States Graph Generation
from dynamic Priority Time Petri Nets

Walid Karamti, Adel Mahfoudhi

CES Laboratory, ENIS Soukra km 3,5, University of Sfax, B.P.: 1173-3000 Sfax, Tunisia
e-mail: walid.karamti@ceslab.org
e-mail: adel.mahfoudhi@fss.rnu.tn

(Communicated by Noômen Jarboui)

Abstract
dynamic Priority Time Petri Nets (dPTPN) is a mathematical formalism dedicated to modeling Real-Time System (RTS) and checking its schedulability. The present paper proposes a states graph generation from a reduced dPTPN model in order to deal with the scheduling analysis. Based on hierarchical modeling, the present model presents only the interaction between all RTS components and excluding their internal behavior. According to this reduction, a new definition of state is given. Hence, all reachable states and edges connecting between them are generated to show a prediction of the RTS scheduling. Thus, the resulting graph gives birth to an open research area in the purpose of checking its properties and deducing the schedulability.

Keywords: RTS, dPTPN, States Graph, Scheduling analysis.

1 Introduction

The system analysis at an early stage in the development cycle is always an open problem for researchers in computer science. In fact, Real-Time Systems (RTS) have been omnipresent in several domains over the past 30 years and their analysis has grown over the last decade. Recently, a new generation of architectures (i.e. Multiprocessor and Multicore) has emerged. Thus, new methods and techniques are required for modeling and analysis.
The scheduling analysis of RTS running on multiprocessor architecture presents an interesting research field. In this vein, formal methods, based on mathematical principles and abstractions, are the cornerstone of analysis techniques. The light is mainlt shed on model checking approaches, which attempt to provide a push-button approach to verify and integrate well into standard development processes. However, since using such technique can derive a states-explosion problem, it is primordial to master the states generation before checking the properties.

First of all, the system must be modeled in a formal description language such as Time Petri Nets [14], timed automata. In order to deal with the states-explosion problem, and based on Time Petri Nets model, Berthomieu in [2] proposed a graph of class states. In fact, starting from graph composed with infinite states, he suggests a technique for grouping states in a finite number of classes. This technique is used in the PrTPNs (Priority Time Petri Nets) [3] so as to analyze the schedulability. An improvement of this approach was proposed in [16], in which the authors propose a new extension of Time Petri Nets STPN (Scheduling Timed Petri Nets) and a reduced states graph compared to [2]. Both of [2] and [16], produce a reduced states graph that is not so expressive to check the schedulability on immediately. So, the authors use timed automata as observers to check properties of the Petri Model and deduce the schedulability.

PrTPNs and STPN are concerned with the static priority for scheduling analysis. In multiprocessor systems, it is necessary to specify the scheduling analysis through dynamic priority [5]. So, it is not efficient to use them in scheduling analysis of multiprocessor system.

In [10], the authors have proposed the first Time Petri Nets extension dPTPN (dynamic Priority Time Priority Time Petri Nets) dealing with dynamic priority via the introduction of a new component. Indeed, the priority is relative to the model state. The scheduling analysis is shown through the support of the scheduling policy LLF (Least Laxity First) [7] 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) [12]. 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 [9]. In this stage, the authors have proposed a Petri model for the scheduling analysis. Hence, a detailed model for the periodic dependent tasks is presented. However, the size and the complexity of the entire RTS system model has increased even though the considered RTS is more complex. Hence, the determination of all reachable states corresponding to the model and the checking of its properties is more difficult.

In [8], the authors present a new modeling strategy to master the complexity
of the dPTPN model building on Object modeling, as well as a new dPTPN model component and identified how it can be instanced to specify the scheduling analysis model. Nevertheless, the exploitation of this abstraction of the model in the construction of model states and the properties checking is not detailed.

Contrary to states graph reduction techniques, the current paper presents a technique of states graph generation based on a reduced Petri Nets Model. In fact, to master the states-explosion problem, we start with a reduced model and we produce a finite and oriented states graph where the schedulability can be checked on it immediately (without observers automata). We apply the proposed approach on a partitioned real-time multiprocessor system characterized with dependent periodic tasks.

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 presents the considered RTS and how it can be modeled with the hierarchical modeling strategy in order to provide a reduced dPTPN RTS model. As for Section 4, it presents the generation of the states graph. Starting with the developed model, a set of states connected with edges are generated to describe a prediction of the RTS scheduling. Next, an open problem is detailed in Section 5. Finally, the proposed approach is briefly outlined.

