Hierarchical modeling impact on states graph generation for the dynamic Priority Time Petri Nets

Adel Mahfoudhi[1] and Walid Karamti[2]

**[1] College of Computers and Information Technology, Taif University
Taif, Saudi Arabia**

**[2] CES Laboratory, ENIS, University of Sfax
ENIS Soukra km 3,5, B.P.:w 1173-3000, Sfax, Tunisia**

Abstract

In previous work we have proposed the hierarchical modeling in order to reduce the dynamic priority Time Petri Nets (dPTPN) model. In the present paper, we focus on the demonstration of the impact of this modeling on the generation of a reduced states graph. We aim to explore the abstraction of the hierarchical modeling in order to produce a graph describing only the interaction states of a real-time system (RTS). Thus, a new definition of states graph is given in this paper, and the corresponding generation algorithm is detailed.

Keywords: *dPTPN, RTS, Hierarchical modeling, states graph.*

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 [1]. Recently, a new generation of architectures (i.e. Multiprocessor and Multicore) has emerged [18]. 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 mainly shed on model checking approaches, which attempt to provide a push-button approach to verification and integrate well into standard development processes. However, since using such technique can derive a state-explosion problem, it is primordial to master the states generation before the properties checking.

Before verification can be applied, the system must be modeled in a formal description language such as Time Petri Nets [15], timed automata. In order to deal with the state-explosion problem, and based on Time Petri Nets model, Berthomieu in [3] 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) [4] so as to analyze the schedulability. An improvement of this approach was proposed in [17], in which the authors propose a new extension of Time Petri Nets STPN (Scheduling Timed Petri Nets) and a reduced states graph compared to [3].

Both of [3] and [17], 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 [7]. So, it is not efficient to use them in scheduling analysis of multiprocessor system.

In [11], 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) [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) [13]. 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 [10].

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 have 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 [9], 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 state-explosion problem, we start from a reduced model and we produce a finite and oriented states graph where the schedulability can be checked on it immediately (without observer's 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 of the proposed 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 from the developed model, a set of states connected with edges are generated to describe a prediction of the RTS scheduling. Finally, the proposed approach is briefly outlined.

2. dynamic Priority Time Petri Nets: dPTPN

A Petri Nets [16] can be defined as 4-tuplet :

$$PN = <P, T, B, F> \tag{1}$$

,where:
(1) $P = \{p1, p2, ..., pn\}$ is a finite set of places $n > 0$;
(2) $T = \{t1, t2, ..., tm\}$ is a finite set of transitions $m > 0$
(3) $B :(P \times T) \rightarrow N$ is the backward incidence function;
(4) $F :(P \times T) \rightarrow N$ is the forward incidence function;
Each system state is represented by a marking M of the net and defined by : $M : P \rightarrow N$.
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 Tcp is a transition with a preprocessing that precedes the crossing to calculate its priority. Indeed, if two Tcp transitions are enabled and share at least a place in entry, then the preprocessing is made to determine the transition which will be fired, with a priority changing according to the state of the network described by the marking M.

The dPTPN is defined by the 7-tuplet :

$$dPTPN =<PN,Tcp,Tf ,BT ,FT ,coef,M0> \tag{2}$$

(1) PN: is a Petri Nets;
(2) $Tcp = \{Tcp1,Tcp2, \cdots ,Tcp \}$: is a finite set of compound transition $k> 0$;
(3) $Tf : T \rightarrow Q+$ is the firing time of a transition.
$\forall t \in T$, t is a temporal transition $\Leftrightarrow Tf (t) \neq 0$.
If $Tf (t)=0$,then t is an immediate transition. Each temporal transition t is coupled with a local timer (Lt (t)), with $Lt : T \rightarrow Q+$.
(4) $BT :(P \times Tcp) \rightarrow N$ is the backward incidence function associated with compound transition;
(5) $FT :(P \times Tcp) \rightarrow N$ is the forward incidence function associated with compound transition;
(6) $coef :(P \times Tcp) \rightarrow Z$ is the coefficient function associated with compound transition;
(7) M0 : is the initial marking;
The semantics of firing in dPTPN is based on the partial order theory 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 dFTs composed of enabled temporal transitions FTs and enabled compound transitions FTsTcp . The initialization is called Firiability processing.

