E. Grolleau and A. Choquet-Geniet, Off-line computation of real-time schedules by means of Petri nets, Workshop On Discrete Event Systems, WODES2000, *Kluwer Academic Publishers, Ghent, Belgium, pp. 309-316, 2000*

Off-Line Computation of Real-Time Schedules by Means of Petri nets

Emmanuel Grolleau and Annie Choquet-Geniet
LISI-ENSMA, Téléport 2 - 1, rue Clément Ader, BP 40109, 86961 Futuroscope-Chasseneuil Cedex, France, grolleau@ensma.fr, ageniet@ensma.fr

Keywords: Petri nets, earliest firing rule, real-time systems, off-line scheduling.

Abstract: We present an off-line methodology of analysis of real-time systems, composed of periodic, precedence and resource constrained real-time tasks. As there is no polynomial optimal scheduling technique for such tasks sets, we present an enumerative method based on the construction of the state graph of a Petri net. The time is modeled by the Petri net through the earliest firing rule.

1. INTRODUCTION

Real-time systems, most of the time dedicated to process control, are characterized by temporal parameters, induced by the dynamic of the controlled process. We assume that these parameters are a priori known, i.e. we are only interested in deterministic real-time systems, since they are the only systems for which the respect of the temporal constraints can be guaranteed.

A real-time application is designed as a set of mostly periodic interacting tasks, with fixed temporal characteristics. Once the application is designed, and the functional correctness proven, the system must be temporally validated, i.e. the temporal correctness must be proven, which expresses that all the temporal constraints are met, provided an appropriate scheduling policy is used. Two approaches are usually considered in order to solve the scheduling problem. The on-line approach : a scheduling policy is implemented within the scheduler ; and the off-line approach : a schedule is stored in a table used by a dispatcher.

The scheduling algorithms used on-line are based on priorities, mostly derived from the temporal parameters (e.g. Rate Monotonic, Earliest Deadline, Least Laxity), and they are polynomial in time [5, 6](see [8] for a survey). Under some specific assumptions (e.g. for Earliest Deadline,

independent tasks or precedence constrained tasks), some of them are optimal in the following sense : a scheduling policy is said to be optimal if, for a given task system, either the policy computes a feasible[1] schedule, or there is no feasible schedule of the task system. The scheduling problem becomes NP-hard [7] when shared resources are involved (e.g. shared memory or control terminal). The main difficulty comes from the duration of the blockage of a task requesting a resource. In order to analyze the schedulability of such a task system, the duration of each task is increased by the duration of the longest blockage it could suffer. In this case, we then only dispose of sufficient conditions of feasibility. Furthermore, the temporal parameters of the tasks may then become unrealistic, this implies that the more the number of task interactions by means of shared resources increases, the more the efficiency of feasibility tests of on-line schedulability decreases. A second difficulty comes from the lack of optimal on-line algorithms for task systems where critical resources are involved.

Off-line scheduling methodologies have been studied in order to validate highly constrained task systems. The methodologies are usually exponential in time, but can lead to a feasible schedule, provided such one exists. Those approaches are either exhaustive (enumeration techniques based on Petri net modeling [3, 4] or branch and bound techniques [10, 1, 2]), or stochastic (genetic algorithms, simulated annealing).

We present a methodology based on a fine modeling of a task system, and on the modeling of time by means of the earliest firing rule [9]. We then present some properties of the obtained Petri net and of its state graph. Finally, we show how specific feasible schedules can be computed.

2. THE TASK MODEL

We deal only with periodic tasks. We suppose that sporadic tasks with hard deadlines are modeled by periodic servers, and that background scheduling is used on-line for non-periodic tasks with soft deadlines. Each periodic task τ_i is characterized by four temporal parameters [6]:

- r_i, its first release date
- C_i, its processor load, or its worst-case computation time on each release
- D_i, its relative deadline, the time by which τ_i has to be computed relative to its last release
- P_i, its release period: τ_i is released at the dates r_i+kP_i, $k \in \mathbb{N}$.

[1] A schedule where all tasks meet their deadlines.

For short, we note $\tau_i<r_i,C_i,D_i,P_i>$, and $S=\{\tau_i<r_i,C_i,D_i,P_i>\}_{i=1..n}$ denotes a task system. We suppose that $min_{i=1..n}\{r_i\}=0$. We denote $r=max_{i=1..n}\{r_i\}$. The tasks are said synchronous if $r=0$, and asynchronous if $r>0$.

Our aim is to compute an infinite schedule meeting all the deadlines of the tasks. Moreover, when interactions of tasks are involved (message passing and use of critical resources), the implied structural constraints (precedence constraints and mutual exclusion) must be respected.

