A FORMAL METHOD FOR SCHEDULING ANALYSIS OF A PARTITIONED MULTIPROCESSOR SYSTEM
Dynamic Priority Time Petri Nets

Walid Karamti, Adel Mahfoudhi, Yessine Hadj Kacem and Mohamed Abid
CES Laboratory, ENIS Soukra km 3,5, University of Sfax, B.P.:w 1173-3000 Sfax, Tunisia

Keywords: Real-time System, Petri Nets, dPTPN, Scheduling Analysis, Least Laxity First.

Abstract: In order to examine whether the timing constraints of a Real-Time application are met, we propose an extension of Time Petri Nets model that takes into account the scheduling of a set of tasks distributed over a multiprocessor architecture. This paper is concerned with dynamic Priority-driven scheduling, whose policy is known to be supported by a new formalism called dynamic Priority Time Petri Nets (dPTPN). Its ultimate objective is to show how to deal with the Least Laxity First Scheduling policy with a set of periodic independent tasks. Besides, the use of dynamic priorities gives a determinism aspect to the model in which a crossing of concurrent transitions exists. Therefore, the execution of the model is accelerated and the number of accessible states is decreased.

1 INTRODUCTION

Real-time systems (RTS) are characterized by complex applications that require powerful architectures to satisfy them. The trend is to use multiprocessor architectures, among which we can distinguish the presence of two families of multiprocessor scheduling. The first is the family of global scheduling in which each task of application can migrate among the processor resources to be executed. The major drawback of the global family is the absence of an optimal scheduling algorithm (Kwang and Leung, 1988). The second family is the partitioned scheduling, in which one task of the application can be executed on a single processor resource. It is to bring the multiprocessor scheduling problem in uniprocessor scheduling problems in which there exist optimal scheduling algorithms (Liu and Layland, 1973). The family of partitioned scheduling involves two steps; a step of assigning tasks to processors, and another for analyzing the scheduling of each partition (Sha et al., 2004). It is important to detect errors of scheduling as early as possible in order to minimize the costs for its correction. The analytical techniques used are based on the analytical work of Liu and Layland (Liu and Layland, 1973).

The specification with formal methods could be able to reduce the problem impact. In fact, it represents a recent research area in order to develop a consistent approach. Starting from an abstract modeling that takes into account a set of constraints, the main objective of this technique is to determine and check the system properties.

Many varieties of formal methods exist and the choice of the appropriate one depends on the characteristics of the system and the properties to be checked. Deductive reasoning and model checking are two major categories of proving methods. Using the deductive method, a specification gives an abstract description of the significant behavior of the required system. This behavior is checked for the defined implementation through the proving of theorems which constitute the rules of refinement. The major limitation of this category is that the user interacting with the prover must be able to guide him.

1.1 Related Work

The technique of model checking is of an irrefutable advantage, allowing early and economical detection of errors at an early stage of the design process. This explains the growing popularity it enjoys in the industrial world.

Aiming at developing a system model, designers must ensure a sufficient accuracy to preserve all the properties for check and a level of abstraction appropriate not to penalize the analysis phase. There are many formalisms proposed for modeling real-time

Karamti W., Mahfoudhi A., Hadj Kacem Y. and Abid M..
A FORMAL METHOD FOR SCHEDULING ANALYSIS OF A PARTITIONED MULTIPROCESSOR SYSTEM - Dynamic Priority Time Petri Nets.
DOI: 10.5220/0003809103170326
In *Proceedings of the 2nd International Conference on Pervasive Embedded Computing and Communication Systems* (PECCS-2012), pages 317-326
ISBN: 978-989-8565-00-6

systems. Each one has its specific characteristics, more or less relevant in a specific design and according to the type of analysis considered.

Time, parallel processing and synchronization are primordial characteristics for scheduling analysis. Compared with other formalisms, the Time Petri Nets (TPNs) (Merlin, 1974) can easily support these features without leading to combinatorial explosion. Therefore, TPNs face problems when modeling a scheduling policy. Indeed, TPNs is a non-deterministic formalism because when two or more transitions are enabled it is not specified which one has to fire. In reverse, the scheduling algorithm always chooses an event from the available ones according to a defined policy. Several extensions have been created to support scheduling policy by adding priorities to transitions.

It should be born in mind that the absence of a TPNs extension supports dynamic priorities. That is why we detail the extensions with static priority used for scheduling analysis.

Roux (Roux and Déplanche, 2002) presented an STPN (Scheduling Timed Petri Nets) to analyze periodic tasks on a multiprocessor architecture. The priorities were introduced by the inhibitor arcs to support a fixed priority driven scheduling policy such as RM (Rate Monotonic) (Liu and Layland, 1973). The contribution of his proposal lies in the calculation of a reduced state space compared to that evoked by (Berthomieu and Diaz, 1991). Such proposal was improved by (Lime and Roux, 2004) and (Lime and Roux, 2009) to support the tasks with variable time execution.

The STPN (Roux and Déplanche, 2002) adds constraints on the crossing of the transitions to check the respect for the firing interval. Therefore, the check of these constraints is a new dimension added to the problem of scheduling analysis.

Berthomieu (Berthomieu et al., 2006)in his turn utilized the inhibitor arcs to introduce the notion of priorities within the Petri nets through the extension of PrTPNs (Priority Time Petri Nets). His proposal was based on a method of temporal analysis of the network. Indeed, from a sequence of non-temporal transitions, his method was to recover the possible durations between the firing of transitions in order. The durations are the solutions of a linear programming problem.