2 dynamic Priority Time Petri Nets: dPTPN

The dynamic Priority time Petri Nets is a mathematical formalism used for analyzing, specially, the Real-Time Systems behaviors. whose definition and firing semantics of events are presented in this section.

2.1 Formal definition

A Petri Nets [15] can be defined a 4-tuplet :

\[ PN = (P, T, B, F) \]  

where:

1. \( P = \{p_1, p_2, \ldots, p_n\} \) is a finite set of \( n \) places;
2. \( T = \{t_1, t_2, \ldots, t_m\} \) is a finite set of \( m \) transitions;
3. \( B : (P \times T) \rightarrow \mathbb{N} \) is the backward incidence function;
4. \( F : (P \times T) \rightarrow \mathbb{N} \) is the forward incidence function.

Each system state is represented by a marking \( M \) of the net and defined by :

\[ M : P \rightarrow \mathbb{N} \]
In the PN standard, the events (transitions) have the same grade of emergency. Thus, when transitions conflict, there are no favorable transition to cross before the other. Besides, the time is not specified in PN.

A new extension, dynamic Priority time Petri Nets (dPTPN), is proposed to meet the time specification and solve the transitions conflict. In fact, two transition types are proposed. First, the $T$ transition is characterized by a date of firing. Second, the $T_{cp}$ is a transition with a preprocessing that precedes the crossing to calculate its priority. Indeed, if two $T_{cp}$ transitions are enabled and share at least one 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 $M$.

The dPTPN is defined by the 7-tuplet:

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

1. $PN$: is a Petri Nets;
2. $T_{cp} = \{T_{cp1}, T_{cp2}, \ldots, T_{cpk}\}$: is a finite set of $k$ compound transition;
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 timer ($Lt(t)$), with $Lt : T \mapsto \mathbb{Q}^+$.
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. $\text{coef} : (P \times T_{cp}) \mapsto \mathbb{Z}$ is the coefficient function associated with compound transition;
7. $M_0$ : is the initial marking;

### 2.2 The dPTPN Firing Machine

To deal with the problem of the state space explosion, it is worthwhile to mention the contribution of the methods considered as partial order [1, 11, 4] 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 transitions conflict. The strategy is based 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. The entry of the machine dPFM is a marking $M$ of the dPTPN network. For each entry, the machine builds the $dFT_s$ set, consisting of two sub-sets $FT_s$ and $FT^{TCP}_s$, 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 $FT^{TCP}_s$ is empty) and the dPFM machine stops.

The dPFM distinguishes between the temporal and immediate events. To do so, the machine supplies a chain of processing. Indeed, firstly, it determines the valid temporal transitions set $VT_s$ from the $FT_s$ set, 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 $FT^{TCP}_s$, 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 $FT^{TCP}_s$. 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 $FT^{TCP}_s$.

\[
dFT_s = FT_s \cup FT^{TCP}_s.
\]

Let $t \in T, t \in dFT_s \iff t \in FT_s \lor t \in FT^{TCP}_s$

with

\[
\begin{align*}
FT_s &= \{ t \in T/B (.,t) \leq M \} \\
FT^{TCP}_s &= \{ t \in T/B_{TCP} (.,t) \leq M \}
\end{align*}
\]

For each subset $FT_s$ and $FT^{TCP}_s$ is associated with a function indicator $\chi_{FT_s}$ and $\chi_{FT^{TCP}_s}$.

\[
\begin{align*}
\chi_{FT_s} & : T \rightarrow \{0,1\} \\
x & \mapsto \begin{cases} 
1 & \text{if } x \in FT_s \\
0 & \text{otherwise}
\end{cases} \\
\chi_{FT^{TCP}_s} & : T \rightarrow \{0,1\} \\
x & \mapsto \begin{cases} 
1 & \text{if } x \in FT^{TCP}_s \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]
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 $FT_s$ ($VT_s \subseteq FT_s$).

$$VT_s = \{ t \in FT_s / Lt (t) = T_f (t) \}$$  \hspace{1cm} (7)

$\chi_{VT_s}$ is the function indicator of $VT_s$ with:

$$\chi_{VT_s} : FT_s \rightarrow \{ 0,1 \}$$  \hspace{1cm} (8)