Communications are modeled by one to one mailboxes, and by assumption, the emission rate to a mailbox is equal to the reception rate. The resources can be accessed either in write mode, or in read-only mode (i.e. several read-only accessed can occur simultaneously). Moreover, some parts of the tasks can be non-preemptible.

3. THE MODELING STEP

As the interactions of the tasks are complex, the expressiveness of a Petri net (PN) model is appropriate. Furthermore, operational semantic involving the time must be associated to the PN modeling the tasks system. We assume that a quantum of preemption is defined: a running task can be preempted by any pending task at each preemption point. Therefore, the scale used to express the time is given in terms of preemption points, and a preemptible task can be preempted each time unit.

3.1 The modeling of time

3.1.1 The earliest firing rule (EFR)

A PN obeys the EFR if each time a firing of transitions occurs, a maximal set of transitions simultaneously enabled fires: let I be the set of enabled transitions for a marking M (note that I is not a multiset, since a transition t cannot fire more than once in the same firing step). A maximal set of transitions simultaneously enabled $I' \subseteq I$ is defined by: $\forall t \in I \setminus I'$, t is in conflict with some transitions of I' (i.e. the transitions of I are not in conflict, and any other enabled transition is in conflict with some transition of I').

A PN with the EFR has been shown to be equivalent to a timed PN at maximal speed [9]. It behaves exactly like a timed PN at maximal speed where all the durations of the transitions are one, nevertheless, their implementation is easier. Furthermore, the expressiveness of PN with the EFR is equivalent to the expressiveness of a Turing machine, therefore it can model the whole set of interactions between tasks.

3.1.2 The modeling of periodic actions

The *Figure 1*.a presents the basic PN component which models the periodicity of an elementary action *e*, occurring with period P_e (with P_e>1).

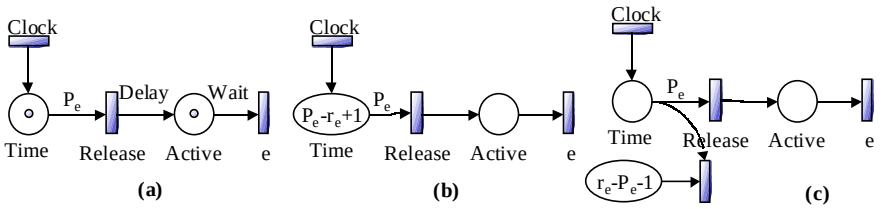

Figure 1.(a) Petri net model for a periodic action *e* (b) *e* is differed by $r_e \leq P_e+1$ (c) *e* is differed by $r_e > P_e+1$.

The transition *Clock* is a source transition which is always enabled without any conflict. It follows that each time a maximal set of transitions is fired, the transition *Clock* is fired, producing a token in the place *Time* which acts like a local time marker. Thus, we then assimilate the firing of *Clock* to a time unit in our scale of time. As soon as *Time* holds P_e tokens, the transition *Release* is fired (simultaneously with *Clock* which is always fired), so a token is produced in the place *Active* each P_e units of time at the dates $(k\text{-}1)P_e$ for $k \in \mathbb{N}$. Finally, as each time *Active* contains a token, the transition *e* is fired, the action *e* occurs at the dates kP_e.

The same principle is used to model differed periodic action (first release date greater than 0). If the first release date of the action *e* is $r_e \leq P_e+1$, the model differs from the previous model only by the initial marking (see *Figure 1*.b), and if $r_e > P_e+1$, a place and a transition are added to the model (see *Figure 1*.c).

On *Figure 1*.c, the transition *Wait* is fired during the $r_e - P_e - 1$ first units of time. Then the place *Time* needs P_e units of time to enable *Release*, which is fired at the date $r_e - 1$, so *e* is fired for the first time at the date r_e.

3.2 The complete model

The *Figure 2*. represents a Petri net modeling the task system $S=\{\tau_1<r_1=1,C_1=2,D_1=3,P_1=4>,\ \tau_2<r_2=0,C_2=1,D_2=2,P_2=2>\}$. Two main parts compound the model : the clock system, and the task system.

3.2.1 The clock system

The clock system models the behavior of the time in paralleling the clock systems of section 3.1.2. The body of the tasks can be complex and is described in the next section, but remark that the $Release_i$ transition produces a token of color *a* (for activation) in $Active_i$, and that when the task

completes its execution, a *b*-token is produced in $Active_i$. This token means that the task has completed its last execution before another one. This token allows the PN to forbid task reentrance. In addition, some constraints are used on the allowed markings in order to model the deadlines of the tasks.

