Hierarchical Modeling with dynamic Priority Time Petri Nets for Multiprocessor Scheduling Analysis

Walid Karamti[1], **Adel Mahfoudhi**[1], **and Yessine Hadj Kacem**[1]
[1]CES Laboratory, ENIS Soukra km 3,5, University of Sfax,
B.P.:w 1173-3000 Sfax TUNISIA

Abstract— *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.*
The present paper presents a new modeling strategy with dPTPN *based on object modeling concept. Thus, a new component is proposed and the scheduling model is constituted with different instances of it. The scheduling is assured through the Earliest Deadline First with a set of dependent tasks. We prove the capacity of our approach to detect the non-schedulable sequences via an experiment.*

Keywords: Real-Time System; Scheduling analysis; EDF; dPTPN

1. Introduction

Multiprocessor architectures are becoming increasingly used in several systems such as the Real-Time system (RTS). It can explain the growth of the variety of research results. The main research area is the scheduling analysis of the real-time application running on a multiprocessor architecture. Hence, two main scheduling families exist. The first family is called the global scheduling in which all the tasks are charged on only one queue. In fact, although each task can migrate among the processor resources to achieve its execution, the cost of migration is so important and there are no optimal scheduling algorithm [16]. As for the second family, it is the partitioned scheduling in which each processor resource has its own queue. When a task is assigned to one processor, then it cannot migrate to another. In fact, this strategy presents a reduction of the multiprocessor scheduling to single-processor where the optimality is proved [19].
The partitioned scheduling is based on two procedures, the first of which is assigning tasks to processors and the second is analyzing the scheduling of each partition [20]. It is so important to detect the scheduling faults at an early stage in order to minimize the costs for its correction.
Therefore, to protect such systems from problems and failure, it is necessary to implement formal techniques intended to make reliable the development process of the real-time applications, from their design to checking. This allows designers to accurately validate systems, and check the required properties of their behavior.
The choice of the adequate formal method from the existing varieties depends on the characteristics of the considered system and the properties to check. 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.
Particularly, Petri Nets (*PNs*) presents an appropriate 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 *PNs* for the scheduling analysis of an *RTS* running on a Multiprocessor architecture.
The Multiprocessor scheduling analysis with *PNs* is a recent research area, which explains the scarcity of Petri Nets dealing with it. The dynamic priority presents a primordial factor in the Multiprocessor scheduling [11] but we distinguish a limitation of *PNs* extensions that support it is distinguished. It can be explained by the difficulty to introduce such characteristic in *PNs*. In what follows, we present the *PNs* extensions with fixed priority and next we detail the existing extension with dynamic priority.
The *STPN* [24] is a temporal *PNs* extension dedicated to analyze periodic tasks on a multiprocessor architecture. It is able to support a fixed priority scheduling policy such as *RM* (Rate Monotonic) [19] thanks to the use of the inhibitor arcs. The contribution of its proposal lies in the calculation of a reduced state space compared to that evoked by [3]. Such proposal has been improved by [18] and [17] to support the tasks with variable time execution.
Before the crossing of transitions, the *STPN* [24] adds constraints 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.
The *PrTPNs* (Priority Time Petri Nets) [4] also utilized the inhibitor arcs to present the notion of fixed priority. The authors propose 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* present the priority 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 [12], 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 [12]. 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.
In [13], the authors have proposed the first *PNs* extension *dPTPN* (dynamic Priority Time Priority Time Petri Nets) dealing 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) [8] 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]. In the same vein, the authors have proven the capacity of the *dPTPN* to deal with *EDF* as well as with the dependent tasks in [14]. However, the size and the complexity is increased even though the considered *RTS* is more complex. Hence, the execution of the model and the checking of its properties is more difficult.
The main contribution in this paper is the proposition of a new modeling strategy to master the complexity of the *dPTPN* Model. Building on Object modeling, we propose a new *dPTPN* component and identify how it can be instanced to specify the scheduling analysis model.
The present paper is organized as follows. Firstly, we start with presenting the experimentation (robot footballer) in section 2. Next, the definitions of the *dPTPN* and its semantics are detailed in section 3. Next, section 4 shows the Object modeling approach and the creation of a new component. In this section, the modeling of the experiment is shown with different instances of the new component. In section 5, we present the *dPTPN* Scheduling analysis tool (*dPTPNS*). Finally, the proposed approach is briefly outlined and future perspectives are given.