$$dFTs = FTs \cup FTsTcp \tag{3}$$

$$\begin{aligned} &\text{let } t\in T,\ t\in dFTs \Leftrightarrow t\in FTs \vee t\in FTsTcp \\ &\text{with : } \quad FTs = \{t\in T/B (., t)\leq M\} \\ &\qquad\qquad FTsTcp=\{t\in T/BTcp (., t)\leq M\} \end{aligned} \tag{4}$$

Next, valid transitions are selected from FTs to V Ts by applying the Validity processing. All urgent transitions must be indicated in V Ts to be ready for firing.

$$VTs = \{t\in FTs/Hl (t) = Tf (t)\} \tag{5}$$

The dFTsTcp 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 dFTsTcp 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 Tcp transition having the highest priority according to its neighborhood (eq. 6).

$$\forall Tcp1,Tcp2 \in Tcp,\ Tcp1 \text{ is a neighbor of } Tcp2 \Leftrightarrow \exists p\in P \text{ such that } BTcp (p,Tcp1)\neq 0 \wedge BTcp (p,Tcp2) \neq 0 \tag{6}$$

In this step, the proposed dPTPN is able to support a selection policy. In [11], the authors have proved that dPTPN can attribute the priorities to transitions sharing the place processor according to the LLF policy. In [10] 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:

$\forall FT \in \{VTs, FTsTcp\}$, Firing (FT)$\Rightarrow$

FT = VTs, M'=M $+\sum_{t\in FT}((F(.,t)-B(.,t))$

or

FT = FTsTcp, M'=M $+\sum_{t\in FT}((FTcp(.,t)-BTcp(.,t))$ (7)

More details about the dpfm and the firing process can be found in [11].

3. Multiprocessor RTS modeling with dPTPN

The dPTPN is dedicated for analyzing the schedulability of the RTS running on multiprocessor architecture [11, 10]. 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:

$$\Omega = \langle Task, Proc, Alloc, Prec\rangle \quad (8)$$

with:

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

$$Taski = \langle Ri,Pi,Ci\rangle \quad (9)$$

- Ri: the date of the first activation
- Pi: the period associated with the task
- Ci: the execution period of the task for the Pi period

(2) Proc : a finite set of processors.

(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.

$\forall t1 \in$ Task, $\forall$ P1,P2 $\in$ Proc,

$$Alloc(t1)= P1 \wedge Alloc(t1)= P2 \Rightarrow P1 = P2 \quad (10)$$

(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 [11, 10], 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 [9]. 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:

$$TaskC = \langle dPTPN,II,OI\rangle \quad (11)$$

with:

(1) dPTPN: is the task dPTPN model presented in ([9], Fig4);

(2) II = {PUncreated,PReceivedData,PgetProc}: is the place that composes the Input Interface;

(3) OI= {PReady, PRemainingPeriod, PSendData, PReleasing,PDeadline} : is the place that composes the Output Interface;

The dependency between tasks is specified in Ω with the function Prec. However, in dPTPN modeling, we distinguish between two dependency relations.

Concerning the first, it is the precedence relation between tasks as described with 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 PTask2Task, defined in the following:

Let TaskC1,TaskC2 $\in$ TaskC be two task models of T1,T2$\in$ Task, respectively.

$\forall$ Ti $\in$ {TaskC1,TaskC2} Create :

PTi2Ti in PTask2Task;
TSending,TReceiving in T;
B (Ti $\rightarrow$ PSendData,TSending)=1;
B (PTi2Ti,TReceiving)=1;
F (PTi2Ti,TSending)=1;
F (Ti $\rightarrow$ PReceivedData,TReceiving)=1;

If Prec (T1,T2)=1Create:

PT12T2 in PTask2Task;
B (PT12T2 ,TReceiving)=1;
F (PT12T2 ,TSending)=1;

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 Pi$\in$ Proc is modeled with a simple place and its marking describes the state of the resource.

