SCHEDULING REAL-TIME SYSTEMS BY MEANS OF PETRI NETS

Emmanuel Grolleau and Annie Choquet-Geniet

LISI-ENSMA, Teleport 1, BP 40109, 1 avenue Clement Ader,86961 Chasseneuil-Futuroscope Cedex, FRANCE

Abstract: We focus on the off-line scheduling of periodic real-time task systems where the first release date of some tasks can differ from the others. We first determine the length of the sequences to construct off-line, which was uninvestigated in the general case. The proposed method is based on the simulation of a Petri net and allows the extraction of optimal schedules regarding several criteria for a chosen set of tasks (e.g. minimizing response time, maximizing importance, ...)

Keywords: Real-time systems, Scheduling algorithms, Petri-nets

1. INTRODUCTION

[1] 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.

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 pre-run-time 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 (Leung and Merrill, 1980; Liu and Layland, 1973)(see (Stankovic *et al.*, 1995) 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[2] schedule, or there is no feasible schedule for the task system. The scheduling problem becomes NP-hard (Mok, 1983) when shared resources are involved (e.g. shared memory or control terminal). The main difficulty comes from the blockages due to the fact that a task can wait for a resource locked by a lower priority task. If specific resource management protocols (Baker, 1991; Chen and Lin, 1990) are used, the blockage of a task is bounded by the longest critical section of the lower priority tasks. This implies that, in order to analyze the schedulability of a task system, the duration of the critical sections of the tasks have to be increased of the duration of the longest blockage they could suffer. But the temporal parameters 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.

[1] IFAC Workshop on Real-Time Programming, WRTP'2000, Palma de Mallorca, may 17-19 2000

[2] A feasible schedule is an infinite schedule where all the temporal constraints are met

Off-line scheduling methodologies have been studied in order to validate highly constrained task systems. Those approaches are either exhaustive (Petri net modeling (Choquet-Geniet *et al.*, 1996; Grolleau, 1999), branch and bound technique (Bratley *et al.*, 1973; Xu and Parnas, 1990)), or stochastic (simulated annealing, genetic algorithms,). A completely deterministic schedule can then be implemented. Off-line approaches could seem less flexible than on-line ones, in particular regarding to aperiodic tasks which could occur during the life of the process, but as the schedule is known in advance, idle slots can be accurately handled by an on-line scheduler, particularly if those slots are thoroughly distributed upon the pre-run-time schedule (Grolleau, 1999).

In the case of off-line scheduling, the length of a pre-run-time schedule to compute has to be known. The off-line approaches are mostly dealing with non periodic task systems, but, using a theorem of (Leung and Merrill, 1980), the authors claim that their approaches can be applied to periodic task systems where all the tasks are first released simultaneously. These task systems are called synchronous task systems. In fact, in this case, at the date 0, each task is released, and one hyperperiod later, at the date $P = lcm$(periods of the tasks) (where lcm is the least common multiple) each task is released again, and if the schedule is feasible, the task system is in the same state than at the date 0. Therefore, in this case, each periodic task τ_i with period P_i is decomposed into $\frac{P}{P_i}$ non periodic tasks. The problem of determining the cyclicity of schedules for non synchronous task systems has been investigated in (Grolleau, 1999). This enables off-line study of non synchronous task systems.

The paper is organized as follows. In section 2, the Petri net model used in order to enumerate the entire set of feasible schedules is presented. In section 3, the method used in order to get optimal schedules is explained.

2. SCHEDULING WITH PETRI NETS

2.1 *Task model*

A real-time application is designed as a set of mostly periodic interacting tasks, whose temporal characteristics are fixed. 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. The temporal model mostly used in real-time scheduling theory is the model of (Liu and Layland, 1973) (see fig. 1) where each task τ_i is characterized by four parameters:

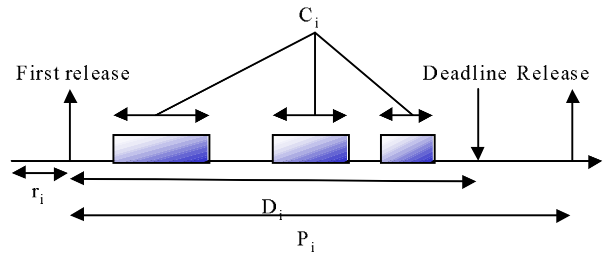

Fig. 1. Temporal parameters of a periodic real-time task $\tau_i\langle r_i, C_i, D_i, P_i\rangle$