2. Robot footballer experimentation

The experiment presents a football player robot application [22] 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 are used 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. 1 as follows: As for the system architecture, it is com-

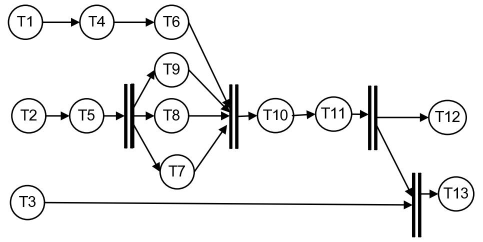

Fig. 1: Task graph of Robot footballer application

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

$$\Omega = \langle Task,\ Proc,\ Alloc,\ Prec\rangle \tag{1}$$

with:

- $Task : \{T1, T2, \cdots, T13\}$,
  each $Task_i \in Task$ is determined by

  $$Task_i = \langle R_i,\ P_i,\ C_i\rangle \tag{2}$$

  - 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.

3. dynamic Priority Time Petri Nets - Preliminaries

The integration of the dynamic calculation of priorities in Petri Nets presents the ultimate objective of the *dPTPN* [13]. In fact, to solve the conflict problem of enabled transitions, the priority changes at runtime according to the Nets state. The *dPTPN* distinguishes between temporal and concurrent events that are sources of conflict. Indeed, two types of transitions T (temporal transition Fig. 2) and T_{cp} (compound transition Fig. 3) are proposed.

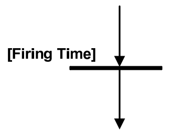

Fig. 2: T-Transition [13]

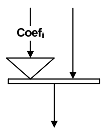

Fig. 3: T_{cp}-Transition [13]

With respect to temporal transition T (Fig. 2) 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 [12], [21], [13].
As for the second type of transitions, T_{cp} (Fig. 3), 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 the *dPTPN* formal definition, then we explain the semantic of execution. Next, we have shown the internal behavior of real-time task with the *dPTPN*.

3.1 Formal Definition

A Petri Net [23] can be defined as 4-tuplet :

$$PN = \langle P,\ T,\ B,\ F \rangle \tag{3}$$

, 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 :

$$dPTPN = \left\langle PN, T_{cp}, T_f, B_{T_{cp}}, F_{T_{cp}}, coef, M_0 \right\rangle \tag{4}$$

(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 semantics 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* semantics is presented with a *dPTPN* firing machine (*dPFM*). For each marking M, the *dpfm* initializes a set of transitions dFT_s composed of enabled temporal transitions FT_s and enabled compound transitions $FT_{s_{T_{cp}}}$. The initializations is called *Firiability* processing.

$$dFT_s = FT_s \cup FT_{s_{T_{cp}}}. \tag{5}$$

$$let\ t \in T, t \in dFT_s \Leftrightarrow t \in FT_s \vee t \in FT_{s_{T_{cp}}} \tag{6}$$

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

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.

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

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 neighborhood (eq. 8).

$$\forall T_{cp_1}, T_{cp_2} \in T_{cp}, T_{cp_1}\ is\ a\ neighbor\ of\ T_{cp_2}$$
$$\Leftrightarrow \exists p \in P\ such\ that\ B_{T_{cp}}\,(p, T_{cp_1}) \neq 0 \wedge B_{T_{cp}}\,(p, T_{cp_2}) \neq 0 \tag{8}$$