Both of PrTPN and STPN are TPNs extensions, so they retain these properties, i.e., the indeterminacy of the date of crossing of an event. However the modeling of a scheduling policy of a hard real-time system requires that all dates used are determined. The priority is modeled through the inhibitors arcs added as new components to those of PNs. The RTS modeling with Petri nets gives rise to the models that are often complex. Moreover, the addition of an inhibitor arc makes the model more complex and therefore the extraction of properties becomes more difficult.
A new extension PTPN (Priority Time Petri Nets) was proposed in (Kacem et al., 2010), in which a crossing date is associated with each temporal event. In fact, a transition is valid when the clock shows the date of firing. In addition, PTPN uses a new method of priorities integration to address the problem of transitions conflict. In this method, a priority is inserted on the input arcs of the dependent transitions (Kacem et al., 2010).Moreover, this method allows to master the complexity of the PTPN model by eliminating the use of another component, such as inhibitor arcs, to specify priorities. To control the size of the PTPN model, a hierarchical modeling was proposed in (Mahfoudhi et al., 2011).

1.2 Contribution and Outline of the Paper

Most of the Time Petri Nets extensions dedicated for scheduling analysis such as PrTPN (Berthomieu et al., 2006), STPN (Roux and Déplanche, 2002) and PTPN (Kacem et al., 2010) use a fixed priority. However, the use of a fixed priority allows the covering of a reduced space of RTS such as periodic tasks and fixed Priority-driven scheduling strategies.

On a single processor platform, the LLF (Least Laxity First) scheduling policy, a dynamic Priority-driven scheduling, is known as the most general algorithm on RTS. It is proven as an optimal preemptive scheduling algorithm (Goossens et al., 2004). However, its features upon multiprocessor platforms have been little studied so far.

In this paper, we account for the scheduling analysis of multiprocessor system via a new extension of Time Petri Nets. The major advantage of the proposed extension is that it gives a solution to the modeling and analysis of a dynamic priority-driven scheduling policy (LLF) taking into account periodic independent system tasks partitioned over a multiprocessor platforms.

The present paper is organized as follows. Firstly, the definitions and execution semantics of the proposed dynamic Priority Petri Nets (dPTPN) are overviewed in section 2. Next, section 3 shows how our extended formalism is used for specifying the scheduling analysis problem through experiments. Finally, the proposed approach is briefly outlined and future perspectives are given.

2 DYNAMIC PRIORITY TIME PETRI NETS: dPTPN

Our challenge is the integration of a dynamic-Priority driven scheduling policy in Petri Nets. This strategy is based on the dynamic calculation of priorities. Indeed, the value of a priority changes at runtime.

In the standard PN, the events (transitions) had the same grade of emergency. Thus when transitions conflict, there are no favorable transition to cross before the others. Accordingly, the semantics of the standard PN does not support a scheduling policy.

A real-time scheduling policy is characterized by a sequence of timed events and no events conflict. To reach our goal, we will integrate these two features in PN. To do so, we distinguish between temporal events and concurrent events that are sources of conflict. We propose a new extension of PN with two types of transitions T (temporal transitions) and T_{cp} (compound transition Fig. 1).

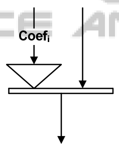

Figure 1: T_{cp} compound.

The latter is all about a transition with a preprocessing that precedes the crossing to calculate its priority. Indeed, if two Tcp transitions are enabled and share at least a place in entry then the preprocessing is made to determine the transition which will be fired, with a priority changing according to the state of the network described by the marking M. By analogy in real-time scheduling with dynamic priorities such as LLF, the priority changes according to laxity.

In this section, we present the syntax of the new Petri Net extension (dynamic priority Petri Nets $dPTPN$) as well as its firing semantics. The simulation of the sequences of firing is then explained.

2.1 Syntax

A Petri Nets (Petri, 1962) can be defined as 4-uplet :

$$PN = \langle P,\ T,\ B,\ F\rangle \qquad (1)$$

, 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 and defined by : $M : P \mapsto \mathbb{N}$.
The $dPTPN$ is defined by the 7-uplet :

$$dPTPN = \left\langle PN, T_{cp}, T_f, B_{T_{cp}}, F_{T_{cp}}, coef, M_0 \right\rangle \qquad (2)$$

(1) PN: is a Petri Net;
(2) $T_{cp} = \left\{T_{cp_1}, T_{cp_2}, \cdots, T_{cp_k}\right\}$: is a finite set of compound transition $k > 0$;
(3) $T_f : T \mapsto \mathbb{Q}^+$ is the firing time of a transition.
$\forall t \in T$, t is a temporal transition $\iff T_f(t) \neq 0$.If $T_f(t) = 0$ then t is an immediate transition. Each temporal transition t is coupled with a local clock ($Hl(t)$),with $Hl : T \longrightarrow \mathbb{Q}^+$.
(4) $B_{T_{cp}} :\ (P \times T_{cp}) \mapsto \mathbb{N}$ is the backward incidence function associated with compound transition;
(5) $F_{T_{cp}} :\ (P \times T_{cp}) \mapsto \mathbb{N}$ is the forward incidence function associated with compound transition;
(6) $coef :\ (P \times T_{cp}) \mapsto \mathbb{Z}$ is the coefficient function associated with compound transition;
(7) M_0 : is the initial marking;