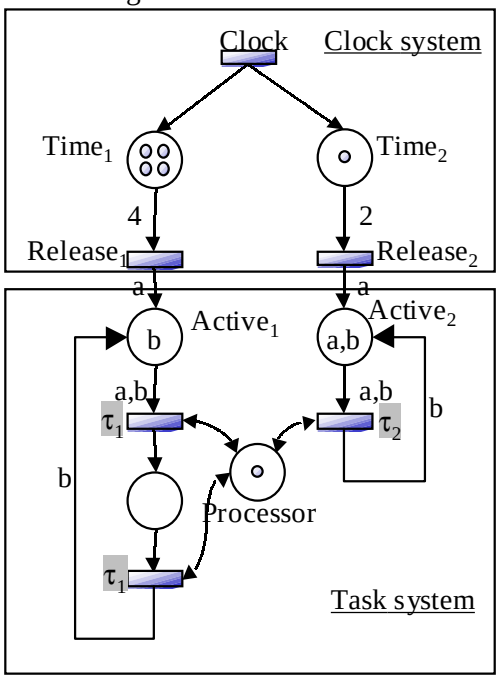

Figure 2. Petri net model for the task system $S=\{\tau_1<r_1=1,C_1=2,D_1=3,P_1=4>, \tau_2<r_2=0,C_2=1,D_2=2,P_2=2>\}$

When $Time_1$ contains *4* tokens, the task τ_1 has been released for *3* time units, thus, it must have completed its execution (i.e. $M(Active_1)=\{b\}$). It follows that the deadline of τ_1 is met if and only if $M(Time_1)=4 \Rightarrow M(Active_1)=\{b\}$ **(i)**.

For the particular case of τ_2 whose deadline $D_2=P_2$, $Time_2$ cannot contain more than D_2 tokens. But as soon as $M(Time_2)=1$, τ_2 is released, so it must have completed its last execution. It follows that τ_2 meets its deadline if and only if $M(Time_2)=1 \Rightarrow M(Active_2)=\{a,b\}$ **(ii)**. The constraints (i) and (ii) are called terminal constraints and a marking meeting these constraints corresponds to a state of the system where all deadlines are met. Thus, an infinite behavior of the Petri net where all the reached markings meet the terminal constraints corresponds to a behavior of the modeled system where all the deadlines are met. For a task τ_i, the terminal constraints are given by: if $D_i<P_i$, $M(Time_i)=D_i+1 \Rightarrow M(Active_i)=\{b\}$, and if $D_i=P_i$,

$M(Time_i)=1 \Rightarrow (M(Active_i)=\{a,b\}$ or $M(Active_i)=\{b\})$ (the last case covers the case where $r_i \geq P_i$).

3.2.2 The task system

Each task body of duration C_i of the tasks system is modeled by a series of C_i transitions (note that a series of $n \geq 3$ transitions can be compressed into a series of *3* transitions), each of them using a common resource: the processor. It follows that one transition of the tasks can be fired at a time. Interactions between the tasks can be modeled in an usual way by mailbox places, and resource places (in exclusion mode or in read/write mode). The beginning of a non-preemptible part is modeled by the fact that the first transition of the part uses the processor but does not restitute it: the processor is liberated by the last transition of the non-preemptible component.

Since we focus on the sequence of actions, each component transition of τ_i is labeled with τ_i, and the other transitions are labeled with the empty word. It follows that the language of the PN where all the reached markings meet the terminal constraints (the center of the terminal language) is exactly the set of feasible conservative[2] schedules for the modeled system. It must be noted that it exists some tasks system for which no feasible conservative schedule exist but some feasible non-conservative schedules exists. In order to obtain non-conservative schedules, a idle task is added to the PN model. This task handles the idle slots and extends the language to the non-conservative schedules. Thus the center of the terminal language is the whole set of feasible schedules for the modeled system: each word of this language (i.e. each path in the state graph) is a feasible schedule.

4. ANALYSIS OF THE SYSTEM

Since the length of each word is infinite, we have to show that each word w of the PN language can be written $w_a w_c^*$ (where * is the Kleene star). In the case of synchronous task systems, at the date 0, all the tasks are simultaneously released. One major period later $P=lcm_{i=1..n}\{P_i\}$, all the tasks are in the same state as at the date 0. It follows that in this case, the marking of the PN at the date P is the same as M_0. Thus, in the state graph of the PN, M_0 is an home marking, and each word can be written w_c^* where $|w_c|=P$ ($|w|$ denotes the length of w).

[2] A conservative schedule does not let the processor idle while some pending tasks are to process.