In this step, the proposed *dPTPN* is able to support a selection policy. In [13], the authors have proved that *dPTPN* can attribute the priorities to transitions sharing the place processor according to the *LLF* policy. In [14] the authors has further proven their extension with the Earliest Deadline First policy (*EDF*).
Finally, the *dpfm* fires all transitions in the updated sets. The firing is described by the following equation:

$$\forall FT \in \left\{VT_s, FT_{s_{T_{cp}}}\right\}, Firing\,(FT) \Longrightarrow$$
$$\begin{cases} FT = VT_s \\ \Leftrightarrow M' = M + \sum_{t \in FT}\left(F\,(.,t) - B\,(.,t)\right) \\ \\ FT = FT_{s_{T_{cp}}} \\ \Leftrightarrow M' = M + \sum_{t \in FT}\left(F_{T_{cp}}\,(.,t) - B_{T_{cp}}\,(.,t)\right) \end{cases} \tag{9}$$

More details about the *dpfm* and the firing process can be found in [13].

3.2 Model construction with dPTPN

In the [13] and [14], the authors have suggested a specification with *dPTPN* of the important component of the *RTS* : the Real-Time Task. In fact, the internal behavior of tasks is presented through two major patterns, the first of which describes the creation, the activation and the deadline model of the tasks. This pattern is critical at the scheduling analysis of the *RTS*. It is modeled for describing a *stop-Marking* when it was a temporal fault in the system.
As for the second pattern, it is the modeling of the allocation and execution of the task on the processor. The processor is a shared resource between the tasks of the same partition. The allocation event is modeled through a T_{cp} transition and the transition having the highest priority, under a defined policy (*EDF* in [13], *LLF* [14]) allocates the processor and begins its execution. The execution modeling is dedicated to discrete time and for each tic of clock the task is asked for liberation of the processor if a new coming task has the highest priority. Figure (Fig. 4) presents

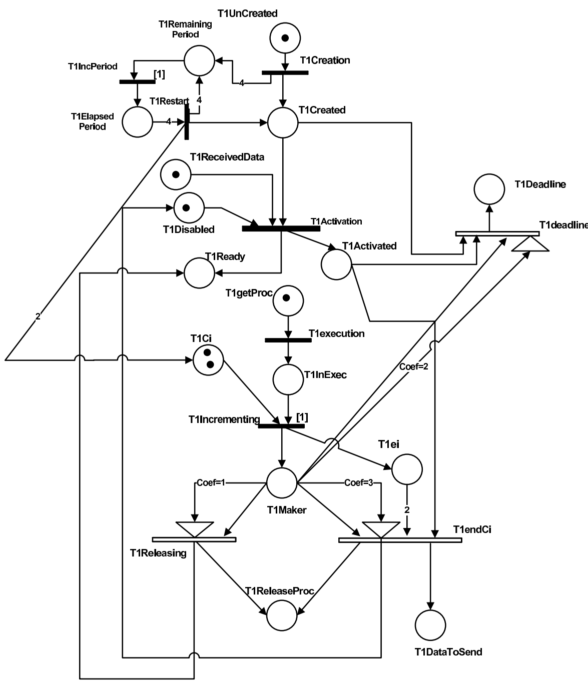

Fig. 4: RTS Task internal behavioral with dPTPN

the completed *dPTPN* model of the internal behavior of the task $T_1(0,4,2) \in Task$. It can be noted that this model is composed of 16 places, 7 *T-Transitions* and 3 T_{cp}-*Transitions* and for modeling the system Ω those numbers are increased. Thus, the complexity of the modeling and its interpretation become more and more difficult.
We distinguish that for each task of Ω, the model *dPTPN* is similar. In fact, just the initialization of the model with the firing times and the weights of arcs change. The modification correspond to the chosen task for modeling. We can consider the *dPTPN* model as an Object and each task $T_i \in Task$ is an instance of this Object.
In the coming section, we propose the definition of the Object Task. Next, we define the new model of Ω using the instances of the proposed Object.

4. Object modeling with dPTPN

Using Petri Nets to specify the behavioral specification of objects is a major tendency to integrate between objects and *PNs*. Indeed, the networks are used to describe the internal behavior of objects. Besides, the internal state of objects is indicated by the marks in the network places. Moreover, the execution of the methods of an object is described with the 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 *PNs* with a class of objects rather than with an individual object.
Based on this approach, we now present the definition of the new object "*Task*" and we specify the communication between the different instances of this object in order to model and analyze the schedulability of the system Ω.