- r_i first release time of τ_i
- C_i run-time of τ_i
- D_i deadline of τ_i
- P_i release period of τ_i

A task is then denoted $\tau_i\langle r_i, C_i, D_i, P_i\rangle$. Several values are used to characterize the whole task system, including the major cycle $P = lcm_{i=1..n}(P_i)$. Once the latest release date $r = max_{i=1..n}(r_i)$ is reached, the set of the local clocks of the tasks behaves cyclically upon the major cycle. The utilization factor $U = \sum_{i=1}^{n} \frac{C_i}{P_i}$ is a significant measure of the processor load : since $\frac{C_i}{P_i}$ is the processor part required by τ_i, U is the processor part required by the whole task system. If it is greater than one, the task system is not feasible. If U is less than one, then the processor is cyclically idle : idle slots occur periodically due to the lack of processor request during the life of the system. It can be shown that in a window of size P, $P(1-U)$ idle slots occur. Therefore, the periodic idle slots can be handled by an idle task $\tau_0\langle r_0, P(1-U), P, P\rangle$, which brings the processor load U to one hundred percents. The release date of the idle task is $r_0 = 0$ when all the tasks are synchronous, but will be determined in sec. 2.2.3 when some tasks are not synchronous.

Communications are introduced through asynchronous message passing by means of mailboxes. Those communications induce precedence constraints among the tasks. The usual way to deal with precedence constrained tasks consists in slicing them at the communication points, getting canonical tasks, and to modify the temporal parameters of the new tasks in order to fit the precedence constraints (Blazewicz, 1976). But the inclusion of the precedence constraints within the temporal parameters is achieved differently regarding the chosen policy. Therefore, in the case of an off-line approach, which is not based on a specific priority driven scheduling policy, this way to handle precedence constraints is not achievable. Consequently, the whole precedence constraints of the tasks have to be modeled in an off-line approach. On one hand, this implies a more complicated model than the usual on-line one but on the other hand, the task systems which are handled by our model is less restrictive than the usual one, since when slicing tasks, communications cannot occur within a critical section.

2.2 *Modeling task systems with Petri nets*

This section introduces the model used to schedule real-time task systems. The model is a constrained marking colored Petri net (R. Valk, 1981), under the maximal firing rule (Starke, 1990). This rule inserts time in the behavior of the Petri net since a transition is not fired alone, but a fire involves a maximal set of enabled transitions simultaneously. The expressiveness of Petri nets under the maximal firing rule is equivalent to the expressiveness of T-timed Petri nets under the earliest firing rule.

2.2.1. *The Petri net model* Let $S = \{\tau_0\langle 0, 6, 20, 20\rangle, \tau_1\langle 0, 2, 4, 4\rangle, \tau_2\langle 0, 1, 1, 5\rangle\}$ be a task system where τ_1 and τ_2 share a resource R during their execution. The Petri net modeling S is represented on Fig. 2. The model is decomposed into two parts :

- The temporal component containing :
  A global clock RTC (Real Time clock), which fires whatever the set of fired transitions is, since this transition is always enabled. RTC acts as a timer of the Petri net under the maximal firing rule, since it produces a token in each local clock of the tasks at each time a set of transitions is fired. Therefore in the sequel, the term "time unit" is interchangeable with "a fire under the maximal firing rule"
  The local clocks of the tasks enable to periodically release the tasks. The local clock of a task τ_i is compounded with a place $Time_i$ accumulating the elapsed time since the last release of τ_i ($Time_i$ collects a token produced by RTC each time unit) ; and the transition $Clock_i$, which is enabled when this place holds P_i tokens, fires at the following time unit, producing a token of color a (for activation) in place $Activ_i$
- The tasks body where each task competes for the processor : each transition of the task system is in mutual exclusion with all other transitions of the tasks body thanks to a place *Processor* which is not represented in order to compress the graphical representation of the model. A task is represented classically with one serialized transition per time unit of processor load, but blocks of $d > 3$ time units are compressed into 3 transitions (see task τ_0 on Fig. 2). Communications are