The main advantage of the 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 to its neighborhood. First of all, we present the neighborhood of transition $T_{cp_i}$:

$$\forall T_{cp_1}, T_{cp_2} \in T_{cp}, T_{cp_1} is a neighbor of T_{cp_2} \iff \exists p \in P such that B_{T_{cp}} (p, T_{cp_1}) \neq 0 \land B_{T_{cp}} (p, T_{cp_2}) \neq 0$$  \hspace{1cm} (9)

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

$$N_e : T_{cp} \times P \rightarrow \{ 0,1 \}$$

$$(t_{cp},p) \mapsto \begin{cases} 1 \text{ if } B_{T_{cp}} (p, t_{cp}) > 0 \\ 0 \text{ otherwise} \end{cases}$$  \hspace{1cm} (10)

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 $T_{cp}$ calculates its priority using the scalar product of $coef$ matrix with the marking vector $M$.

$$Prio : FT_{T_{cp}} \rightarrow \mathbb{Z}$$

$$t_{cp} \mapsto \langle coef (., t_{cp}) | M \rangle$$  \hspace{1cm} (11)

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

$$Prod : FT_{T_{cp}} \times P \rightarrow \mathbb{Z}$$

$$(t_{cp},p) \mapsto Prio (t_{cp}) N_e (t_{cp},p)$$  \hspace{1cm} (12)
the third step, the transition having the highest priority per palce is selected from the vector \( \text{Prod}(.,p_i) \).

\[
\text{Min} : \quad P \rightarrow FT_{stcp} \\
p \rightarrow t_{cp}
\]

(13)

with \( \{\forall t_{cp}, \neq t_{cp},\text{Prod}(t_{cp},p) < \text{Prod}(t_{cp},p)\} \)

Finally, the \( FT_{stcp} \) must be updated to present only the selected transitions:

\[
\forall t_{cp} \in T_{cp}, \forall p \in P, \quad \text{Prod}(t_{cp},p) \neq 0 \land t_{cp} \neq \text{Min}(p) \implies \chi_{FT_{stcp}}(t_{cp}) = 0
\]

(14)

2.2.4 Firing

The firing of a given \( FT \) vector is defined. In \( dPTPN \), \( FT \) can be a \( VT_s \) vector or an \( FT_{stcp} \) vector:

\[
\forall FT \in \{VT_s, FT_{stcp}\}, \quad \text{Firing}(FT) \implies \begin{cases} 
FT = VT_s \\
\iff M' = M + \sum_{t \in FT} (F(.,t) - B(.,t)) \\
FT = FT_{stcp} \\
\iff M' = M + \sum_{t \in FT} (F_{T_{cp}}(.,t) - B_{T_{cp}}(.,t))
\end{cases}
\]

(15)

2.2.5 Timers management

According to \( dPTPN \), the timers management abides to some rules, which are:

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

Rule2(reset): After firing \( VT_s \), the timers of valid transitions will reset.

Rule3(increment timer): The timer of a transition is incrementing if the transition is enabled, no valid transition exists and the \( FT_{stcp} \) is empty.

\[
\text{SetIncrementTimer()} \text{ is activated} \iff \begin{cases} 
FT_s \neq \emptyset \land FT_{stcp} = \emptyset \land VT_s = \emptyset
\end{cases}
\]

(16)

3 Real-Time Multiprocessor System

The \( dPTPN \) is dedicated for analyzing the schedulability of Real-Time System (RTS) running on multiprocessor architecture [10, 9]. Such systems are characterized with dynamic priority-driven scheduling where the \( dPTPN \) has proved its capacity to deal with it.

The present section defines the RTS and how it can be modeled with \( dPTPN \).
3.1 RTS definition

The considered RTS in the current manuscript is a periodic system. Besides, it is partitioned over the processors via a partitioning tool. Our study highlights the analysis of all the generated partitions based on the dPTPN. Ω is the specification of the RTS and it is defined by the 4-uplet:

\[ \Omega = \langle \text{Task}, \text{Proc}, \text{Alloc}, \text{Prec} \rangle \]  

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

\[ Task_i = \langle R_i, P_i, C_i \rangle \]  

- \( R_i \): 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.