In the case of asynchronous task systems, we have shown in [4] that for all feasible schedule, a date t_c can be computed, with $t_c \leq r+P$. This date is unique for a task system, and the state of a system at the date t_c is the same as at the date t_c+P. It follows that each word can be written $w_a w_c{}^*$ where $|w_a| = t_c \leq r+P$ and $|w_c| = P$. Thus the state graph contains a marking at the depth t_c which is an home marking.

In both cases, the depth of the state graph is bounded, and contains an home marking.

The state graph contains the entire set of feasible schedules, and it is possible to extract the optimal schedules for several criteria (based on the response time, the lateness or the reaction rate) on a selected set of tasks. This particular pattern allows us to extract, in a complexity linear in the size of the graph, a sub-graph containing the whole set of optimal schedules for a given criteria. As an example, in order to extract the schedules minimizing the average response time of τ_i and τ_j, each edge of the state graph corresponding to a termination of τ_i or τ_j is weighted by its corresponding response time (which is computed from the depth of the edge). The other edges are weighted by *0*. The optimal schedules for the given criterion are the paths whose cost is minimal. Since each word is a permutation of another, the graph is a diamond, therefore the computational complexity of the shortest path search algorithm is linear in the size of the state graph.

The optimal schedules for the applied criterion are a sub-graph of the state graph, and the technique can be applied recursively on this sub-graph in order to obtain the optimal schedules for several criteria (e.g. best worst reaction rate for τ_k,…).

Note finally that the fact that a finite word *w* reaches only valid markings does not imply that it exists an infinite word in the center of the terminal language whose prefix is *w*. Thus, the construction of the state graph of the center of the terminal language implies backtracking (i.e. some nodes have to be given up). This phenomenon can be reduced thanks to optimization: a necessary condition for schedulability can be tested for each reached marking.

Several heuristics are used in order to reduce the (non polynomial) size of the state graph. One of them significantly reduces the graph in forbidding non necessary preemptions: two concurring parts of tasks, independent from each others cannot interleave. The main advantage of this heuristic is that it does not reduce the scheduling power of the model (i.e. if there exists a feasible schedule, then this heuristic let at least one schedule).

5. CONCLUSION

We present a Petri net with the earliest firing rule which models periodic real-time task systems. The center of its terminal language is exactly the set of all feasible schedules for the modeled system. Since the state graph is bounded, it can be constructed. We use several heuristics in order to reduce it without reducing the scheduling power of the model: in fact it is an optimal scheduler. In our knowledge, our method is the only off-line method dealing with asynchronous periodic tasks, complex communications through mailboxes, resources accessed in read-only or read/write mode, non-preemptible parts. A tool based on this method, called *PeNSMARTS* for *Petri Net Scheduling, Modeling & Analysis of Real-Time Systems*, has been developed.

References

[1] K. R. Baker and Z. S. Su, Sequencing with due-dates and early start times to minimize maximum tardiness, Naval Research Logistic Quarterly, 21 (1974), pp. 171-176.

[2] P. Bratley, M. Florian and P. Robillard, Scheduling with earliest start and due date constraints on multiple machines, Naval Research Logistic Quarterly, 22 (1975), pp. 165-173.

[3] A. Choquet-Geniet, F. Cottet and D. Geniet, Exhaustive computation of the scheduled task execution sequences of a real-time application, FTRTFP, Lecture Notes in Computer Science, 1135 (1996).

[4] E. Grolleau, Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de Petri en environnement monoprocesseur et multiprocesseur, Université de Poitiers-ENSMA, 1999, pp. 235.

[5] J. Leung and M. Merrill, A note on preemptive scheduling of periodic real-time tasks, Information Processing Letters, 11 (1980), pp. 115-118.

[6] C. L. Liu and J. W. Layland, Scheduling algorithms for multiprogramming in real-time environment, Journal of the ACM, 20 (1973), pp. 46-61.

[7] A. K. Mok, Fundamental design problems of distributed systems for the hard real-time environment, Department of Electrical Engineering and Computer Science, Massachussets Institute of Technologie, Cambridge, 1983.

[8] J. A. Stankovic, M. Spuri, M. D. Natale and G. Buttazzo, Implications of classical scheduling results fo real-time systems, IEEE Computer, 28 (1995), pp. 1-24.

[9] P. H. Starke, Some properties of timed nets under the earliest firing rule, Advances in Petri nets 1989, Venice in Lecture Notes in Computer Science, 424 (1990), pp. 418-432.

[10] J. Xu and D. Parnas, Scheduling processes with release times, deadlines, precedence, and exclusion relations, IEEE Transactions on Software Engineering, 16 (1990), pp. 360-369.