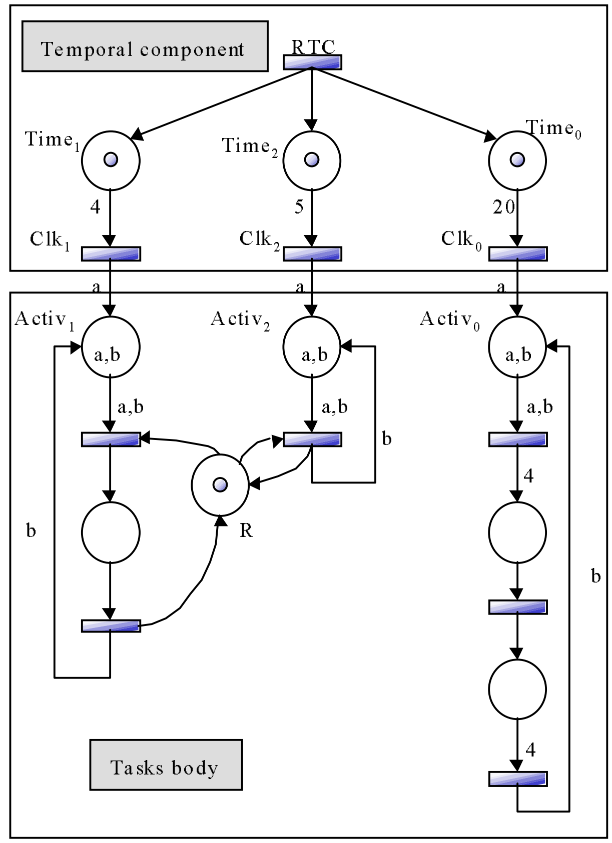

Fig. 2. A Petri net modeling a system of three tasks. Each transition of the temporal specification of the Petri net is labeled by the empty word, and each transition of the tasks body is labeled by the name of the task it belongs to

modeled by mailbox places, and mutual exclusions are modeled by resource places. The first place of each task τ_i, named $Activ_i$, is a colored place which can hold $a - tokens$ and $b - tokens$. An $a - token$ indicates that the task is active (it has been released), and a $b - token$ means that the task has completed its last execution. In order to fit the temporal specifications of the tasks, once an $a - token$ is produced in $Activ_i$, indicating an activation of τ_i, the task should have completed its last execution, this means that $Activ_i$ should contain a $b - token$. If the deadline of τ_i is less than its period ($D_i < P_i$), a $b - token$ should be held by $Activ_i$ as soon as the local clock $Time_i$ holds more than D_i tokens. These two constraints are expressed by two marking constraints for each task

- $Mark(Time_i) > D_i \Rightarrow Mark(Activ_i) = \{b\}$
- $Mark(Time_i) = 1 \Rightarrow Mark(Activ_i) = \{a, b\}$

2.2.2. *Initial state and language* We complete the definition of the model by the description of its initial marking. When a task τ_i is released at the beginning ($r_i = 0$) the place $Activ_i$ holds an

$a-token$ because τ_i is active at the beginning plus a $b-token$ in order to fit the marking constraints. The place $Time_i$ contains one token, therefore τ_i is reactivated by the production of a $a-token$ in $Activ_i$ at the date P_i. When a task τ_j is lately released ($r_j > 0$), the place $Activ_j$ contains only a $b-token$ because the task is not released initially. The marking of the local clock $Time_j$ is $P_j - r_j + 1$ in order to release τ_j at the date r_j.

We focus on the language of the Petri net model where all the reached markings meet the marking constraints, and where each word is infinite. This language is called the center of the terminal language. Since a marking meeting the marking constraints corresponds to a state of the task system where no temporal constraint is violated, the center of the terminal language of the Petri net corresponds to feasible schedules of the modeled task system. Recall that the firing rule is the earliest firing rule, therefore the language corresponds only to the whole set of feasible work-conserving[3] sequences. But since an idle task involving the idle slots is always added to the task system, the language of the Petri net computes the whole set of non work-conserving sequences too since the idle slots can be placed when there are other tasks to compute. It is important since when some tasks share resources, the work-conserving sequences are not optimal (Grolleau, 1999).

Therefore, the whole set of feasible schedules is given by the center of the terminal language of the Petri net model. This model can be viewed as a very flexible enumeration method of feasible schedules because we can easily model mailboxes, multi-instance resources, read/write resources, preemptive and non-preemptive parts...