4.1 Task Component

The *PNs* in objects depends on the encapsulation of the various behaviors of the object in a centenary called *PNs* component. We propose a *dPTPN* constituent called "TaskC" to encapsulate the behavior of a Real-Time task.
"TaskC" is characterized by two interfaces which assure the communication with its environment: input and output. In fact, each interface is a finite set of places. The graphical definition of *TaskC* is presented in Fig. 5 and defined with the triplet:

$$TaskC = \langle dPTPN, II, OI \rangle \tag{10}$$

with:

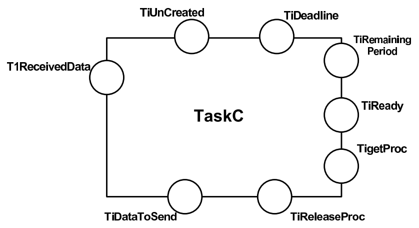

Fig. 5: The Task Object with dPTPN

(1) dPTPN: is the *dPTPN* model presented in Fig. 4;
(2) $II = \{P_{Uncreated}, P_{ReceivedData}, P_{getProc}\}$: is the places that composed the Input Interface;
(3) $OI = \{P_{Ready}, P_{Deadline}, P_{RemainingPeriod}, P_{SendData}, P_{Releasing}\}$: is the places that composed the Output Interface;

Let "$T_1(0,4,2) \in Task$" from Ω. Its corresponding "$TaskC_1$" component instance of "$TaskC$" is created as follows:

- The firing time of the creation event is initialized with "0": $T_{f_{TaskC_1}}(T_{Creation}) = 0$;
- The period is initialized on putting the weight "4" on the coming arcs of the place "$P_{RemainingPeriod}$" and on the outgoing arcs of the place "$P_{ElapsedPeriod}$":
  $F_{TaskC_1}(P_{RemainingPeriod},\ T_{Creation}) = 4$;
  $F_{TaskC_1}(P_{RemainingPeriod},\ T_{Restart}) = 4$;
  $B_{TaskC_1}(P_{ElapsedPeriod},\ T_{Restart}) = 4$;
- The execution time is initialized with adding the weight "2" on the input arcs of the place "P_{Ci}" from the transition $T_{Restart}$ and on the outgoing arc from the place "P_{ei}" to T_{endCi}:
  $B_{TaskC_1}(P_{ei},\ T_{endCi}) = 2$;
  $F_{TaskC_1}(P_{Ci},\ T_{Restart}) = 2$;

4.2 Modeling of the shared processors between Tasks

The processor is the resource responsible of the execution of tasks. In our study, we focus on the partitioned multiprocessor system. In fact, each task is assigned to one processor and the scheduling analysis of the system corresponds to analyzing each processor.
The processor is modeled, with $dPTPN$, by a place and its state is described by the present marking. It is free if a mark exists and occupied otherwise. 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). We consider two tasks ($T1$ and $T2$) are in the same partition and share the processor $P1$ (Alloc(T1)=Alloc(T2)=P1). Fig. 6 presents the *dPTPN* model corresponding to the shared processor $P1$ between the instances $TaskC_1$ and $TaskC_2$ of $TaskC$.
The current state presents a mark in "*P1Ready*", "*P2Ready*" and $Proc1$ to indicate that $T1$ and $T2$ call for the processor $P1$. So, the event of allocation is modeled by a transition

