MSR'99, Modélisation des systèmes réactifs, Cachan, pp. 17-26 , Ed. Hermes, 24-25/03/1999

Modélisation de systèmes temps réel par réseaux de Petri autonomes en vue de leur analyse hors ligne

Emmanuel Grolleau, Annie Choquet-Geniet, Francis Cottet

LISI/ENSMA
Téléport 2 - BP 109
96960 Futuroscope Cedex
grolleau,ageniet,cottet@ensma.fr

RÉSUMÉ. *Nous présentons l'extension d'une méthodologie de détermination de toutes les séquences d'ordonnancement des systèmes de tâches temps réel. La méthode, jusqu'alors appliquée aux systèmes de tâches étant activées à la même date, est étendue aux systèmes de tâches dont certaines peuvent être initialement retardées. Elle se base sur une représentation du système de tâches par des réseaux de Petri colorés avec ensemble terminal fonctionnant avec la règle de tir maximal.*

ABSTRACT. *We present an extension of a methodology used to determinate the whole schedule set of real-time systems. This methodology which was ever applied to task systems with the same first release date is generalized to systems where some tasks have a non zero first release date. Colored Petri net running under the maximal firing strategy is used to generate the schedules.*

MOTS-CLÉS : *temps réel, ordonnancement, réseaux de Petri.*

KEY WORDS: *real-time, scheduling, Petri nets.*

1. Introduction

Un système temps réel se distingue des autres systèmes informatiques par le fait qu'il est soumis à des contraintes temporelles, dues à la criticité de certaines actions qui lui permettent d'interagir avec l'environnement d'un procédé à contrôler [STA 88]. Il peut être modélisé par un système de tâches périodiques [LIU 73], qui peuvent communiquer et partager des ressources comme dans la plupart des systèmes informatiques.

Valider un système temps réel consiste en une validation algorithmique et temporelle des tâches. Nous nous intéressons à l'aspect temporel de la validation des systèmes temps réel, c'est à dire à l'ordonnancement de tâches périodiques.

Dans le cas de systèmes de tâches indépendantes, il existe des algorithmes d'ordonnancement de complexité polynomiale basés sur des priorités allouées aux tâches, dont la validation temporelle peut être obtenue à l'aide de techniques

analytiques ne nécessitant pas la construction de séquences d'ordonnancement [LEU 80][LIU 73][STA 95]. Ces méthodes ont été étendues aux systèmes de tâches soumises à des contraintes de précédence (i.e. tâches communicantes) [CHE 90]. Cependant, dès lors que les tâches partagent des ressources critiques, le problème de l'ordonnancement est NP-difficile [MOK 83]. En effet, en dehors des risques d'interblocages dus aux ressources, l'un des phénomènes les plus caractéristiques engendré par l'adjonction de ressources dans un système est l'inversion de priorité, le temps perdu par une tâche à cause de l'inversion de priorité n'étant, dans le cas général, pas borné. Pour palier à ce problème, les algorithmes classiques d'ordonnancement ont été enrichis de protocoles de gestion de ressources, tels le protocole à priorité héritée [SHA 90], le protocole à priorité plafond [CHEN 90][SHA 90][SPU 94] ou le protocole de gestion des ressources par piles [BAK 91]. La propriété de tels protocoles est de limiter, voire de borner la durée des blocages dus aux attentes de ressources.

Ces approches nécessitent une simulation hors ligne de l'ordonnancement. Cependant, si l'ordonnancement construit doit être implanté en ligne (i.e. par ordonnanceur, qui implémente une politique d'ordonnancement, et non par séquenceur, qui suit une séquence prédéfinie), la durée des sections critiques des tâches doit être augmentée de la durée de blocage de celles-ci, qui dépend de l'algorithme d'ordonnancement et du protocole de gestion des ressources. Après cette intégration, si certaines sections critiques sont relativement longues, la charge théorique du système peut dépasser la capacité de traitement du processeur, ce qui rend le système non ordonnançable.

