Using dynamic Priority Time Petri Nets for scheduling analysis via Earliest Deadline First policy

Walid Karamti
CES Laboratory, ENIS Soukra km 3,5, University of Sfax
B.P.:w 1173-3000 Sfax TUNISIA
Email: walid.karamti@ceslab.org

Adel Mahfoudhi
CES Laboratory, ENIS Soukra km 3,5, University of Sfax
B.P.:w 1173-3000 Sfax TUNISIA
Email: adel.mahfoudhi@fss.rnu.tn

Yessine Hadj Kacem
CES Laboratory, ENIS Soukra km 3,5, University of Sfax
B.P.:w 1173-3000 Sfax TUNISIA
Email: yessine.hadjkacem@ceslab.org

***Abstract*—In a previous paper [14], we investigated the expressiveness of Time Petri Nets extended with dynamic Priorities and showed that it is able to analyze the schedulability of a partitioned real-time system over a multiprocessor architecture. The scheduling analysis is proven via the Least Laxity First (LLF) policy and a set of independent tasks. In the same vein, we investigate in the present paper the applicability of the proposed approach for the Earliest Deadline First (EDF) policy to reduce the cost of preemption and we tackle the dependent tasks problem. Through an experiment, we present the capacity of our approach to detect the temporal faults in scheduling.**

***Keywords*-Real-Time System; Scheduling analysis; Earliest Deadline First; Petri Nets; dPTPN**

I. Introduction

Real-time systems (RTS) are now omnipresent in modern societies in several domains such as avionics, control of nuclear power station, multimedia communications, robotics, systems on chip, etc. Hence, RTS are characterized by complex applications that require powerful architectures to satisfy them. In practice, such architectures can be specified via a powerful single-processor or a multiprocessor architecture composed of a set of low processors. For the same power, a multiprocessor architecture is much cheaper, which represents an economical motivation for researchers to target such kind of architectures.

It is noteworthy to mention that two major families of multiprocessor scheduling can be distinguished. The first one is the global scheduling family in which each application task can migrate among the processor resources to be executed. However, the cost of migration is so important and without an optimal scheduling algorithm [16]. As for the second one, it is the partitioned family, which is based on sharing tasks on different processors without migration. In this family, the optimality can be dedicated because the multiprocessor scheduling is reduced to a single-processor scheduling where optimal algorithms exist [19]. This family includes two events, the first of which is assigning tasks to processors and the second is analyzing the scheduling of each partition [20]. An important key challenge is to detect the errors of scheduling as early as possible in order to minimize the costs for its correction.

The specification with formal methods has proven to be useful for making the development process reliable. 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. 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 systems. Each one has its specific characteristics, more or less relevant in a specific design and according to the type of analysis considered.

Particularly, Petri Nets (PN) present an adequate model checking thanks to their great expressivity dynamic vision and executable aspect. Besides, they have been successfully used in RTS specification. Thus, it is interesting to use the PN for the scheduling analysis of an RTS running on a multiprocessor system.

A. Related work

In this section, we focus on Petri Nets dedicated to scheduling analysis. However, the scheduling analysis of multiprocessor systems is a recent research area, which explains the scarcity of Petri Nets dealing with it.

Via the Time Petri Nets extension (TPNs) [22], PN was able to specify the primordial characteristics of an RTS such as time, parallel processing and synchronization. Therefore, TPNs face problems when modeling a scheduling policy. In fact, TPNs is defined as a non-deterministic formalism because when two events are enabled, it is not specified which one has to be crossed. In reverse, the scheduling policy is known with its capacity to solve the events conflict.

