, No. 75-11026
15. Murata T (1989) Petri Nets: properties, analysis and applications. Proc IEEE 77(4):541–580
16. Pailler S, Geniet AC (2004) Off-line scheduling of real-time applications with variable duration tasks. In: 7th workshop on discrete events systems, September, pp 373–378
17. Petri CA (1962) Fundamentals of a theory of asynchronous information flow. In: IFIP congress, pp 386–390
18. Ramchandani C (1974) Analysis of asynchronous concurrent systems by timed Petri Nets. Technical report, Cambridge, MA, USA
19. Robert PH, Juanole G (2000) Modélisation et vérification de politiques d'ordonnancement de tâches temps-réel. In: 8ème colloque Francophone sur l'ingénierie des protocoles-CFIP'2000, 17–20 October, pp 167–182.
20. Roux OH, Déplanche AM (2002) A t-time Petri net extension for real time-task scheduling modeling. Eur J Autom (JESA) 36(7):973–987
21. Sha L, Abdelzaher T, Arzen KE, Cervin A, Baker T, Burns A, Buttazzo G, Caccamo M, Lehoczky J, Aloysious KM (2004) Real time scheduling theory: a historical perspective. Real-Time Syst J 28(2/3):101–155
22. Singhoff F, Legrand J, Nana LT, Marcé L (2004) Cheddar: a flexible real time scheduling framework. ACM Ada Lett J 24(4):1–8
23. Xu D, He X, Deng Y (2002) Compositional schedulability analysis of real-time systems using time Petri Nets. IEEE Trans Softw Eng 28:984–996