3. Alloc : Task \( \rightarrow \) 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.

\[ \forall t_1 \in \text{Task}, \forall P_1, P_2 \in \text{Proc}, \]  

\[ \text{Alloc}(t_1) = P_1 \land \text{Alloc}(t_1) = P_2 \Rightarrow P_1 = P_2 \]  

4. Prec : Task \( \times \) Task \( \rightarrow \) \{0, 1\}, a function which initializes precedence relations between tasks.

3.2 RTS Modeling

The Task presents the first main component of Ω, that is why we are interested on presenting its dPTPN specification and later we define a prediction of its behavior according to its neighborhood.

In previous research work [10, 9], we modeled the internal behavior of a real-time task with dPTPN. In scheduling analysis, although the external behavior of task is an important key, it depends on analyzing the internal one. In order to synchronize between those two behaviors, our proposal builds on hierarchical modeling to present only the external events. In fact, the states issues from internal events will be masked. Thus, a new dPTPN component for modeling the real-time task is defined [8].

TaskC is characterized by two Interfaces that assure the communication with
its environment: Input and Output. Actually, each interface is a finite set of places. The graphical definition of TaskC is defined with the triplet:

\[ \text{TaskC} = \langle \text{dPTPN}, II, OI \rangle \]  

with:

1. \text{dPTPN}: is the task dPTPN model presented in ([8], Fig. 4);
2. \( II = \{ P_{\text{Uncreated}}, P_{\text{ReceivedData}}, P_{\text{getProc}} \} \): is the place that composes the Input Interface;
3. \( OI = \{ P_{\text{Ready}}, P_{\text{RemainingPeriod}}, P_{\text{SendData}}, P_{\text{Releasing}}, P_{\text{Deadline}} \} \): is the place that composes the Output Interface;

The dependency between tasks is specified in \( \Omega \) with the function \( \text{Prec} \). However, in dPTPN modeling, we distinguish between two dependency relations. Concerning the first, it is the precedence relation between tasks as described with \( \text{Prec} \) function.

As for the second, it is the precedence relation between the instances of the same task that must be specified in the dPTPN model.

In order to model the exchange of information according the dependency relations, we propose a set of places, called \( P_{\text{Task2Task}} \), defined in the following:

Let \( \text{TaskC}_1, \text{TaskC}_2 \in \text{TaskC} \) be two task models of \( T_1, T_2 \in \text{Task} \), respectively.

\[
\forall T_i \in \{ \text{TaskC}_1, \text{TaskC}_2 \} \text{ Create : } \\
\begin{cases}
P_{T_1T_2} \text{ in } P_{\text{Task2Task}}; \\
T_{\text{Sending}}, T_{\text{Receiving}} \text{ in } T; \\
B (T_i \rightarrow P_{\text{SendData}}; T_{\text{Sending}}) = 1; \\
B (P_{T_1T_2}, T_{\text{Receiving}}) = 1; \\
F (P_{T_1T_2}, T_{\text{Sending}}) = 1; \\
F (T_i \rightarrow P_{\text{ReceivedData}}; T_{\text{Receiving}}) = 1; \\
\end{cases}
\]

If \( \text{Prec} (T_1, T_2) = 1 \text{ Create : } \\
\begin{cases}
P_{T_1T_2} \text{ in } P_{\text{Task2Task}}; \\
B (P_{T_1T_2}, T_{\text{Receiving}}) = 1; \\
F (P_{T_1T_2}, T_{\text{Sending}}) = 1; \\
\end{cases}
\]

The second most important RTS component is the execution resource. In our study, we shed the light on the processor resources in order to execute the system tasks. Hence, 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. With dPTPN, each processor \( P_i \in \text{Proc} \) is modeled with a simple place and its marking describes the state of the resource.

Let \( P_{\text{res}} \): the set of places modeling the processors \( \text{Proc} (\in \Omega) \), with \( |P_{\text{res}}| = |\text{Proc}| \).

\[
\forall p \in P_{\text{res}}, M (p) = \begin{cases}
1: \text{the resource is free and ready to execute a task}; \\
0: \text{resource is occupied by a system task}; 
\end{cases}
\]
The allocation of the processor depends on the used scheduling strategy. In the current paper, we are interested in the strategy based on the Earliest Deadline First (EDF) \[12\].