Several extensions have been proposed to support scheduling policy by adding priorities to transitions.
Roux [25] 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) [19]. The contribution of his proposal lies in the calculation of a reduced state space compared to that evoked by [3]. Such proposal was improved by [18] and [17] to support the tasks with variable time execution.
The STPN [25] 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 [4] 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 crossing date of an event. However the modeling of a scheduling policy of a hard real-time system requires that all the 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 more difficult.
A new extension PTPN (Priority Time Petri Nets) was proposed in [13], 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 [13]. 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 [21].
The previously presented research studies are concerned with the static priority for scheduling analysis. In multiprocessor systems, it is necessary to specify the scheduling analysis through dynamic priority [12]. In this context, we make out two PNs extensions dealing with dynamic priority from which only one is dedicated for scheduling analysis.
Bause in [7] presents an analysis of Petri nets by introducing dynamic priorities to reduce the reachability graph. To deal with it, the authors have defined classes of priorities that describe the different variances of a transition enabled by different markings in different model states. Besides, the authors indicate that transitions in structural conflict belong to the same priority class. However, such a condition does not support a scheduling strategy which aims to lift conflicts.
In [14], the authors proposed a new TPNs extension *dPTPN* (dynamic Priority Time Priority Time Petri Nets) with dynamic priority via a new component. Indeed, the priority is relative to model state. The scheduling analysis is shown through the scheduling policy LLF (Least Laxity First) [9] and a set of independent periodic tasks running on a multiprocessor architecture. However, the LLF is not frequently used in practice because the cost of preemption is so high compared to the Earliest deadline First (EDF) [19].

B. Contribution and outline of the paper

The dPTPN is considered as the first PNs extension with dynamic priority dedicated for scheduling analysis. However, it was proved with the LLF strategy but this policy is not frequently used in practice because the cost of preemption is so high compared to the Earliest deadline First not frequently used in practice because the cost of preemption is so high compared to the Earliest deadline First. Besides, [14] is limited to deal with a set of independent tasks and no experiment exists to proving the results.
Compared to [14], in this paper we are interested to update the *dPTPN* to deal with the optimal dynamic scheduling policy (EDF) and we show the scheduling analysis via an application characterized by dependent tasks running on multiprocessor architecture.
The present paper is organized as follows. Firstly, an overview of the $dPTPN$ is detailed in section 2. Next, section 3 shows the experimentation (robot footballer) and how the *dPTPN* formalism is used for specifying the scheduling analysis associated to it. Finally, the proposed approach is briefly outlined and future perspectives are given.

II. Overview of dPTPN:

The major challenge of *dPTPN* is the integration of a dynamic-Priority driven scheduling policy in Petri Nets, which is a strategy based on the dynamic calculation of priorities. Indeed, the value of a priority changes at runtime to solve the conflict problem of enabled events.
The *dPTPN* distinguishes between temporal and concurrent events that are sources of conflict. Indeed, two types of transitions T (temporal transition Fig. 1) and T_{cp} (compound transition Fig. 2) are proposed.

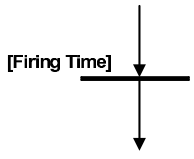

Figure 1. Temporal Transition

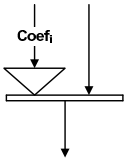

Figure 2. T_{cp} Transition

The temporal transition T (Fig. 1) is an ordinary PNs transition with a firing date presented with an integer value between braces. This presentation of Time is dedicated to deterministic Real Time Systems [13], [21], [14].
The second type of transitions, T_{cp} (Fig. 2), is a transition with a preprocessing that precedes the crossing to calculate its priority. In fact, when two T_{cp} 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*.
We start with the presentation of *dPTPN* syntax, then we explain the semantic of execution to support EDF strategy. To do so, we illustrate the PNs definition.
A Petri Net [24] can be defined as 4-tuplet :

$$PN = \langle P,\ T,\ B,\ F \rangle \tag{1}$$

, 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-tuplet :

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

(1) PN: is a Petri Net;
(2) $T_{cp} = \{T_{cp_1}, T_{cp_2}, \cdots, T_{cp_k}\}$: 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 $\Longleftrightarrow T_f(t) \neq 0$.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;
The semantic of firing in *dPTPN* is based on the partial order theory [2] [15] [6] building on a relation of equivalence between various sequences of possible crossings, 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.
The *dPTPN* semantic is presented with a *dPTPN* firing machine ($dPFM$). For each marking M, the *dpfm* initializes a set of transitions dFT_s composed by enabled temporal transitions FT_s and enabled compound transitions $FT_{s_{T_{cp}}}$. The initializations is called *Firiability* processing.