2.2.3. *Depth of the state graph* The feasible schedules are obtained through the construction of the state graph of the Petri net, where each word is infinite. As in practice, we cannot deal with infinite state graph, we focus now on the cyclicity of the schedules in order to bound the depth of the state graph to compute. As an example, see on Fig. 3 the set of feasible schedules (i.e. the state graph) obtained from the simulation of the Petri net given on Fig. 2. The depth of the state graph is the length of the schedules of the modeled task system. If all the tasks are synchronous, then this depth is $P = lcm(P_i)$ since the state of the system (and equivalently the marking of the Petri net) at the date 0 is the same than at the date P. Therefore, in this case, the initial marking is an home marking. If some tasks are non synchronous, (Grolleau, 1999) has shown that all feasible schedules behave cyclically after the

[3] A work-conserving sequence is a sequence where the processor cannot be idle if there is some work to process

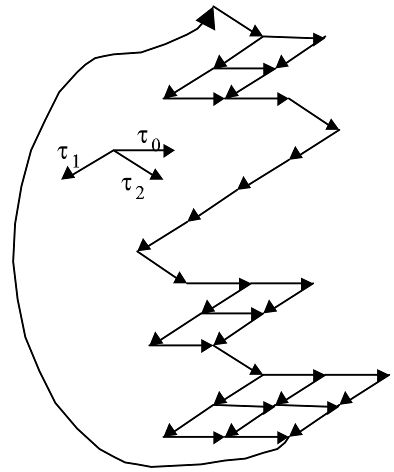

Fig. 3. The state graph of the Petri net given on Fig. 2 and equivalently the set of all feasible schedules for the modeled task system.

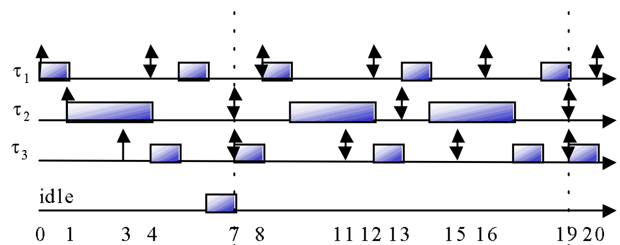

Fig. 4. The schedule of S_2 given by an earliest deadline priority assignment

last acyclic idle slot, with a period P, and that the last acyclic idle slot occurs before the date $r + P$. Therefore, the depth of the state graph to construct is at most $r + 2P$. The concept of acyclic idle slot is the main point of the cyclicity of the schedules. In order to focus on this concept, let's study a task system with a processor utilization $U = 1$. Let $S_2 = \{\tau_1\langle 0,1,4,4\rangle,\ \tau_2\langle 1,3,6,6\rangle,\ \tau_3\langle 3,1,4,4\rangle\}$ with $U_{S_2} = 1$. The Fig. 4 shows that the schedule produced by an earliest deadline priority assignment produces an idle slot at the date 6. This idle slot is an acyclic idle slot since it occurs one time in the infinite schedule (the cyclic ones occur periodically, because there are exactly $P(1 - U_S)$ cyclic idle slots each P units of time). After this acyclic idle slot, the schedule behaves cyclically upon $P = 12$ units of time. The acyclic idle slot is due to the initial processor load, and it does not depend on the scheduling policy: the idle slots occur at the same date whatever the work-conserving scheduling policy is (see Fig. 5). Here is an idea of the proof of cyclicity for independent task systems with a processor utilization $U = 1$. Let t_c be the date of the last acyclic idle slot. Since $U = 1$, the sum of processor requests in the interval $[t_c..t_c + P[$ is P. Since there is exactly on idle slot in the interval $[t_c..t_c + P[$, it remains exactly one unit of time to treat at the date $t_c + P$. Therefore, the processor processes this time unit,