2.2 Semantics

To deal with the problem of the state space explosion, it is worthwhile to mention the contribution of the methods considered as partial order (Antti, 1989) (Kimmo, 1994) (Buy and Sloan, 1994) building on a relation of equivalence between various sequences of possible firings, starting from the same state. In fact, when two sequences are found to be equivalent, then only one of them is selected. This relation of equivalence is based on the notion of independence of transitions. Two transitions are independent if they are not in the same neighborhood (see eq. 9).
Based on the approach of partial order and the T_{cp} compound, $dPTPN$ relies on a well-defined strategy for the firing during a conflict of transitions. The strategy builds on the construction of a sequence of the highest priority and valid transitions dFT_s. For the firing of a marking M, $dPTPN$ offers a machine of firing called *dPTPN Firing Machine dPFM*, whose mechanism is described in Fig. 2. The entry of the machine $dPFM$ is a marking M of the $dPTPN$ network. For each entry, the machine builds the dFT_s set, which consists of two sub-sets FT_s and $FT_{s_{T_{cp}}}$, which in turn present the firing transitions of T and those of T_{cp}, respectively. When it is about a marking dead-end, the dFT_s set is empty (FT_s is empty and $FT_{s_{T_{cp}}}$ is empty) and the $dPFM$ machine stops.

The modeling of an RTS with $dPTPN$ distinguishes between the temporal and immediate events. To do so, the machine $dPFM$ supplies a chain of processing. Indeed, the machine, firstly, determines the valid temporal transitions set VT_s from the FT_s set,

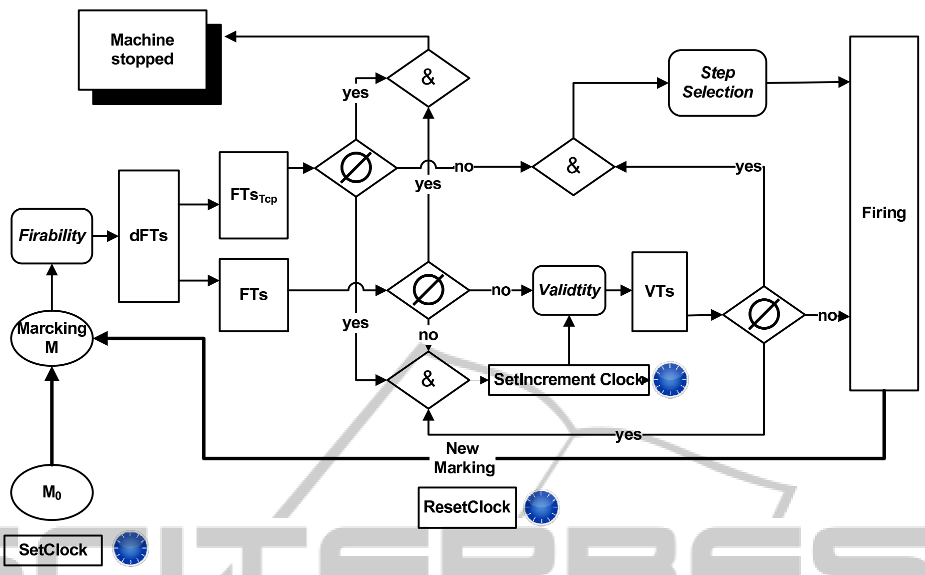

Figure 2: dPTPN firing machine.

and secondly, the VT_s is analyzed. The machine fires any valid transition before passing to the following stage and for each firing the dFT_s is reconstructed (FT_s and $FT_{s_{Tcp}}$, respectively). The last stage consists in solving the problem of conflict by choosing among the compound transitions with the highest priority that will be fired.

All immediate transitions must be crossed before the analysis of $FT_{s_{Tcp}}$. In fact, the immediate transitions are more urgent and their firing can give birth to a marking that presents a new conflict of transitions.

2.2.1 Firability

dFT_s presents the enabled transitions set. In fact, dFT_s is the union of the enabled temporal transitions set FT_s and the enabled compound transitions set $FT_{s_{Tcp}}$.

$$dFT_s = FT_s \cup FT_{s_{Tcp}}. \tag{3}$$

$$let\ t \in T, t \in dFT_s \Leftrightarrow t \in FT_s \vee t \in FT_{s_{Tcp}} \tag{4}$$

$$\text{with} \begin{cases} FT_s = \{t \in T / B(\,.\,,t) \leq M\} \\ FT_{s_{Tcp}} = \{t \in T / B_{Tcp}(\,.\,,t) \leq M\} \end{cases}$$

For each subset FT_s and $FT_{s_{Tcp}}$ is associated with a function indicator χ_{FT_s} and $\chi_{FT_{s_{Tcp}}}$.

$$\chi_{FT_s} : T \longrightarrow \{0,1\} \tag{5}$$

$$x \longmapsto \begin{cases} 1\ if\ x \in FT_s \\ 0\ otherwise \end{cases}$$

$$\chi_{FT_{s_{Tcp}}} : T \longrightarrow \{0,1\} \tag{6}$$

$$x \longmapsto \begin{cases} 1\ if\ x \in FT_{s_{Tcp}} \\ 0\ otherwise \end{cases}$$

2.2.2 Validity

A transition is valid if it is enabled and respects its firing date. We should bear in mind that T_{cp} is an immediate transition. Therefore, we define the Valid Transition Set (VT_s) as a subset of FT_s ($VT_s \subseteq FT_s$).

$$VT_s = \{t \in FT_s / Hl(t) = T_f(t)\} \tag{7}$$

χ_{VT_s} is the function indicator of VT_s with:

$$\chi_{VT_s} : FT_s \longrightarrow \{0,1\} \tag{8}$$

The main advantage of utilization of the T_{cp} component is to solve the problem of transition conflict. The best way is to select the T_{cp} transition having the highest priority.