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

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

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

Next, valid transitions are selected from FT_s to VT_s by applying the *Validity* processing. All urgent transitions must be indicated in VT_s to be ready for firing.

$$VT_s = \{t \in FT_s / Hl(t) = T_f(t)\} \tag{5}$$

The $dFT_{s_{T_{cp}}}$ presents all concurrent transitions. To solve this conflict, the *dpfm* calculates the priority of each transition using the marking M and the $coef$ matrix. Then, the $dFT_{s_{T_{cp}}}$ is filtered to present only the transitions with the highest priority. This filtering is made with the *Step Selection* processing. In fact, this processing is able to select the T_{cp} transition having the highest priority according to its neighborhood6.

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

$$\Leftrightarrow \exists p \in P\ such\ that\ B_{T_{cp}}(p, T_{cp_1}) \neq 0 \wedge B_{T_{cp}}(p, T_{cp_2}) \neq 0 \tag{6}$$

III. Robot footballer experimentation

The experiment presents a football player robot application [23] in which the video tasks for object detection, wireless communications for message exchanging with other devices, motors controls, sensor acquisition, image processing and decision computation are included. The studied system is composed of four major parts:

- Acquiring and processing image. It is handled through tasks T2, T5, T7, T8 and T9;
- Communication HF: The information exchanges between the robot, the players and coaches are made by the following tasks: T1, T4,
- T6 and T12. Knowing that while T12 is used to send data, T1, T4 and T6 serve for reception;
- Data fusion by task T10 and path computation through T11;
- Control of location: it is done through the new trajectory coordinates calculated by the task T11 and through the current robot position. The location is computed through task T3. Thereafter, T13 controls the motors;

The dependencies between the 13 studied tasks are defined in Fig. 3 as follows:

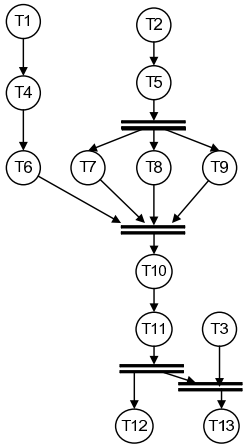

Figure 3. Task graph of Robot footballer application

As for the system architecture, it is composed of four processors. In addition, the robot architecture includes a set of memories: cache memory, DMA and RAM. It also covers a battery and a communication bus.

A. Model construction

The system Ω presents the scheduling formal specification of the robot footballer experiment. It is defined by the 4-tuplet:

$$\Omega = \langle Task,\ Proc,\ Alloc,\ Prec \rangle \tag{7}$$

with:

- $Task : \{T1, T2, \cdots, T13\}$,
  each $Task_i \in Task$ is determined by
  $$Task_i = \langle R_i,\ P_i,\ C_i \rangle \tag{8}$$
  - 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.
- $Proc : \{P1, P2, P3, P4\}$.
- $Alloc : \ Task \mapsto Proc$, 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.
- $Prec : \ Task \times Task \mapsto \{0, 1\}$, a function which initializes precedence relations between tasks.

To model the system Ω with the new *dPTPN* extension, we specify each component of Ω. First, we present the *dPTPN* model of Task, we specify with Task $T1$. Then, we focus on the specification of dependency between tasks. To do so, we distinguish two types of dependency models: shared resources model and precedence model.

1) Task Model:

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

- Creation and Activation:

As shown in Fig. 4, each state is modeled 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.
At first, $T1$ is in an uncreated state, it is presented by a marked place ($T1UnCreated$). The creation is modeled by a temporal transition "$T1Creation$" which takes a date of firing "0".