Une approche complémentaire de l'ordonnancement en ligne est donc requise, notamment pour les systèmes de tâches fortement interagissantes. En effet, si une séquence construite hors ligne, sans prise en compte de la durée de blocage des tâches lors de leur entrée en section critique, est valide, alors son implantation via un séquenceur respecte les contraintes temporelles. Différentes approches d'ordonnancement hors ligne ont été étudiées dans la littérature : algorithmes génétiques, recuit simulé [BEA 98], logique temporelle [ALU 92][CLA 86], réseaux de Petri temporels [BER 91][JUA 97].

Notre approche [CHO 96][GROc 97] se base sur les réseaux de Petri autonomes, dont la puissance d'expression est augmentée à l'aide de contraintes terminales et du mode de fonctionnement sous la règle de tir maximal [STA 90]. Elle permet l'extraction de séquences d'ordonnancement optimales au vu de certains critères, après construction d'un ensemble exhaustif de toutes les séquences possibles. Bien sûr, étant donné la complexité temporelle et spatiale du problème, des heuristiques ont été mises en oeuvre.

Dans la section 2, le modèle utilisé est présenté. Dans la section 3, il est montré que le langage du réseau de Petri est exactement l'ensemble de toutes les séquences d'ordonnancement possibles dans le cas des tâches synchrones au démarrage. En section 4, le modèle est étendu au cas des systèmes de tâches non synchrones au démarrage.

2. Modélisation des systèmes de tâches par réseau de Petri

Les tâches considérées sont périodiques. Chaque tâche τ_i est caractérisée temporellement par les paramètres suivants [LIU 73] :

– r_i sa date de première activation (date de réveil).
– C_i la durée nécessaire à chacune de ses exécutions (charge).
– R_i son délai critique, temps alloué à τ_i pour s'exécuter après chacune de ses activations.
– T_i sa période d'activation.

Une tâche est notée $\tau_i=<r_i,C_i,R_i,T_i>$. Chaque tâche est munie d'une horloge locale démarrant à l'instant r_i et de période T_i. La métapériode du système $H=PPCM_{i=1..n}(T_i)$ donne la période au bout de laquelle l'ensemble des horloges locales des tâches se trouve dans le même état. La date de réveil la plus tardive est notée $r= max_{i=1..n}(r_i)$.

On définit la charge U d'un système de n tâches temps réel par : $U=\sum_{i=1}^{n}\frac{C_i}{T_i}$

Sachant que C_i/T_i représente le taux d'utilisation requis par la tâche τ_i lors de chacune de ses activations, U représente le taux d'utilisation requis par l'ensemble des tâches. Un système de tâches n'est pas ordonnançable sur un processeur de même puissance que celui qui a servi à calculer les charges C_i si sa charge U est supérieure à 1. Par contre, si elle est inférieure à 1, alors le processeur ne sera pas utilisé à chaque instant par les tâches du système. On peut montrer qu'à chaque métapériode, le processeur ne fera rien (on parle de temps creux cycliques) pendant $P(1-U)$ unités de temps. L'adjonction d'une tâche oisive $\tau_0=<0,P(1-U),P,P>$ à tout système de tâche permet d'occuper ces temps creux.

2.1. *Modélisation*

Dans ce paragraphe nous présentons la modélisation de systèmes de tâches temps réel à l'aide des réseaux de Petri colorés avec marquages terminaux [VAL 81] [CAR 84] proposée dans [CHO 96]. Pour une définition des réseaux de Petri, le lecteur peut se reporter à [CHO 93].

Le modèle présenté permet d'obtenir toutes les séquences d'ordonnancement possibles d'un système de tâches modélisé. Le réseau de Petri n'est pas temporisé, et la notion de temps n'y est introduite que par sa structure et par son fonctionnement sous la règle de tir maximal. Ce mode de fonctionnement se différencie du fonctionnement classique des réseaux de Petri par le fait que les transitions ne sont pas tirées une à une mais par ensemble maximal de transition. A chaque tir, un ensemble maximal de transitions est enclenché (i.e. consommation des jetons nécessaires, mais pas production), puis, lorsque le marquage, ainsi modifié, ne permet plus aucun enclenchement de transition, chaque transition enclenchée est déclenchée (i.e. production des jetons). L'enclenchement et le déclenchement de l'ensemble maximal de transitions sont simultanés.