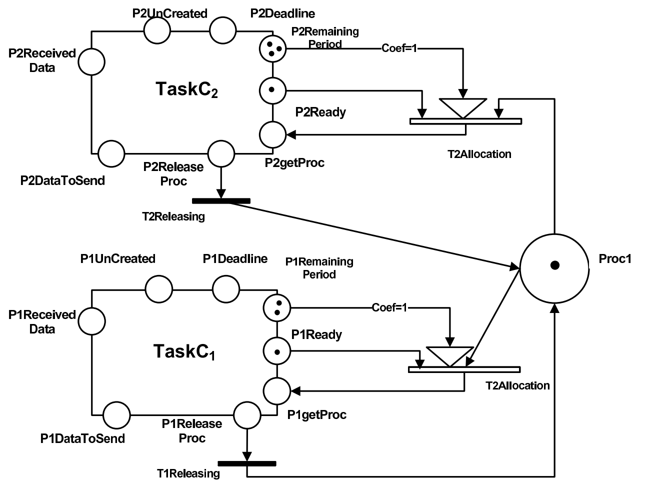

Fig. 6: Allocation processor using the EDF policy

"*T1Allocation*" and "*T2Allocation*" for $T1$ and $T2$, respectively. The processor will be attributed to the task component 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 theinput place "P_i*getProc*" of the corresponding task component.
In Fig. 6, the earliest deadline value is presented with the marking of the place "*P1RemainingPeriod*" and "*P2RemainingPeriod*".
The main interest of "$coef$" matrix is to provide a solution for presenting the arithmetic operators. Indeed, in [13] 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*" associated with $TaskC_1$ (as well as for *T2RemainingPeriod*" and the "*T2Allocation*" associated with $TaskC_2$).
Based on the semantics 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 $T1$ releases the processor $P1$ on firing the transition "*T1Releasing*" associated to the $TaskC_1$ component. The crossing allows the liberation of the processor by putting a token in the place "Proc1" (Fig.6).

4.3 Modeling of the communication between the instances of TaskC

The considered application (Robot Footballer) requires the transmissions of data between the tasks. Indeed, some tasks are preceded by some others as indicated in Fig. 1. Thus, the preceded task can be activated only after receiving data from its corresponding preceding task. 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. Formally, the precedence relations between all tasks are described in Ω via the $Prec$ function.

Fig. 7 shows the *dPTPN* model for the communication

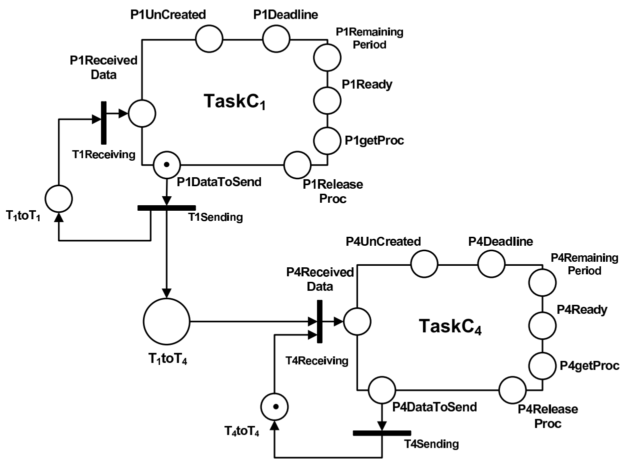

Fig. 7: Communication between T1 and T4

between $T1$ and $T4$. The current marking presents a mark in the output place "P1DataToSent" of the "$TaskC_1$". It indicates that the task $T1$ has finished an instance of execution during its period and is ready to send the necessary data for the activation of $T4$.

The immediate transition "$T1Sending$" is enabled and its firing allows the putting of one mark in the place "T_1toT_1" and one in "T_1toT_4". The main object for using the place "T_itoT_i" is to indicate the precedence between the different instances of execution of the task Ti.

The new marking enables and validates the transition "*T4Receiving*". Since its crossing, the task $T4$ has all necessary data to activate a new instance.

5. Tool and model execution

The *dPTPN* is accompanied with a scheduling analysis tool called *dPTPNS* [14] (dynamic Priority Time Petri Nets for Scheduling analysis). Indeed, it presents a Petri Nets editor and executer model.

The editor is implemented under the Graphical Modeling Framework (*GMF*) founded on Eclipse Modeling Framework (*EMF*). Hence, the *dPTPN* Meta Model represents the starting point of the editor's generation process. The ordinary Petri Nets Meta Model is extended with the addition of two Meta class: *Temporal* and *Tcp*. The created models are checked through a set of constraints expressed with the Object Constraint Language (OCL) [9]. The validation doubles through the verification during and after constructing the model.