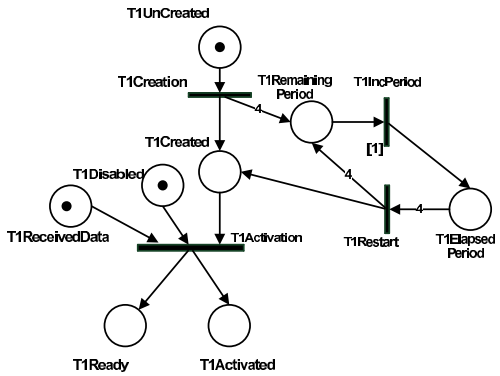

Figure 4. Creation and Activation

The crossing of "*T1Creation*" allows to register a mark in the place "*T1Created*" and "4" tokens in "*T1RemainingPeriod*" to indicate the start of the first execution period. For each tic of the global clock of the *dPTPN*, the transition "*T1IncPeriod*" is validated. Besides, its firing leads to the decrease of the remaining duration before the deadline and increase of the exhausted time of the period. When the 4 tokens are moved towards the place "*T1ElapsedPeriod*", the period is exhausted, a new period must be activated and a new instance of $T1$ must be created.
To model the activation of an instance of the task $T1$, we use a mutual exclusion described by two places "*T1Disabled*" and "*T1Activated*". Moreover, we use a place "*T1ReceivedData*" to present all necessary data for $T1$ to be activated. $T1$. The first instance of $T1$ is deactivated, where from the presence of a token in "$T1Disabled$". $T1$ has no precedent, so no data is necessary to activate the first instance and the "*T1ReceivedData*" is initialized with one mark. The current marking allows the firing of the transition "*T1Activation*" and the firing allows the supply of a mark in places "*T1Ready*" and "*T1Activated*". The current state proves that the current instance of the task is activated and lends itself to be run on the processor resource.

- Execution and Deadline:

The "execution" pattern presents the Task State after the allocation of the shared processor. It will be started as soon as the marking of the model indicates the presence of a mark in the place "*T1getProc*" (Fig.5), the task begins the

execution by firing the immediate transition "*T1execution*".

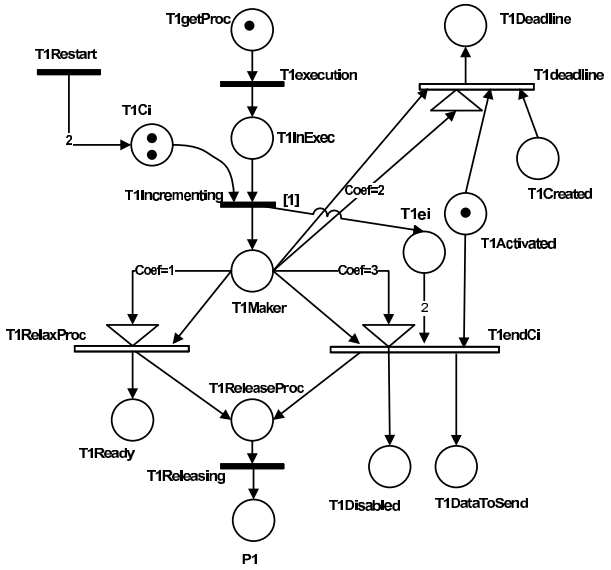

Figure 5. Execution and Deadline

EDF is a preemptive scheduling strategy. 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 "*T1incrementing*" with a date of firing equal to 1. The firing of this event allows not only the removal of a mark from the place "*T1Ci*" and another from the place "*T1InExec*" but also the registration of a mark in the place "*T1maker*" as well as the incrementation of the tokens of "*T1ei*" by another mark. Such incrementation indicates that the task ended a unit of execution during the activated period.

It should be noted that there is an emergence of three events "*T1endCi*", "*T1endDeadline*" and "*T1RelaxProc*".The scenario can be considered when the three events are enabled. The place "*T1maker*" 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 "$T1endCi$" transition is higher than "*T1endDeadline*". To do so, we attribute the coefficients "3" to the input arc of "*T1endCi*" and "2" to the input arc of "*T1endDeadline*".

The non-schedulable system is described when the "*T1endDeadline*" and "*T1Rel-axProc*" are enabled. It means that a new period is triggered when the last period is still activated. Since, the "*T1endDeadline*" must immediately be fired, we must attribute the higher priority to it and a smaller priority to "*T1RelaxProc*" ($coef = 1$). The crossing make a token in "*T1Deadline*" and retain the token from "*T1activated*". This marking presents a stop marking to the *dPTPN* Task model because it retains the token from the places of mutual exclusion. When this marking is reachable, the task is announced as non-schedulable.

2) Allocation Processor:

The resource processor is a shared resource between the various tasks of the same partition, and for a given moment, a single task occupies it. Each processor $P_i \in Proc$ is modeled with a simple place. If this place is marked then P_i is free otherwise it is allocated to one task of its partition. The allocation of the processor depends on the used scheduling strategy. In our study, we are interested in the strategy based on the Earliest Deadline First (EDF).