Nous donnons sur la figure 1 la modélisation d'un système de tâches temps réel :

$S=\{\tau_1=<0,2,4,4>, \tau_2=<0,1,1,5>, \tau_0=<0,6,20,20>\}$ où τ_1 et τ_2 partagent la ressource R pendant toute leur exécution.

Le modèle se décompose en deux parties:

— La structure temporelle qui contient :

– L'horloge globale notée RTC (*Real Time Clock*). Etant donné le fonctionnement sous la règle de tir maximal du réseau de Petri, cette transition est tirée à chaque tir de transitions. Elle va ainsi temporiser le comportement du réseau en produisant à chaque tir un jeton dans chaque horloge locale. Dans la suite on associe « unité de temps » à « tir d'un ensemble maximal de transitions ».

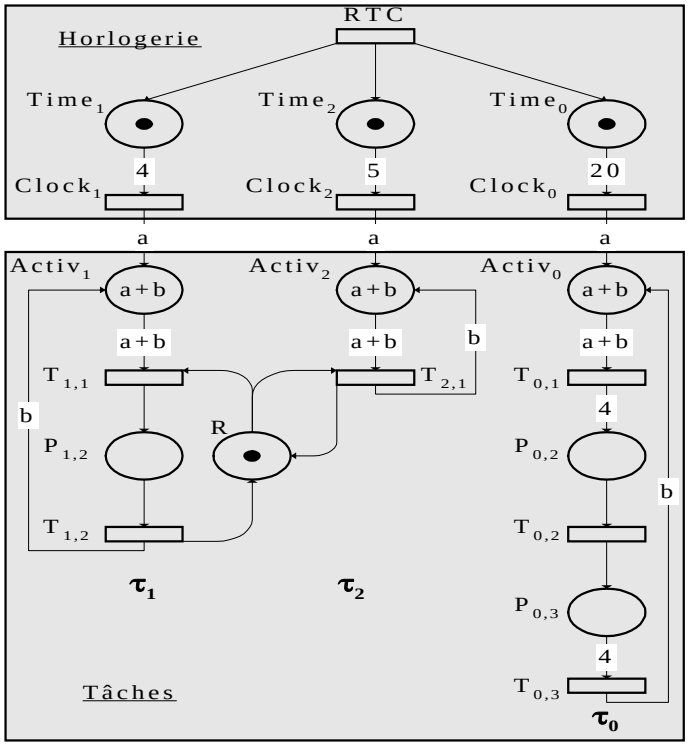

Figure 1. *Modélisation d'un système de 3 tâches avec une ressource critique.*

– Les horloges locales des tâches permettent de réactiver les tâches à chacune de leurs périodes. L'horloge locale d'une tâche τ_i est composée d'une place accumulatrice de temps $Time_i$, qui reçoit à chaque unité de temps un jeton de l'horloge globale RTC. Lorsqu'elle contient T_i jetons (à chacune de ses périodes), elle permet à la transition Clk_i de tirer un jeton de couleur *a* (pour activation) dans la première place de la partie système de tâches de la tâche τ_i.