2.2.3 Step Selection

$\forall T_{cp_i} \in T_{cp}$, T_{cp_i} is selected if and only if it is enabled and has the highest priority compared with its neighborhood. First of all we present the neighborhood of transition T_{cp_i}:

$$\forall T_{cp_1}, T_{cp_2} \in T_{cp}, T_{cp_1}\ is\ a\ neighbor\ of\ T_{cp_2}$$

$$\Leftrightarrow \exists p \in P\ such\ that\ B_{T_{cp}}(p, T_{cp_1}) \neq 0 \wedge B_{T_{cp}}(p, T_{cp_2}) \neq 0 \tag{9}$$

We consider a matrix N_e to indicate the neighborhood of all transitions T_{cp} compared to each place P:

$$N_e : T_{cp} \times P \longrightarrow \{0,1\} \tag{10}$$

$$(t_{cp}, p) \longmapsto \begin{cases} 1\ if\ B_{T_{cp}}(p, t_{cp}) > 0 \\ 0\ otherwise \end{cases}$$

Three steps are applied to select the transition that has the highest priority. The first step is the calculation of the priority for each enabled transition T_{cp}. In fact, the priority depends on the state of the dPTPN model and the matrix $coef$. Each T_{cp} calculate its priority using the scalar product of $coef$ matrix with the marking vector M.

$$Prio : FT_{s_{T_{cp}}} \longrightarrow \mathbb{Z} \qquad (11)$$
$$t_{cp} \longmapsto \langle coef\,(.,t_{cp}) \mid M \rangle$$

In the second step, we mark the corresponding priorities to each T_{cp} neighborhood. More precisely we define the following function:

$$Prod :\ FT_{sT_{cp}} \times P \mapsto \mathbb{Z} \qquad (12)$$
$$(t_{cp}, p) \longmapsto Prio\,(t_{cp})\, N_e\,(t_{cp}, p)$$

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

$$Min : P \longrightarrow FT_{s_{T_{cp}}} \qquad (13)$$
$$p \longmapsto t_{cp}$$
$$\text{with } \{\forall t_{cp_i} \neq t_{cp},\ Prod\,(t_{cp}, p) < Prod\,(t_{cp_i}, p)\}$$

Finally, the $FT_{s_{T_{cp}}}$ must be updated to present only the selected transitions: $\forall t_{cp} \in T_{cp}, \forall p \in P$,

$$Prod(t_{cp}, p) \neq 0 \wedge t_{cp} \neq Min(p) \Longrightarrow \chi_{FT_{s_{T_{cp}}}}(t_{cp}) = 0 \qquad (14)$$

2.2.4 Firing

We define the firing of a given FT vector. In $dPTPN$, FT can be an VT_s vector or an $FT_{s_{T_{cp}}}$ vector:

$$\forall FT \in \left\{VT_s, FT_{s_{T_{cp}}}\right\}, Firing\,(FT) \Longrightarrow$$

$$\begin{cases} FT = VT_s \\ \Leftrightarrow M' = M + \sum_{t \in FT} \left(F\,(.,t) - B\,(.,t)\right) \\ \\ FT = FT_{s_{T_{cp}}} \\ \Leftrightarrow M' = M + \sum_{t \in FT} \left(F_{T_{cp}}\,(.,t) - B_{T_{cp}}\,(.,t)\right) \end{cases} \qquad (15)$$

2.2.5 Clocks Management

According to dPTPN, the clocks management abides some rules, which are:

Rule1 (Set): When a transition is enabled, for its first time, then its matching clock is activated and initialized with zero.

Rule2 (Reset): After firing VT_s, the clocks of valid transitions will reset.

Rule3 (Increment Clock): The clock of a transition is incrementing if the transition is enabled, no valid transition exists and the $FT_{s_{T_{cp}}}$ is empty.

$SetIncremetClock()$ is activated $\Longleftrightarrow$

$$FT_s \neq \oslash \wedge FT_{s_{T_{cp}}} = \oslash \wedge VT_s = \oslash \qquad (16)$$

2.2.6 dPTPN State Graph

In $dPTPN$, a state is a node composed of (M, tmp), where M is a marking and tmp is the time for its firing. The state graph SG is defined by: $SG = \langle S, A \rangle$ with:

- S: is a set of state;
- $A : S \times S \longrightarrow dFT_s$: presents the arcs connecting the reachable states. The connection arcs will be established only when two markings of two states are reachable via dFT_s vector.

The process of construction of the states graph corresponds to the execution of the Algorithm 1. Indeed, it deals with a repetitive execution of the $dPFM$ machine and for every iteration a new state is constructed.

During the first tic of the clock associated with

Algorithm 1: State graph construction.

```
 1: $M$: Marking
 2: $dFT_s$: enabled transitions set
 3: $FT_s$: enabled temporal transitions set
 4: $VT_s$: valid temporal transitions set
 5: $Hl$: clocks set
 6: $FT_{s_{T_{cp}}}$: enabled compound transitions set
 7: initializeMarking($M$)
 8: SetClock(Hl)
 9: StateConstruction($M$)
10: while M $\prec$ B and M $\prec B_{T_{cp}}$ do
11:     Firability($dFT_s$)
12:     $FT_s \leftarrow temporalTransition\,(dFT_s)$
13:     $FT_{s_{T_{cp}}} \leftarrow CompoundTransition\,(dFT_s)$
14:     if $FT_s \neq \oslash$ then
15:         $VT_s \leftarrow$ Validity($FT_s$)
16:         if $VT_s \neq \oslash$ then
17:             Firing($VT_s$)
18:             StateConstruction($M$)
19:             ResetClock($Hl\,(VT_s)$)
20:         else if $FT_{s_{T_{cp}}} = \oslash$ then
21:             SetIncrementClock($Hl\,(FT_s)$)
22:         end if
23:     else if $FT_{s_{T_{cp}}} \neq \oslash$ then
24:         StepSelection($FT_{s_{T_{cp}}}$)
25:         Firing($dFT_s$)
26:         StateConstruction($M$)
27:     end if
28: end while
```