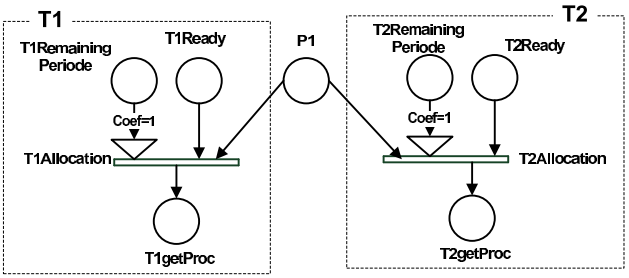

Figure 6. Allocation processor using EDF strategy

We consider that $T1$ and $T2$ are in the same partition and share the processor $P1$ (Alloc(Task1)=Alloc(Task2)=P1 Fig. 6). The current state presents a mark in "*T1Ready*", "*T2Ready*" and $P1$ to indicate that $T1$ and $T2$ call for the processor $P1$. So, the event of allocation is modeled by a transition "*T1allocation*" and "*T2allocation*" for $T1$ and $T2$, respectively. The processor will be attributed to the task having the transition "T_i*allocation*" with the highest priority (having the earliest deadline). Indeed, the allocation is modeled by a registration of a mark in the place "T_i*getProc*".

In Fig. 6, the earliest deadline value is presented with the marking of the place "*T1RemainingPeriod*" and "*T2RemainingPeriod*".

The main interest of "$coef$" matrix is to provide a solution for presenting the arithmetic operators. Indeed, in [14] it is used to model the equation L (to calculate the Laxity) with *dPTPN*. In the current study, we intercalate the coefficient "1" on the arc connecting the place "T1RemainingPeriod" and the "*T1Allocation*" (as well as for "*T2RemainingPeriod*" and the "*T2Allocation*").

Based on the semantic of *dPTPN*, the priority of "*T1Allocation*" is the multiplication of the "*T1RemainingPeriod*" marking and the corresponding coefficient of $coef$ matrix ($coef = 1$). In (Fig.6), "*T1Allocation*" is the highest priority because it has the earliest deadline.

After execution, the task $T1$ release the processor $P1$ on

firing the transition "*T1Releasing*". The crossing allows the liberation of the processor by putting a token in the place "$P1$" (Fig. 5).

3) Dependency between Tasks:

The Robot Footballer application requires data transmission between the tasks. Precedence constraint is dealt with to indicate any communication between tasks (Fig.3). For example, the task $T4$ is preceded by $T1$, so, each instance of $T4$ can be activated only after receiving data from its corresponding preceding $T1$ instance. Hence, the transmissions time of data between tasks is negligible thanks to the high-speed of the used DMA. As a consequence, the input task sends the information as soon as it finishes all or a part of its activity without the risk of waiting.

In Ω, the precedence relations between all tasks are described via $Prec$ function. (i.e. $Prec(T1, T4) = 1$ indicate that $T1$ precedes $T4$). The (Fig. 7) shows the *dPTPN* model for the communication between $T1$ and $T4$. The current marking indicates, on the one hand, that $T1$ has finished an instance of execution and is ready to send the necessary data to $T4$. On the other hand, $T4$ is disabled and still waiting for data from $T1$ to switch in activated mode.

It should be born in mind that each instance is preceded by another instance of the same task. So, we present a place (i.e. "*T1toT1*" according to task $T1$) to indicate that one instance has finished its execution on the processor and another can be activated in the next period.

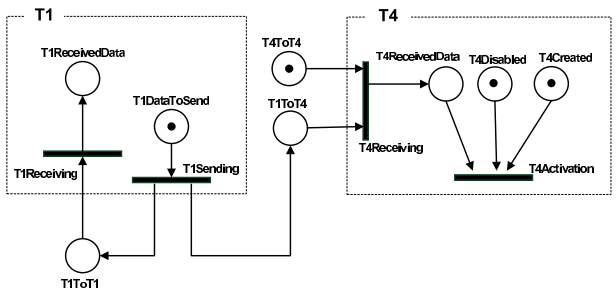

Figure 7. Communication between T1 and T4

"*T1Sending*" is enabled with the current marking. The crossing is immediate and it allows to put one mark in "*T1toT1*" and one in "*T1toT4*". The new marking indicate that the next instance of $T1$ can be activated in the next period and the necessary data for the activation of an instance of $T4$ is transmitted (Fig. 7).

"*T4Receiving*" is enabled and validated according to the new marking. Since its crossing, the task $T4$ has all necessary data to activate an instance and "*T4Activation*" is therefore enabled.