Let Pres : the set of places modeling the processors Proc ($\in$ Ω), with |Pres| = |Proc|.

$\forall$ p $\in$ Pres,M (p)=

1: the resource is free and ready to execute a task;
Or
0: resource is occupied by a system taskΩ;

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) [13].

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

Let T1,T2 $\in$ Task and P1 $\in$ Proc with Alloc (T1)= Alloc (T2)= P1;

In dPTPN, the corresponding components are: TaskC1,TaskC2 $\in$ TaskC and P1 $\in$ Pres, with:

$\forall$ Ti $\in$ {TaskC1,TaskC2} Create :

Tiallocation in Tcp;
Tireleasing in T;
BTcp (Ti $\rightarrow$ PReady,Tiallocation)=1;
BTcp (P1,Tiallocation)=1;
coef (Ti $\rightarrow$ PRemainingPeriod,Tiallocation)=1;
FTcp (Ti $\rightarrow$ PgetProc,Tiallocation)=1;
B (Ti $\rightarrow$ PReleasing,Tireleasing)=1;
F (P1,Tireleasing)=1;

After developing the RTS model with dPTPN, it is important to precise its initial marking as described in previous research works [10, 9].

At this stage, the 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 dPTPN RTS model, we propose a states graph G, which defines all the states and edges connecting between them.

4.1 Graph definition

The dPTPN states graph is defined with the triplet:

$$\mathbf{G} = <S, \tau, \rho> \tag{12}$$

with:

• S = {S0,S1, ··· ,Sn}: is a finite set of states with n> 0;

Each state Si $\in$ S is determined with :

$$Si = \{II, OI, PTask2Task,Pres\} \times TaskC \rightarrow N \tag{13}$$

Si is presented as a matrix. The columns are the set of TaskC and the lines are the different places related to a TaskC. In fact, the input/output interfaces (II and OI) and the communications places (PTask2Task) are included in the Si lines. Besides, the processor resources places (Pres) are presented as matrix lines to describe the assignment of each task.

• τ = {τ1, ··· ,τm}: is a finite set of edges connecting states with m> 0;

• ρ: is an incidence relation indicating the successor of a given state through an edge and it is defined as follows:

$$\begin{aligned} \rho : \quad & S \times \tau \rightarrow S \cup \varnothing \\ & (Si,\tau j) \rightarrow Sh \text{ if } Sh \text{ is a successor of } Si; \\ & \qquad \varnothing \text{ if } Si \text{ has no successor;} \end{aligned}$$

S0 is the initial state creating from the initial marking of the places (II, OI, PTask2Task, Pres) 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.

Procedure 1 dPFM (**var** dFT_s, **var** M, **var** Lt)

```
$dFT_s = \oslash$
if $M \geq B$ or $M \geq B_{T_{cp}}$ then
  Firability($dFT_s$)
  $FT_s \leftarrow$ temporalTransition($dFT_s$)
  $FT_{s_{T_{cp}}} \leftarrow$ CompoundTransition($dFT_s$)
  if $FT_s \neq \oslash$ then
    $VT_s \leftarrow Validity(FT_s, Lt)$
    if $VT_s \neq \oslash$ then
      Firing($VT_s$,$M$)
      ResetTimer(Lt($VT_s$))
    else if $FT_{s_{T_{cp}}} = \oslash$ then
      SetIncrementTimer(Lt($FT_s$))
    end if
  else if $FT_{s_{T_{cp}}} \neq \oslash$ then
    StepSelection($FT_{s_{T_{cp}}}$)
    Firing($dFT_s$,$M$)
  end if
end if
```

The dPFM accelerates the firing process with the firing of a set of transitions, dFTs, 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 dFTs contains only independent transitions [6, 14] 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 dFTs 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 initializeMarking(M) allows the initialization of the marking vector M and the function SetTimer initializes the local timers. Thus, the initial state is created via the function StateConstruction(M). From this state, a repetitive process is executed to define all reachable successors' states and each one is added to S set.