$dPTPN$, the marking M is initialized by the *initializeMarking(M)* function and the first state is built by the function *StateConstruction(M)*. The process of construction starts with applying an iterative work. For each iteration, the machine $dPFM$ begins with the determination of the fired transitions in the vector dFT_s by applying the function $Firability(dFT_s)$. Vectors FT_s and $FT_{s_{T_{cp}}}$ are initialized from dFT_s to distinguish between the temporal transitions and the T_{cp}

transitions.

The validation of the temporal transitions is analyzed via the function *Validity(FT_s)*. To check the existence of the valid transitions, a conditional structure appears to fire them at once. However, the absence of the valid transitions allows us to check the existence of the compound transitions. The presence of the transitions in $FT_{s_{T_{cp}}}$ requires a selection of the highest priority to fire it. Any firing leads to an update of the marking M and a construction of a new state to connect it with the precedent one through a weight arc dFT_s.

The absence of the valid transitions and the compound transitions leads to an incrementation of the global clock of the $dPTPN$ by calling the function *SetIncrementClock()*. This process is repeated as long as the marking M satisfies at least a transition. The Process ends as soon as it is in a marking dead-end or during a reset of the marking.

3 MODEL CONSTRUCTION

The RTS Ω is defined by the 3-uplet:

$$\Omega = \langle Task,\ Proc,\ Alloc \rangle \tag{17}$$

with:
(1) Task : is a finite set of real-time tasks with each $Task_i \in Task$ determined by

$$Task_i = \langle R_i,\ P_i,\ D_i,\ C_i \rangle \tag{18}$$

- R_i: the date of the first activation.
- P_i: the period associated with the task.
- D_i: the deadline associated with the task, we consider an RTS with $P_i = D_i$.
- C_i: the execution period of the task for the P_i period.

(2) $Proc$: a finite set of processors.
(3) $Alloc$: $Task \mapsto Proc$, a function which allocates a task to a processor. $Alloc$ is a surjective function. In fact a processor is allocated to at least one task. But a task must be assigned to only one processor.
$\forall t_1 \in Task,\ \forall P_1, P_2 \in Proc,$

$$Alloc(t_1) = P_1 \wedge Alloc(t_1) = P_2 \Rightarrow P_1 = P_2 \tag{19}$$

To model the system Ω with the new $dPTPN$ extension, we specify each component of Ω. The model of scheduling analysis is described through the communication between the various $Task$ and $Proc$ components. Each processor is modelled by a simple place, whose marking describes the state of the resource. The presence of a mark indicates that the resource is free and ready to execute a task. The absence of the mark indicates that the resource is occupied by a system task Ω.

3.1 Task Model

The modeling can be divided into two major patterns. The first one pertains to the notion of creation, activation and deadline, as for the second, it describes the concept of execution task.

3.1.1 Creation, Activation and Deadline

At first, a task $Task_i \in Task$ is created. For a date ”Ri”, the creation is made, leading to the passage of the task towards the created state. So, the period ”Pi” starts. As shown in Fig. 3, each state is modelled by a place and every event by a transition. When it is about an event accompanied with a date of progress, the transition will take a date of firing similar to that of the event. When it is about a non-created state, it is presented by a marked place ($UnCreated$). The creation is modelled by a temporal transition $Creation$ which takes a date of ”Ri” firing.

When the local clock of $Hl(Creation)$ indicates the date of firing R_i, the transition is then valid and the firing allows to register a mark in the place $Created$ and P_i tokens in $RemainingPeriod$. Indeed, the marking of the place $Created$ shows that the task is created and P_i marks present the remaining duration before the deadline. For each tic of the global clock of the $dPTPN$, the transition $IncPeriod$ is validated. Besides, its firing leads to a decrementation of the remaining duration before the deadline and an incrementation of the exhausted time of the period. When Pi tokens are moved towards the place $ElapsedPeriod$, the period is exhausted and a new period must be activated. The event of activation is modelled by a transition $Restart$, whose firing activates a new period by depositing P_i marks in $RemainingPeriod$ and the creation of a new instance of the task by putting down a token in the place $Created$.

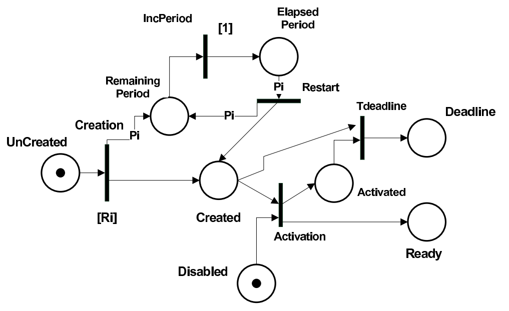

Figure 3: Creation, activation and deadline.

We present a first scenario in which the first activation of the task $Task_i$ exists. We will then introduce a scenario of deadline event provided with the mechanism of its detection by the model $dPTPN$.