B. Tool and model execution

This section introduces dPTPNS (dynamic Priority Time Petri Nets for Scheduling analysis), the tool that we have implemented to concretize our proposed formalism and the implementation uses the Graphical Modeling Framework (GMF). We subsequently present the scheduling analysis result of the experimentation with *dPTPNS*.

1) The dPTPNS tool:

The implemented tool takes the form of a Petri Net editor and an executer model of the modeled net. Indeed, the developed editor of our *dPTPN* relies on the GMF founded on Eclipse Modeling Framework (EMF). Besides, the definition of the *dPTPN* Meta Model represents the starting point of the editor's generation process. As shown in Fig. 8, the regular Petri Nets Meta Model is extended with the attribute of two types of transitions: Temporal and Tcp. Thus, the production of an editor plug-in allows the interactive edition of the *dPTPN* (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) [10]. The validation doubles through the verification during and after constructing the model. It is obvious that

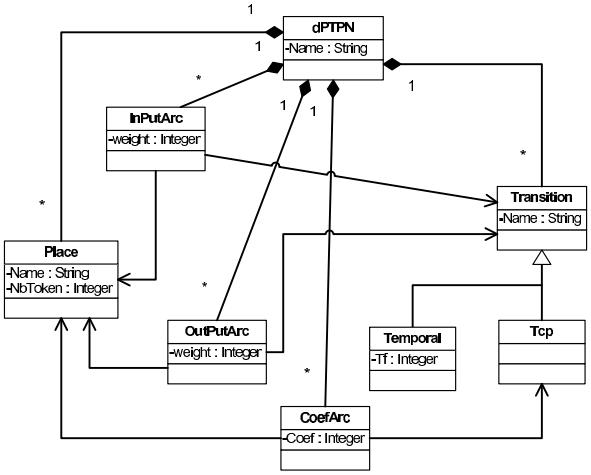

Figure 8. dPTPN Metamodel

the created model is built around a drawing composed of places, transitions and arcs. In fact, we need to easily extract the existing data from the editor. Fortunately, the created model can also be serialized to generate an XML (Extensible Markup Language) or XMI (XML Metadata Interchange) file. The generated file conforms to the dPTPN 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 *dPTPN* models. Despite the representation of *dPTPN* elements provided by the editor, the palette is equipped with *dPTPN* components in order to facilitate the illustration of complex tasks and computing resources. So, it is sufficient for the developer to select the structured *dPTPN* class from the palette with the communication means.

2) Model execution:

In this section we have shown how *dPTPNS* can be used to specify and analyze the robot footballer application. It should be born in mind that the *dPTPNS* is a tool responsible for scheduling the analysis of a given Tasks/Processors partition. However, to generate different partitions, a specific partitioning tool is required such as *RTDT* [28] and for each partition the *dPTPNS* is used for analysis.

Table.I presents the characteristics and the corresponding allocation of each task of the system Ω that is adequate to a given partition.

Table I
THE CHARACTERISTICS AND THE DISTRIBUTION OF EACH TASK

$Task_i$	R_i	P_i	C_i	$Alloc$	$Task_i$	R_i	P_i	C_i	$Alloc$
1	0	20	8	P1	8	0	45	8	P2
2	0	30	15	P1	9	0	40	6	P3
3	0	70	8	P4	10	40	40	10	P3
4	0	20	6	P1	11	40	20	12	P4
5	0	40	15	P2	12	70	20	12	P4
6	0	20	4	P1	13	70	30	10	P4
7	0	40	15	P2	-	-	-	-	-

Using the *dPTPNS* editor and considering the task model presented in the previous section, we have created task T_i of Ω and initialized the model with the initialed marking. In fact, each place "T_i*Increated*", "T_i*Disabled*" and "$T_i to T_i$" are initialized with one token. Each place "T_i*Ci*", "T_i*RemainingPeriod*" are initialized with the corresponding C_i and P_i marks, respectively.

The *dPTPNS* is accompanied with a simulator that implements the semantics of the *dPTPN* formalism. After creating the Ω model with *dPTPN* editor, the simulation is started.