Ω offers the function \(\text{Alloc}\) in the aim to specify the different partitions Tasks/Processors. The corresponding modeling with \(\text{dPTPN}\) is detailed as follows:

Let \(T_1, T_2 \in \text{Task}\) and \(P_1 \in \text{Proc}\) with \(\text{Alloc}(T_1) = \text{Alloc}(T_2) = P_1\);

In \(\text{dPTPN}\), the corresponding components are: \(\text{TaskC}_1, \text{TaskC}_2 \in \text{TaskC}\) and \(P_1 \in \text{P}_{\text{res}}\), with:

\[
\begin{align*}
T_{\text{allocation}} & \text{ in } T_{\text{cp}}; \\
T_{\text{releasing}} & \text{ in } T; \\
\mathbb{B}_{T_{\text{cp}}} (T_i \rightarrow P_{\text{Ready}}, T_{\text{allocation}}) & = 1; \\
\mathbb{B}_{T_{\text{cp}}} (P_1, T_{\text{allocation}}) & = 1; \\
\text{coef} (T_i \rightarrow P_{\text{RemainingPeriod}}, T_{\text{allocation}}) & = 1; \\
\mathbb{F}_{T_{\text{cp}}} (T_i \rightarrow P_{\text{getProc}}, T_{\text{allocation}}) & = 1; \\
\mathbb{B} (T_i \rightarrow P_{\text{Releasing}}, T_{\text{releasing}}) & = 1; \\
\mathbb{F} (P_1, T_{\text{releasing}}) & = 1;
\end{align*}
\]

∀\(T_i \in \{\text{TaskC}_1, \text{TaskC}_2\}\) Create :

After developing the RTS model with \(\text{dPTPN}\), it is important to precise its initial marking as described in previous research works \[9, 8\]. At this stage, the \(\text{dPTPN RTS}\) model is ready for execution to determine all its reachable states. So, we propose, in the next section an algorithm for the generation of its corresponding states graph.

### 4 dPTPN States Graph

In order to present the reachable states of the \(\text{dPTPN RTS}\) model, we propose a states graph \(G\), which defines all the states and edges connecting between them.

#### 4.1 Graph definition

The \(\text{dPTPN}\) states graph is defined with the triplet:

\[
G = (S, \tau, \rho)
\]

with:

- \(S = \{S_0, S_1, \cdots, S_n\}\): is a finite set of states with \(n > 0\);

Each state \(S_i \in S\) is determined with :

\[
S_i = \{II, OI, P_{\text{Task2Task}}, P_{\text{res}}\} \times \text{TaskC} \rightarrow \mathbb{N}
\]

\(S_i\) is presented as a matrix. The columns are the set of \(\text{TaskC}\) and the lines are the different places related to a \(\text{TaskC}\). In fact, the input/output
interfaces ($II$ and $OI$) and the communications places ($P_{task2task}$) are included in the $S_i$ lines. Besides, the processor resources places ($P_{res}$) are presented as matrix lines to describe the assignment of each task.

- $\tau = \{\tau_1, \cdots, \tau_m\}$: is a finite set of edges connecting states with $m > 0$;
- $\rho$: is an incidence relation indicating the successor of a given state through an edge and it is defined as follows:

$$\rho : S \times \tau \rightarrow S \cup \emptyset$$

$$(S_i, \tau_j) \mapsto \begin{cases} S_h & \text{if } S_h \text{ is a successor of } S_i; \\ \emptyset & \text{if } S_i \text{ has no successor}; \end{cases}$$

$S_0$ is the initial state creating from the initial marking of the places ($II$, $OI$, $P_{task2task}$, $P_{res}$) according to each component of $TaskC$. Next, we explain the generation of all successor states and the edges allowing the reachability.

### 4.2 States Graph generation

We aim to generate an oriented states graph. Indeed, it is a prediction of all states that $RTS$ can reach them in scheduling. Thus, from an initial state, with respect to temporal constraints, a successor is generated with the firing of the set of $dPTPN$ transitions constituting a graph edge.

To do so, firstly, we present the step of finding the successor state and, secondly, the connection of all founded states with edges.

The procedure (Algo. 1) presents the arrangement of the $dPFM$ activities. The procedure header defines three parameters:

- $dFTs$: is an output parameter to present the fired transitions;
- $M$: is an input/output parameter to present the input and the updated model marking;
- $Lt$: is an input/output parameter to specify the transitions timers.