The place *Created* indicates the presence of a mark. To model the activation of the first instance of the task, we use a mutual exclusion described by two places *Disabled* and *Activated*. According to the initial scenario, the first instance of the task is deactivated, where from the presence of a token in *Disabled*. The current marking allows the firing of the transition *Activation* and the firing allows the supply of a mark in places *Ready* and *Activated*. The current state proves that the current instance of the task is activated and lends itself to be run on the processor resource.

With regard to the second scenario, it describes the activation of a new period while the old instance of the task has not finished its execution yet. The model presents a mark in the place *Created* and another in the place *Activated*. This marking allows the firing of the transition *Activation* for the activation of the new instance and the transition $Tdeadline$ to indicate that the task overtook its deadline. The firing of $Tdeadline$ allows the registration of a mark in the place *Deadline* and the blocking of the entire model $dPTPN$ of the task.

3.1.2 Execution

The execution of a task is described by two main events, the first of which pertains to the allocation of the resource processor and the second represents the execution of the task on the processor.

The resource processor is a resource shared between the various tasks of the same partition, and for a given moment, a single task occupies it. The allocation of the processor depends on the used scheduling strategy. In our study we are interested in the strategy based on LLF.

With $dPTPN$, each task calls for the resource processor if and only if it presents a mark in the place *Ready*. So, the event of allocation is modeled by a transition $allocation$ of the type T_{cp}. The processor will be attributed to the task having the transition $allocation$ with the highest priority (having the least laxity). Indeed, the allocation is modeled by a registration of a mark in the place *getProc*. Fig. 4 illustrates two requests of allocation of the processor $P1$ by two tasks (Alloc(Task1)=Alloc(Task2)=P1).

Fig. 4 shows that for each task, there are 4 places and a T_{cp} transition type as well as a place for modeling the processor. Both transitions $T1allocation$ and $T2allocation$ that are fired by the current marking are in conflict because of the shared place $P1$. The priorities of the transitions will be calculated according to the LLF policy. In fact, the LLF makes its schedul-

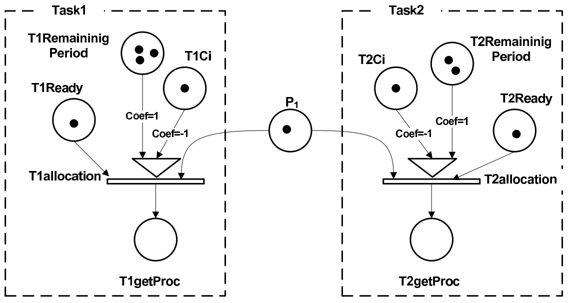

Figure 4: Allocation processor.

ing decisions based on the maximum time span that a request can tolerate before being picked up for execution.

The laxity of a request at any given moment is the time span that it can tolerate before the time it has to be picked up for execution; otherwise its deadline will definitely be missed. As a request is being executed, its laxity does not change. However, the laxity of a task decreases as the request waits (either in the ready-to-run queue or elsewhere).Its chances of being overrun increase. When the laxity of a task becomes zero, it can no longer tolerate waiting.

The laxity of a task is a dynamic value that changes as the circumstances change. Four values affect its computation:(1) the absolute time of the moment when we want to compute the laxity of the task, t,(2) the absolute value of the deadline of this request, D_i,(3) the execution time of this request, C_i, and (4) the execution time that has been spent on this request since it was generated, e_i. The following is the simple formula for computing the laxity of a request with the aforementioned values.

$$L = (D_i - t) - (C_i - e_i) \tag{20}$$

In Fig. 5, the $(D_i - t)$ value is presented with the marking of the place ”RemainingPeriod” and the $(C_i - e_i)$ is presented with the marking of the place ”C_i”.

The main interest of ”$coef$” matrix is to provide a solution for presenting the arithmetic operators. Indeed, to model the equation L with dPTPN, we intercalate the coefficient ”1” on the arc connecting the place ”RemainingPeriod” and the T_{cp} and ”2” on the arc between the place ”C_i” and ”T_{cp}”.

The marking of the model Fig. 4 presents 4 marks in $T_1RemainingPeriod$ and 2 marks in $T_2RemainingPeriod$. The $T_2allocation$ transition is the most priority because it has the least laxity (L=2-1=1). The firing of $T2alloctaion$ involves the extraction of two marks of $P1$ and T_2Ready and a registration of a mark in $T_2getProc$ to indicate that the processor $P1$ is attributed to $task_2$. The place T_1Ready always remains marked because $task_1$ is always pend-

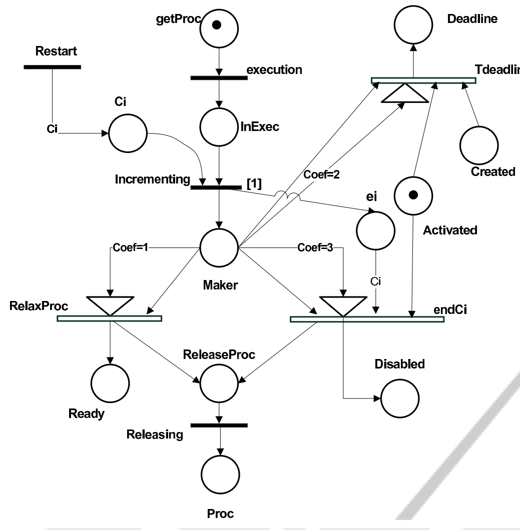

Figure 5: Execution.

ing to the processor P1.

As soon as the marking of the model indicates the presence of a mark in the place *getProc* (Fig.5), the task begins the execution by firing the immediate transition *execution*.