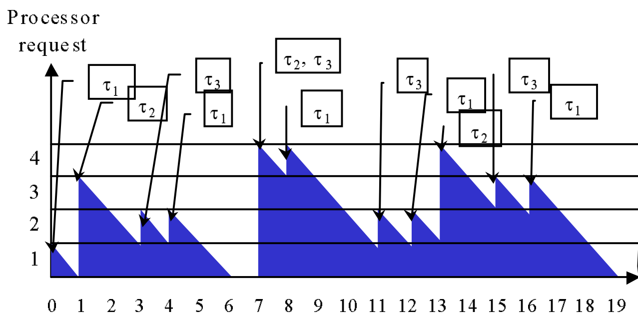

Fig. 5. Processor request diagram for S_2 when a work-conserving scheduling policy is used

and the processor requests at the date $t_c + P + 1$ are given by the releases of the tasks at the date $t_c + P + 1$, which are exactly the same than at the date $t_c + 1$ because $P = lcm_{i=1..n}(P_i)$. So the state of the task system is the same at the date $t_c + P + 1$ than at the date $t_c + 1$. Moreover, we have shown in (Grolleau, 1999) that $t_c < r + P$. This implies that the date of the last acyclic idle slot can be obtained through the construction of a processor request diagram on the interval $[0..r + P[$. The main difficulty when the processor utilization is less than one is that the acyclic idle slots have to be distinguished from the cyclic idle slots which can be handled by an idle task. Since in order to obtain a minimal schedule length, the date t_c has to be as soon as possible, the release date of the idle task is chosen to be $r_0 = t_c + 1$.

As a consequence, the state graph of the Petri net model in the case of non synchronous task system is compounded with a non cyclic part of depth $t_c + 1$ (with $t_c < r + P$), which corresponds to the initial load of the system, and with a steady part between the depth (equivalently the date) $t_c + 1$ and $t_c + P + 1$. There is only one marking at the depth $t_c + 1$, and this marking is an home marking.

3. EXTRACTION OF OPTIMAL SEQUENCES

The state graph obtained by means of simulation of the Petri net model is a diamond[4] graph because each path is a permutation of another path, and the marking following the last acyclic idle slot (or the initial marking in the case of synchronous system) is an home marking.

Therefore, optimal schedules are easy to extract from the state graph: each path (equivalently each schedule) is labeled by the same set of names of tasks, but possibly at different depth (equivalently different dates). Example given, let find in a

[4] A diamond graph is a graph with one source node and one ending node, and each path goes from the source node to the ending node. Moreover, each path is a permutation of another path.

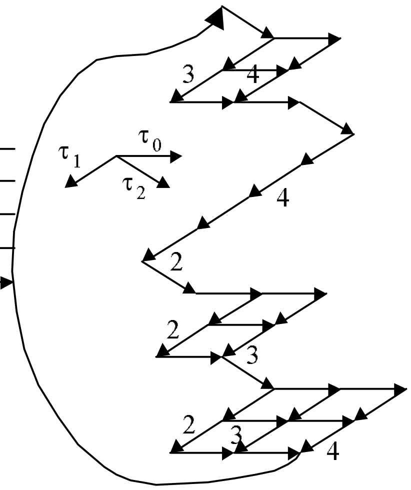

Fig. 6. A weighted state graph for the minimization of the average response times of τ_1. The non-weighted edges are weighted by 0

state graph the optimal schedules for the criteria "minimizing the maximal response time of a task τ_i". A weight is associated to each edge of the graph: each edge corresponding to the completion of τ_i is weighted by the corresponding response time of τ_i, and the other edges are weighted by 0. Since each path has the same number of edges corresponding to the completion of τ_i, the paths minimizing the weights are the paths where the response times of τ_i is minimal.

Let illustrate this technique on the task system $S = \{\tau_0\langle 0,6,20,20\rangle,\ \tau_1\langle 0,2,4,4\rangle,\ \tau_2\langle 0,1,1,5\rangle\}$ whose state graph is given on Fig. 3. The chosen criteria is "minimizing the average response time of τ_1". The Fig. 6 represents the weights associated to the edges. The weight of the nodes is then obtained through a reverse topological algorithm:

- the last node is weighted by 0.
- a node N is weighted by $w(N) = min_{N' \in succ(N)} \{weight(edge(N,N') + w(N'))\}$, so its weight corresponds to the minimal cost of a path from N to the ending node.

This algorithm extracts the set of optimal schedules for the given criteria. It is generalized to the search for optimal schedules of criteria based on the response time of a chosen set of tasks (e.g. response time, reaction rate[5], lateness[6],...).