— Le système de tâches périodiques, mises en parallèle, est représenté de manière classique : un ensemble de C_i transitions mises en série pour chaque tâche, des communications entre elles par place boîte aux lettres, des exclusions mutuelles par places ressource. Une séquence d'instructions indépendantes de durée d≥3 est modélisée par 3 transitions de façon équivalente à la mise en série de d transitions (voir tâche τ_0 sur la figure 1). La première place de chaque tâche τ_i, nommée $Activ_i$, peut contenir deux couleurs de jetons : *a* pour tâche activée, et *b* pour tâche qui a

fini son exécution lors de sa dernière activation. La notation $a+b$ signifie « un jeton de couleur a et un jeton de couleur b ». Lorsqu'une tâche τ_i reçoit un jeton a, elle est réactivée. A ce moment-là, elle doit avoir fini son exécution lors de sa dernière activation, ce qui signifie qu'elle possède (ou reçoit simultanément) un jeton b. Si la tâche n'est pas à échéance sur requête (i.e. $R_i \neq T_i$), lorsque l'horloge locale de la tâche dépasse son délai critique R_i, la tâche doit avoir terminé son exécution, ce qui s'écrit $M(Time_i)>R_i \Rightarrow M(Activ_i)\geq b$ où M est la fonction de marquage. Si la tâche est à échéance sur requête, T_i-1 unités de temps après (ré)activation de τ_i, $M(Time_i)=T_i$, et $M(Activ_i)\leq b$. Une unité de temps plus tard, $M(Time_i)=1$ et $Activ_i$ doit contenir un jeton a et un jeton b. Puisque $M(Time_i)=1$ uniquement lorsque la tâche τ_i vient d'être (ré)activée, il est nécessaire, afin que la tâche respecte son échéance, que $M(Time_i)=1 \Rightarrow M(Activ_i)=a+b$. Les échéances des tâches, à échéance sur requête ou non, nécessitent donc que

$$(M(Time_i)>R_i \Rightarrow M(Activ_i)=b) \qquad [1_i]$$

et

$$(M(Time_i)=1 \Rightarrow M(Activ_i)=a+b) \qquad [2_i]$$

pour un marquage donné M.

Ces deux contraintes, définies pour chaque tâche $\tau_{i,i=0..n}$, garantissent les contraintes temps réel des tâches. Un marquage du réseau de Petri n'est valide que si il respecte les contraintes $[1_i]$ et $[2_i]$ pour toute tâche τ_i du système. Cette notion de validité des marquages se ramène à des contraintes terminales sur le réseau de Petri. L'ensemble des comportements valides (langage terminal) du réseau de Petri est donné par l'ensemble de ses comportements pour lesquels seuls des marquages terminaux (i.e. respectant les contraintes) sont atteints.

Les transitions du système de tâches sont mises en exclusion mutuelle à l'aide d'une place « ressource processeur » qui n'est pas représentée graphiquement pour améliorer la lisibilité. Cette place impose qu'à chaque tir, une seule tâche peut progresser.

Chaque transition d'horlogerie est étiquetée par le mot vide, alors que celles de la partie système de tâches sont étiquetées par le nom des tâches correspondantes. Nous montrons dans les paragraphes suivants que le langage terminal étiqueté du réseau de Petri fonctionnant avec la règle de tir maximal est un ensemble de séquences d'ordonnancement valides du système de tâches modélisé.

2.2. *Tâches synchrones au démarrage*

Lorsque les tâches du système modélisé sont synchrones au démarrage (i.e. r=0), le marquage initial M_0 est le suivant :

i) $M_0(Activ_i)=a+b$ pour toute tâche τ_i : chaque tâche est activée.

ii) $M_0(Time_i)=1$ pour toute tâche τ_i. En effet, puisqu'à chaque unité de temps, un jeton est produit dans $Time_i$ par RTC, à l'instant T_i-1, cette place contient T_i jetons, la transition d'activation de τ_i, $Clock_i$, est ainsi prête à tirer à l'instant T_i. De pa la règle de tir maximal, $Clock_i$ est tirée à l'instant T_i, alors qu'un jeton est produit par RTC dans $Time_i$. Si les contraintes terminales sont respectées, la place $Activ_i$ contient alors $a+b$.

iii) Les places ressources contiennent un nombre de jetons correspondant au nombre de ressources correspondantes disponibles au démarrage du système.

iv)Les places boîtes aux lettres sont vides.
v)Toutes les autres places (i.e. places modélisant le système de tâches) sont vides.