It is obvious that the created model is built around a drawing

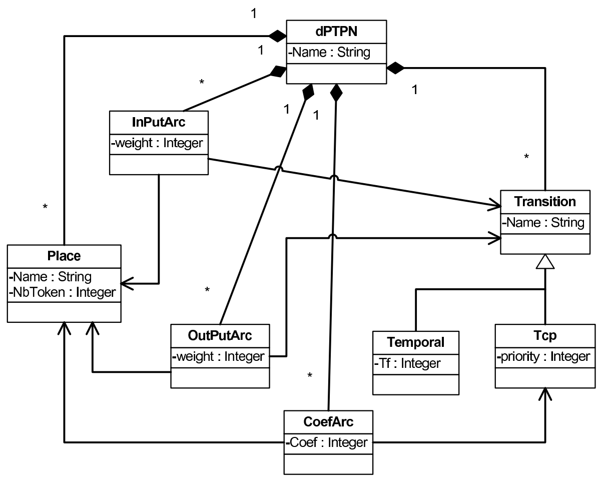

Fig. 8: dPTPN Metamodel

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.

Compared to the existing Time Petri Nets simulators such as ROMEO [7] 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 *PNs* 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 [26], Mast [10], 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 the other extensions presented in Section 2, *dPTPN* offers a strategy that 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 [13].

5.1 Model execution

To show how *dPTPNS* can be used to specify and analyze the robot footballer application, we consider the following table (Tab. 1) to present the specification of the system Ω.
The generation of the different partitions is made through a specific partitioning tool such as *RTDT* [27] and for each partition the *dPTPNS* is used for analysis.

Table 1: The specification of each partition

Partitions	Tasks			
	Name	R_i	P_i	C_i
P1	T1	0	20	8
	T2	0	30	15
	T4	0	20	6
	T6	0	20	4
P2	T5	0	40	15
	T7	0	40	15
	T8	0	45	8
P3	T9	0	40	6
	T10	40	40	10
P4	T3	0	70	8
	T11	40	20	12
	T12	70	20	12
	T13	70	30	10

For modeling the instances of $TaskC$, we just indicate the input and output places for each instance in the editor *dPTPNS*. To model the system Ω, we create 13 instances of *TaskC*.
The initial marking corresponds to initialize "P_i*Increated*", T_i*to*T_i" with one mark and "T_i*Ci*", "T_i*RemainingPeriod*" with the corresponding C_i and P_i marks from Tab. 1, 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 $T1$, $T3$ have the highest priority on $P1$ and $P4$, respectively. After the execution of $T1$, at t=8ms, $T4$ receives all the necessary data to be activated. At this moment, 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 output place "*P4Deadline*" of the $TaskC_4$. As presented in a the model construction with *dPTPN* section, the marking of "*P4Deadline*" 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.

6. Conclusion

The development of dynamic Priority Time Petri Nets (*dPTPN*) [13] models for the scheduling analysis of a multi-processor system has given very important results [13], [14]. In fact, it presents, on the one hand, a detailed specification of Real-Time System behavior. On the other hand, it is able to indicate the exact description of the non-schedulable sequence and request it as a feedback to the partitioning tool to obtain new partitions.
However, as the increasing complexity of the *RTS* gives birth to a very complex *dPTPN* model, in this paper, we present a new modeling technique. Based on the object modeling, we present a new component *TaskC*. Using different instances of it we obtain the new scheduling model. Hence, the scheduling policy considered in this paper is the Earliest Deadline First (*EDF*) [19] dealing with dependent tasks.
In future work, 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*) [25]. In fact, 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] G. Gardey, D. Lime, M. Magnin, and O. H. Roux. Romeo: A tool for analyzing time petri nets. In *CAV*, pages 418–423, 2005.

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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*, pages 317–326, 2012.

[14] W. Karamti, A. Mahfoudhi, and Y. Hadj Kacem. Using dynamic priority time petri nets for scheduling analysis via earliest deadline first policy. In *ISPA*, page to appear, 2012.

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

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

[24] 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.

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

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

[27] 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.