It is worthwhile to mention that the interest of the present paper is the preemptive systems. Indeed, to manage the modeling of this type of problem, we propose that each task occupies the processor during only one unit of time then releases it. The corresponding $dPTPN$ model is described by a temporal transition $incrementing$ with a date of firing equal to 1. The firing of this event allows not only the removal of a mark from the place *Ci* and one from the place $InExec$ but also the registration of a mark in the place *maker* as well as the incrementation of the tokens of *ei* by another mark. Such incrementation indicates that the task ended a unit of execution during the period *Pi*.

It should be noted that there is an emergence of three events *endCi*, $endDeadline$ and *RelaxProc*. We consider the scenario when the three events are enabled.

The place *maker* is a shared place between those events. So they are modeled with T_{cp} transitions and are in the same neighborhood. We must specify the priority of each transition for solving the conflict. The current scenario shows that the task has completed its *Ci* units and the deadline is triggered. In the scheduling analysis, this scenario is not considered as non-schedulable task. Thus, the priority of the *endCi* transition is higher than *endDeadline*. To do so, we attribute the coefficients ”3” to the input arc of *endCi* and ”2” to the input arc of ”endDeadline”.

The non-schedulable system is described when the *endDeadline* and *RelaxProc* are enabled. It means that a new period is triggered when the last period is still activated. Since, the $endDeadline$ must be fired immediately, we must attribute the higher priority to it and a smaller priority to *RelaxProc* (coef=1).

3.2 Case Study

In the present section, we introduce the technique of how $dPTPN$ solve the scheduling analysis problems of the real-time systems. It is through a case study that our $dPTPN$ extension is brought to the light.

In fact, we present a generic experiment which consists of a non-schedulable system. In the latter, we establish how the way how the evolution of $dPTPN$ model 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 three tasks running on two processors. Using definition 17, the specifications of the task characteristics as well as the allocation of the processors by the tasks are described as follows.

$Task = \{T1(0,16,16,4),\ T2(0,8,8,2),\ T3(0,4,4,2),\ T4(0,6,6,4),\ T5(0,7,7,4)\}$;
$Proc = \{P1,\ P2\}$;

$Alloc(T1) = P1 \quad Alloc(T4) = P2$
$Alloc(T2) = P1 \quad Alloc(T5) = P2$
$Alloc(T3) = P1$

The first stage consists in modeling of Ω with the $dPTPN$. As for the second step, it consists in placing the marks on $dPTPN$ constituents: a mark in each constituent ”Processor: P1, P2”. A mark in the places ”Uncreated” and ”Disabled” of ”Task : T1, T2, T3, T4, T5”. We also place 4, 2, 2, 4 and 4 marks on the places ”Ci” of tasks T1, T2, T3, T4 and T5 respectively.

At this level, let us recall that the strength of our $dPTPN$ extension is its capacity to determine step by step the valid scheduling sequence. In fact, the execution of the model based on $dPFM$ 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 supplied scheduling sequence is optimal because $dPTPN$ supports the optimal dynamic priority LLF strategy.

For a better presentation, we have detailed the execution of the model in Table 1. 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

Table 1: Model execution related to the case study.

Step	Time	dFT_s		Step	Time	dFT_s	
		FT_s	$dFT_{s_{Tcp}}$			FT_s	$dFT_{s_{Tcp}}$
0	0	Creating(T1,T2,T3,T4,T5)	−−	20	4	Restart(T3)	−−
1	0	Activation(T1,T2,T3,T4,T5)	−−	21	4	Activation(T3)	−−

2	0	⊘	Allocation(T1,T2,T3,T4,T5)	22	4	⊘	RelaxProc(T2,T5)
							EndCi(T2)

3	1	Execution(T3,T4)	−−	23	4	Releasing(T2,T5)	−−

4	1	Incrementing(T3,T4)
		IncPeriod(T1,T2,T3,T4,T5)
			−−	24	4	⊘	Allocation(T1,T3,T4)

5	1	⊘	RelaxProc(T3,T4)	25	4	Execution(T3,T4)	−−

6	1	Releasing(T3,T4)	−−	26	5	Incrementing(T3,T4)
						IncPeriod(T1,T2,T3,T4,T5)
							−−

7	1	⊘	Allocation(T1,T2,T3,T4,T5)	27	5	⊘	RelaxProc(T3,T4)
8	1	Execution(T3,T5)	−−	28	5	Releasing(T3,T5)	−−

9	2	Incrementing(T3,T5)
		IncPeriod(T1,T2,T3,T4,T5)
			−−	29	5	⊘	Allocation(T1,T3,T4)

10	2	⊘	RelaxProc(T3,T5)
			EndCi(T3)
				30	5	Execution(T3,T4)	−−

11	2	Releasing(T3,T5)	−−	31	6	Incrementing(T3,T4)
						IncPeriod(T1,T2,T3,T4,T5)
							−−

12	2	⊘	Allocation(T1,T2,T4,T5)	32	6	Restart(T4)	−−

13	2	Execution(T2,T4)	−−	33	6	⊘	RelaxProc(T3,T4)
							EndDealine(T4)
							EndCi(T3,T4)

14	3	Incrementing(T2,T4)
		IncPeriod(T1,T2,T3,T4,T5)
			−−	34	6	Releasing(T3,T4)	−−

15	3	⊘	RelaxProc(T2,T4)	35	6	⊘	Allocation(T1,T4,T5)
16	3	Releasing(T2,T4)	−−	36	6	Execution(T1,T5)	−−