Algorithm 2 State graph generation

```
begin
initializeMarking($M$)
SetTimer(Lt)
$S_0 \leftarrow StateConstruction(M)$
$indexState \leftarrow 0$, $indexEdge \leftarrow 0$
repeat
  $S \leftarrow S \cup \{S_{indexState}\}$
  $\tau_{indexEdge} \leftarrow \oslash$, $W \leftarrow M$
  repeat
    dPFM($dFT_s$,$M$,Lt)
    $\tau_{indexEdge} = \tau_{indexEdge} \cup \{dFT_s\}$
  until $W \neq M$ or $\tau_{indexEdge} = \oslash$
  if $W \neq M$ then
    $Succ \leftarrow StateConstruction(M)$
    $\tau \leftarrow \tau \cup \{\tau_{indexEdge}\}$
    $\rho(S_{indexState}, \tau_{indexEdge}) \leftarrow Succ$
    if CheckDeadline($Succ$)= True then
      $FinalState \leftarrow Succ$
      $S \leftarrow S \cup \{FinalState\}$
    end if
  else
    $FinalState \leftarrow S_{indexState}$
  end if
  $indexState \leftarrow indexState + 1$
  $indexEdge \leftarrow indexEdge + 1$
until $Succ \in S$ or $FinalState \neq \oslash$
end
```

The dPFM respects the time constraints during the generation of states and searches for a successor from each current state. These two properties of 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 PDeadline describes that its corresponding task is non-schedulable (the Boolean function CheckDeadline(Succ) is used in the algorithm for checking the marking of the PDeadline places). In fact, according to the dPTPN model of the task, this marking is defined as a stop-Marking but according to the entire 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 dPTPN-Transitions and then the current state SindexState is considered as a final state. As for the third situation, the generation is terminated when the updated marking MI gives rise to a state existing in S and the corresponding edge τindexEdge connects the current state with the existing one.

After the generation of the states graph corresponding to the dPTPN model, it is interesting now 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 counter example 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.

5. Conclusions

The dynamic Priority Time Petri Nets (dPTPN) is considered as the first Petri Nets extension dedicated for 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 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 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 RTS 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 [9], conserving the main properties of the initial model, and we propose its corresponding graph. The generated graph is a prediction of the RTS scheduling, and analyzing its properties is an efficient solution to derive the schedulabilty of the system.

References

[1] D. B. Abdullah and Z. A. Thanoon. Real time network server monitoring using smartphone with dynamic load balancing. IJSCI, 10(1):227–232, july 2013.

[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] 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.

[6] 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.

[7] 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.

[8] J.Goossens and P.Richard. Overview of real-time scheduling problems. In Euro Workshop on Project Management and Scheduling, 2004.

[9] W. Karamti, A. Mahfoudhi, and Y. Hadj Kacem. Hierarchical modeling with dynamic priority time petri nets for multiprocessor scheduling analysis. In ESA, The 2012 International Conference on Embedded Systems and Applications, pages 114–121, 2012.

[10] W. Karamti, A. Mahfoudhi, and Y. Hadj Kacem. Using dynamic priority time petri nets for scheduling analysis via earliest deadline first policy. In ISPA, pages 332–339, Madrid, Spain, 2012.

[11] 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.

[12] 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.

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

[14] 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, 59(3):1478–1503, 2012.

[15] 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.

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

[17] 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.

[18] M. Safar, M. A. El-Moursy, M. Abdelsalam, and A. Salem. Architecture exploration of multicore systems-on-chip using a TLM-based framework. IJSCI, 10(1):4–7, july 2013.

Adel MAhfoudhi is currently an Associate Professor at the University of Taif in Saudi Arabia. He obtained a Diploma in computer engineering in 1992 from the University of Monastir in Tunisia, received his Ph.D. degree in Computer Engineering in 1997 from the University of Valenciennes in France. His current research interests are formal methods for Embedded Real Time System modeling and verification. He is author/co-author of several papers in international conferences and journals.
.

Walid Karamti received a Master degree in Computer science in 2010 from the University of Sfax, Tunisia where he is now completing his Ph.D. thesis. His fields of interest are Real-Time System, Multiprocessor architecture and Model checking methods for scheduling analysis.