Les points *i* et *ii* montrent que les places horloges locales, ainsi que les places d'activation, se trouvent dans le même état qu'à l'état initial à chacune de leurs périodes. Quant aux places, autres que les places $Activ_i$, composant le système de tâches, on peut montrer, de par leur structure sérielle et leur marquage initial vide, qu'elles sont toutes vides à chaque période des tâches associées. Chaque tâche τ_i se trouve donc dans le même état toutes les T_i unités de temps, donc le système est dans le même état à l'instant 0 qu'une métapériode plus tard.

L'étude de l'état des places ressources et boîtes aux lettres permet de conclure sur la cyclicité des comportements du réseau sur la métapériode du système de tâches à condition que les contraintes suivantes soient respectées :

– Chaque ressource est rendue après utilisation.
– Tout message émis est reçu. Cela signifie que les périodes des tâches communicantes doivent être corrélées par le nombre de message qu'elles échangent.

Chaque transition du système de tâches étant étiquetée, le langage terminal étiqueté est exactement l'ensemble de toutes les séquences d'ordonnancement possibles si et seulement si à chaque instant une transition et une seule du système de tâches est tirée. La place processeur garantie qu'à chaque instant, au plus une tâche progresse. Afin de montrer qu'à chaque instant, une transition du système de tâches est tirée, rappelons que les systèmes de tâches ont une charge U=1. Cela implique que dans l'intervalle de temps [0..H-1], H unité de temps devront être consacrées par le processeur aux tâches. Or à l'instant H, toutes les tâches sont réactivées, et doivent donc avoir terminé leurs exécutions. Pour respecter les contraintes temporelles (i.e. les contraintes terminales), une tâche doit donc progresser à chaque unité de temps.

Le langage terminal étiqueté du réseau de Petri est donc bien l'ensemble de toutes les séquences d'ordonnancement valides du système modélisé dans le cas où les tâches sont synchrones au démarrage. De plus, il est obtenu par la construction du graphe des marquages dont la profondeur peut être bornée par H, de par la cyclicité des comportements valides du réseau. Ainsi, si S est une séquence d'ordonnancement extraite du graphe des marquages, de longueur H, alors S^* est une séquence d'ordonnancement infiniment valide du système modélisé.

2.3. *Tâches non synchrones au démarrage*

Dans le cas des tâches non synchrones au démarrage, deux problèmes majeurs se posent. Le premier concerne les temps creux, qui ne peuvent plus tous être pris en compte par une tâche oisive. En effet, si les tâches d'un système ne sont pas synchrones au démarrage, même lorsque la charge est maximale (U=1), il est possible de glisser des temps creux dans les séquences ordonnancements, dus aux perturbations engendrées par la montée en charge du système. Ces temps creux sont qualifiés de temps creux acycliques, car le nombre de leurs occurrences dans un ordonnancement de taille infinie est borné. Pour que le langage étiqueté du réseau de Petri corresponde à l'ensemble des séquences valides, il faut adjoindre au réseau une tâche particulière dont les transitions sont étiquetées par « temps creux

acyclique ». Nous avons montré que le nombre de temps creux acycliques est borné par *r* [GROb 97][GROa 98], la tâche acyclique consiste donc en une unique transition $T_{acyclique}$, utilisant la ressource processeur, et consommant à chaque tir un jeton d'une place « nombre de temps creux acycliques » contenant initialement *r* jetons.

Le second problème concerne la profondeur du graphe des marquages à construire, ainsi que la cyclicité des séquences produites. En effet, alors que le régime permanent est atteint dès l'instant initial (activation de toutes les tâches) dans le cas où *r*=0, le cas *r*>0 implique une phase de montée en charge, suivie du régime permanent. Nous avons montré dans [GROb 97][GROa 98] que la montée en charge se terminait avec l'occurrence du dernier temps creux acyclique lorsque l'ordonnancement était construit au plus tôt (i.e. des temps creux acycliques n'interviennent que si aucune tâche ne peut progresser), et que cette occurrence avait lieu avant la date *r+H*. A partir de cette date, toute séquence d'ordonnancement valide est cyclique de période *H*. Les chemins du graphe des marquages ont donc un comportement cyclique de période *H* à partir du dernier temps creux acyclique. Il convient de borner la date d'occurrence du dernier temps creux acyclique. De par la périodicité des tâches, tous les régimes permanents de séquences d'ordonnancement sont obtenus si les temps creux acycliques sont contraints d'intervenir avant la date *r+H*. La tâche particulière acyclique est couplée à une horloge locale interdisant à partir de la date *r+H* le franchissement de la transition $T_{acyclique}$ (voir figure 2).