17	3	⊘	Allocation(T1,T2,T4,T5)	37	7	Incrementing(T3,T5)
						IncPeriod(T1,T2,T3,T4,T5)
							−−

18	3	Execution(T2,T5)	−−	38	7	Restart(T5)	−−

19	4	Incrementing(T2,T5)
		IncPeriod(T1,T2,T3,T4,T5)
			−−	39	7	⊘	RelaxProc(T1,T4)
							EndDealine(T5)

number of tics 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 time 0 with the initial marking M_0. These two parameters present the necessary entries to launch the $dPFM$ machine. Such machine determines the high-priority t_{cp} and valid transitions t in the columns FT_s and $FT_{s_{Tcp}}$: Transition-name (Task-name). It is the case of ”step0”, in which the column FT_s presents ”Creation(T1, T2,T3)” and no enabled t_{cp} exists. The simultaneous firing of this set of events gives birth to a new marking and a new step ”step1”.

As for ”step2”, it presents no valid temporal event exist and the event ”Allocation(T1,T2,T3,T4,T5)” in the column $FT_{s_{Tcp}}$. This indicates that five tasks are ready to allocate the processors ”P1” and ”P2”. ”Allocation(T1)” and ”Allocation(T2)” and ”Allocation(T3)” belong to the same neighborhood. However, ”Allocation(T3)” has the lowest laxity (highest priority according LLF) (l=2) . Therefore, it will be fired before ”Allocation(T1)” and ”Allocation(T2)”. The transitions with the highest priority are presented with the red color for each $FT_{s_{Tcp}}$.

What is worthy to note is that step ”step38” presents the ”Restart(T5)” event. This indicates that the period $T5$ is provoked and a new $T5$ instance is created while the work of the last instance is not achieved. The firing and the passage to the step ”step39” brings about a new marking which replies to the event ”EndDeadline(T5)”. The firing of this event implies the blocking of the execution, signaling a temporal fault (deadline). The current marking describes the combination of tasks that causes this fault. It will be a useful feedback to the designer to change the allocation Task/Processor.

4 CONCLUSIONS

In this paper, we have presented a new method for scheduling analysis of periodic tasks running on multiprocessor architecture. The analysis technique is based on proposed dynamic Priority Time Petri Nets dPTPN. Compared to the existing research work, the salient characteristic of novelty in $dPTPN$ consists in the attribution of a dynamic priority via a compound transition. Therefore $dPTPN$ supports a dynamic Priority-driven policy (LLF) and its semantics isolate the conflict of enabled transitions. Rather than presenting a solution for the problems of confusion, $dPTPN$ Firing Machine accelerates the $dPTPN$ evolution by applying temporal filtering for temporal transitions and priority filtering for T_{cp} transitions. In regular PN, the firing of a transition requires the identification of the new set of enabled transitions. However, with $dPFM$, this set is established only after the firing of the old one. It is also guided by dynamic priorities. Consequently, starting from a marking M_0, the $dPFM$ fires simultaneously valid temporal transitions and the T_{cp} having the highest priority then returns the new marking M.

In the current paper, we deal with a partitioning strategy of multiprocessors scheduling. This technique is concerned as a static scheduling method. The distribution of tasks on processors is often done through a tool based on heuristics. When the scheduling analysis technique, i.e. dPTPN, detects a non-schedulable system, then the tool generates a new solution. This process causes a waste of time generating problems during implementing of the scheduling strategy. However, dPTPN indicates the exact description of the non-schedulable sequence and just changed this sequence to obtain a valid scheduling. This presents our challenge in future research work. We are also interested in including performance analysis in the verification of RTS and planning to integrate $dPTPN$ design patterns ranging building blocks for an easy model construction and interpretation.

REFERENCES

Antti, V. (1989). Stubborn sets for reduced state space generation. In *Applications and Theory of Petri Nets*, pages 491–515.

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

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

Buy, U. and Sloan, R. (1994). Analysis of real-time programs with simple time petri nets. In *ISSTA '94: Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, pages 228–239, New York, NY, USA. ACM.

Goossens, J., Richard, P., Richard, P., and Bruxelles, U. L. D. (2004). Overview of real-time scheduling problems. In *Euro Workshop on Project Management and Scheduling*.

Kacem, Y. H., Karamti, W., Mahfoudhi, A., and Abid, M. (2010). A petri net extension for schedulability analysis of real time embedded systems. In *PDPTA*, pages 304–314.

Kimmo, V. (1994). On combining the stubborn set method with the sleep set method. In Valette, R., editor, *Application and Theory of Petri Nets 1994: 15th International Conference, Zaragoza, Spain, June 20–24, 1994, Proceedings*, volume 815 of *Lecture Notes in Computer Science*, pages 548–567. Springer-Verlag, Berlin, Germany. Springer-Verlag Berlin Heidelberg 1994.

Kwang, S. H. and Leung, J.-T. (1988). On-line scheduling of real-time tasks. In *IEEE Real-Time Systems Symposium*, pages 244–250.

Lime, D. and Roux, O. (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*, pages 187–196, Washington, DC, USA. IEEE Computer Society.

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

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

Mahfoudhi, A., Hadj Kacem, Y., Karamti, W., and Abid, M. (2011). Compositional specification of real time embedded systems by priority time petri nets. *The Journal of Supercomputing*, pages 1–26. doi 10.1007/s11227-011-0557-9.

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

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

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

Sha, L., Abdelzaher, T., Arzén, K. E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., and Mok, K. A. (2004). Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28:101–155. 10.1023/B:TIME.0000045315.61234.1e.