4. CASE STUDY

Consider a task system dedicated to the control of a mine pump : a mine has to be irrigated, the level of water must lay between a low and a high level, infiltration irrigates it in a natural way, and the task system has to ensure that the

[5] The reaction rate is $\frac{\text{response time}}{D_i}$

[6] The lateness is $D_i - $response time

high level is never exceeded (see (J. Mathai, 1996) for more details). When the water level becomes too high, a pump is triggered until a lower level is reached. Simultaneously, the methane level has to be controlled in order to trigger an alarm when a high level of methane is reached, and to disable the pump if a dangerous level of methane is reached. The entire process is displayed on a control terminal. The task system is implemented with 6 interacting tasks and two highly used shared resources (the control terminal and a buffer shared by acquiring tasks and displaying tasks) which exclude the use of an on-line scheduling approach due to the enlargement of the duration of the tasks using the shared resources in order to avoid the priority inversion problems. Studying the tasks, we get the following task system, where durations are given in milliseconds. CT stands for " uses the resource control terminal " and SB stands for " uses the shared buffer " :

Task	r_i	C_i	D_i	T_i	*CT*	*SB*	*Precedes*
WaterLevel	0	10	100	100	*no*	*yes*	*Control*
MethanLevel	0	10	100	100	*no*	*yes*	*Control*
Control	20	15	100	100	*no*	*no*	*Pump* *Alarm*
Display	10	70	500	500	*yes*	*yes*	
Alarm	0	20	100	100	*yes*	*no*	
Pump	40	12	100	100	*no*	*no*	

Using our tool PeNSMARTS (for Petri Net Scheduling, Modeling and Analysis of Real-Time Systems) on this system, we get a graph of all feasible schedules containing 28649 nodes in less than twenty seconds on a PENTIUM. Using the method described in section 3, we obtain 512 sequences optimizing the average response time of the tasks.

5. CONCLUSION

We propose a method of exhaustive computation of real-time task schedules based on a Petri net model. Given a task system, the language of its associated Petri net is exactly its whole set of feasible schedules. Once the language of the Petri net is stored in the reachability graph, a shortest-path based algorithm allows the extraction of optimal schedules. This algorithm uses the home marking property of the graph. This method was initially achieved on synchronous task systems, and we have shown that it is extendible without modification to asynchronous task systems. This method is the only off-line method, in our knowledge, able to schedule asynchronous task systems, and to extract optimal schedules for several criteria.

6. REFERENCES

Baker, T.P. (1991). Stack-based scheduling of real-time processes. *The Journal of Real-Time Systems* **3**, 67–99.

Blazewicz, J. (1976). Deadline scheduling of tasks - a survey. *Foundations of Control Engineering* **1**(4), 203–216.

Bratley, P., M. Florian and P. Robillard (1973). On scheduling with earliest start and due date constraints with applications to computing bounds for the (n/m/g/fmax) problem. *Naval Research Logistic Quarterly* **20**, 57–67.

Chen, M. and K. Lin (1990). Dynamic priority ceilings: A control protocol for real-time systems. *Real-Time Systems* **2**(4), 325–346.

Choquet-Geniet, A., D. Geniet and F. Cottet (1996). Exhaustive computation of the scheduled task execution sequences of a real-time application. In: *Proc. 4th International Symposium on Formal Techniques in Real Time and Fault Tolerant Systems, Uppsala, Sweden.*

Grolleau, E. (1999). Ordonnancement Temps Réel Hors-Ligne Optimal à l'Aide de Réseaux de Petri en Environnement Monoprocesseur et Multiprocesseur. Optimal Off-Line Scheduling of Real-Time Systems by Use of Petri Nets on an Uniprocessor and a Multiprocessor Architecture. PhD thesis. LISI-ENSMA,Université de Poitiers.

J. Mathai, et al. (1996). *Real-time systems, specification, verification and analysis.* International Series in Computer Science. Prentice Hall.

Leung, J.Y.T. and M.L. Merrill (1980). A note on preemptive scheduling of periodic real-time tasks. *Information Processing Letters* **11**(3), 115–118.

Liu, C.L. and J.W. Layland (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM* **20**(1), 46–61.

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

R. Valk, G. Vidal-Naquet (1981). Petri nets and regular languages. *Journal of Computer and System Sciences.*

Stankovic, J.A., M. Spuri, M. Di Natale and G. Buttazzo (1995). Implications of classical scheduling results for real-time systems. *IEEE Computer* **28**(6), 1–24.

Starke, P.H. (1990). Some properties of timed nets under the earliest firing rule. *Lecture Notes in Computer Science* **424**, 418–432.

Xu, J. and L.P. Parnas (1990). Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on Software Engineering* **16**(3), 360–369.