#### 4.2.1 Finding Successor

Finding a successor, respecting the $dPTPN$ semantic, is the main objective of the $dPTPN$ firing machine. Thus, starting with a given marking, the $dPFM$ looks for the next marking with the firing of the valid and highest priority transitions.
The \( dPFM \) accelerates the firing process with the firing of a set of transitions, \( dFT_s \), simultaneously. This property is valid because the \( dPTPN \) deals with the conflict of enabled transitions problem via a dynamic calculus of priorities and only the transition with the highest value of priority is fired. Consequently, the \( dFT_s \) contains only independent transitions \([6, 13]\) that can all be crossed together. The novel resulting marking is the combination of a collection of sub-states that can be created if each enabled transition is fired apart. This technique is known as partial order reduction technique. The present work, we masks the marking of models into TaskCs and we are interested only to show the marking presenting the RTS model state. However, we respect the masked models implicitly in the firability, validity and selection steps. Hence, in addition to the fired transitions of the RTS model, the result parameter \( dFT_s \) defines the fired transitions of the masked models into TaskC components.

### 4.2.2 Algorithm of generation

The Algorithm (Algo .2) describes all the necessary steps to create the states graph according the \( dPTPN \) RTS model. Starting from \( dPTPN \) model, the function \texttt{initializeMarking}(M) allows the initialization of the marking vector \( M \) and the function \texttt{SetTimer} initializes the local timers. Thus, the initial state is created via the function \texttt{StateConstruction}(M). From this state, a repetitive process is executed to define all reachable successors states and each one is added to \( S \) set.
The \textit{dPFM} respects the time constraints during the generation of states and searches for a successor from each current state. These two properties of \textit{dPFM} give rise to an oriented states graph in which each state can exist only if their precedents are generated. The generation of the states and the relation of reachability are finished when one of three situations is verified. First, the marking of a place of the type $P_{\text{Deadline}}$ describes that its corresponding task is non-schedulable (the Boolean function \textit{CheckDeadline}(\textit{Succ}) is used in the algorithm for checking the marking of the $P_{\text{Deadline}}$ places). In fact, according to the \textit{dPTPN} model of the task, this marking is defined as a \textit{stop-Marking} but according to the entire \textit{RTS} model it is not. Therefore, the algorithm is stopped at this stage and a final state is announced. The second situation is when the current marking $M$ of all the existing places cannot enable any \textit{dPTPN}-Transitions and then the current state $S_{\text{indexState}}$ is considered as a final state. As for the third situation, the generation is terminated when the updated marking $M_I$ gives rise to a state existing in $S$ and the corresponding edge $\tau_{\text{indexEdge}}$ connects the current state with the existing one.
5 Open Problem

After the generation of the states graph corresponding to the \textit{dPTPN} model, it is interesting to check the schedulability in this graph. In fact, based on graph theory, many properties can be checked such as the vivacity of graph. Nevertheless, the schedulability is not a graph property, so, it is primordial to translate it into graph properties and then its checking is available. The checking provides a confirmation of the schedulability or a counterexample otherwise. Thus, this result can be an efficient feedback to the partitioning tool in order to decrease the exploration complexity of the HW/SW space.

6 Conclusion

The \textit{dynamic Priority Time Petri Nets (dPTPN)} is considered as the first \textit{Petri Nets} extension dedicated for \textit{Real-Time System (RTS)} scheduling analysis with dynamic priority. Its mathematical presentation is able to specify the time constrains and a dynamic calculation of priorities in order to deal with transitions conflict problem. Besides, its semantics, presented by the \textit{dynamic Firing Machine (dPFM)}, accelerates the firing process of transitions via the firing of independent transitions set detected with order partial techniques. In the aim to showing all reachable states of a \textit{dPTPN RTS} model, we have proposed in the present manuscript, a generation method of states graph. Compared to the existing techniques of graph generation, we can benefit from a reduced \textit{dPTPN} model and we generate its corresponding graph. In fact, the existing research works are interesting to generate a simple initial graph and then they apply reduction techniques. However, such methods require the validation of the resulting graph relating to conserving the properties compared to the initial one. In our case, we started from a reduced model, based on hierarchical modeling [8], conserving the main properties of the initial model, and we propose its corresponding graph. The generated graph is a prediction of the \textit{RTS} scheduling, and analyzing its properties is an efficient solution to derive the schedulability of the system.

References