At instant t= 0ms, T1, T2 and T3 are enabled and T1, T3 has the highest priority on P1 and P4, respectively. After the execution of T1, at t=8ms, T4 received all the necessary data to be activated. At this instant, the deadline of T4 is earlier than T2, so T4 takes the processor P1.

The *dPTPNS* simulator indicates at t=30 the activation of a new instance of T2 when the previous one does not achieve its execution on the P1 processor. As a consequence, the simulator shows the crossing of the "*T4deadline*" transition and puts a mark in the place "*T4Deadline*". As presented in a task model section, the marking of "*T4Deadline*" is a stop-Marking. Thus, the simulation is stopped and the *dPTPNS* indicates that T1, T2, T4 and T6 are non-schedulable on the processor P1. This description presents not only the scheduling analysis results, but also a useful feedback to the portioning tools to eliminate this task combination during the next generation.

Compared to the existing Time Petri Nets simulators such as ROMEO [8] and TINA [5], the impetus of our tool is the integration of the dynamic 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, in addition to the dynamic priority, we note the following distinctions:

- Contrary to Cheddar tools [27], Mast [11], Times [1], which cannot cover all the possible states of the system, *dPTPN* starts from an initial state to succeed in determining the error source if it occurs.
- Pertaining to other extensions presented in Section 2, *dPTPN* offers a strategy which accelerates the marking and avoids the combinatorial explosion in front of a large number of states. This strategy is based on partial order theory and simultaneous crossing of a set of enabled transitions [14].

IV. CONCLUSION

Dynamic Priority Time Petri Nets (*dPTPN*) represent a powerful formalism for the scheduling analysis of real-time systems running on multiprocessor architecture. The originality of the *dPTPN* semantics, compared to the existing research work, is the dynamic calculation of the priority of transitions in conflict.

In this paper, the main advantage compared to [14] is the use of the scheduling policy (EDF) to reduce the preemption cost. As a result, we have shown that *dPTPN* is able to support any dynamic Priority-driven scheduling policy. Besides, we have demonstrated that *dPTPN* can specify dependent tasks. Finally, the *dPTPNS* tools is presented to support the proposed formalism. It takes the form of a plug-in to allow an interactive edition of the *dPTPN* formalism. We have proved the proposed approach via the robot footballer experiment. The *dPTPNS* is able to detect the combination of tasks causing a temporal fault when the RTS is non-schedulable.

In future research, we are interested in the properties verification such as liveliness and safety to particularly present the behavior of an RTS. Furthermore, we intend to integrate the *dPTPN* formalism into a HW/SW partitioning approach based on a Model Driven Engineering (MDE) [26]. Indeed, we aim at showing how the *dPTPN* can be able to prove an RTS and how it can be useful to reduce the space solutions of the partitioning activity.

REFERENCES

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

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

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

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

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

[6] U. Buy and R.H. Sloan. 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, 1994. ACM.

[7] F.Bause. Analysis of petri nets with a dynamic priority method. In Pierre Azma and Gianfranco Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 215–234. Springer Berlin / Heidelberg, 1997.

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

[9] Joel Goossens, Pascal Richard, P. Richard, and Universit Libre De Bruxelles. Overview of real-time scheduling problems. In *Euro Workshop on Project Management and Scheduling*, 2004.

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

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

[12] J.Carpenter, S.Funk, P.Holman, A.Srinivasan, J.Anderson, and S.Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In *Handbook on Scheduling Algorithms, Methods, and Models*. Chapman Hall/CRC, Boca, 2004.

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

[14] W. Karamti, A. Mahfoudhi, Y. Hadj Kacem, and M. Abid. A formal method for scheduling analysis of a partitioned multiprocessor system: dynamic priority time petri nets. In *PECCS*, 2012, to appear.

[15] V. Kimmo. On combining the stubborn set method with the sleep set method. In Robert Valette, 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, 1994. Springer-Verlag Berlin Heidelberg 1994.

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

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

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

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

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

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

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

[23] H.Kitano M.Veloso, E.Pagello. Robocup-99: Robot soccer world cup iii. In *Velsoso (Eds.)*.

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

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

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

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

[28] H. Tmar, J. P. Diguet, A. Azzedine, M. Abid, and J. L. Philippe. Rtdt: A static qos manager, rt scheduling, hw/sw partitioning cad tool. *Microelectronics Journal*, 37(11):1208–1219, 2006.