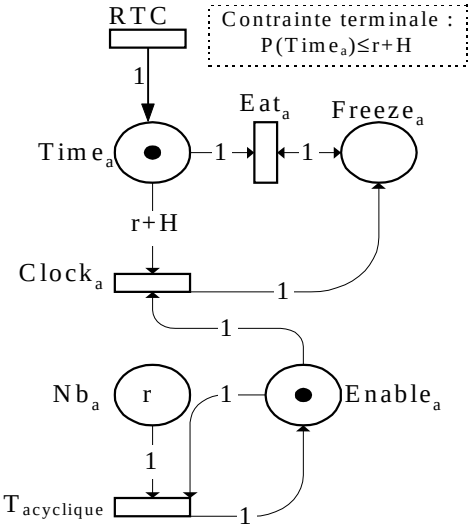

Figure 2. *Adjonction de la tâche acyclique et de son système d'horlogerie au réseau de Petri initial.*

Avant l'instant *r+H*, la transition $T_{acyclique}$ est validée. A l'instant *r+H*, la transition $Clock_a$ est tirée. A partir de cet instant, la transition Eat_a, étiquetée par le mot vide, est activée et consomme les jetons de $Time_a$ au fur et à mesure qu'ils sont produits par RTC.

Le graphe des marquages a une profondeur de *r+2H*. Les feuilles du graphe construit sont identiques aux noeuds de hauteur *r+H*.

Le langage ne contient pas exactement toutes les séquence d'ordonnancement possible. Nous ne représentons pas les séquences dans lesquelles on attend un temps arbitrairement long avant d'introduire les temps creux acycliques. Ce choix provient du fait que l'intégration de ces séquences n'induit pas un potentiel d'ordonnançabilité supérieur.

3. Conclusion

La modélisation d'un système de tâches temps réel par réseaux de Petri autonomes a été étendue au cas des systèmes de tâches non synchrones au démarrage. La modification du modèle se traduit par l'adjonction au réseau d'une tâche particulière, nommée tâche acyclique, permettant de prendre en compte l'occurrence de temps creux acycliques dus à la montée en charge du système. L'horloge locale qui lui est associée permet de borner la taille du graphe des marquages obtenu.

Le langage terminal étiqueté du modèle est donc l'ensemble de toutes les séquences d'ordonnancement du système de tâches dans le cas où r=0, et il contient toutes les séquences d'ordonnancement dont le régime permanent est atteint avant la date $r+H$ lorsque $r \neq 0$.

Cette méthodologie a été implémentée dans un atelier d'aide au choix de séquences d'ordonnancement [GROa 97] [GROb 98]. Après construction du graphe des marquages, il est possible d'extraire en un temps linéaire de la taille du graphe des séquences optimales suivant certains critères, tels que la minimisation du temps de réponse de certaines tâches, la minimisation de leur taux de réaction, etc. [GROc 97]. De plus certaines heuristiques ont été mises en oeuvre (contraintes de successeurs, contraintes absolues) pour réduire la taille du graphe des marquages.

L'extension du modèle et de la méthodologie aux systèmes multiprocesseur et répartis est à l'étude.

4. Bibliographie

[ALU 92] ALUR R., HENZINGER T.A. « Logics and models of real-time : a survey », *Lecture Notes in Computer Science, Real-Time : Theory in Practice*, Springer-Verlag, 1992.

[BAK 91] BAKER T.P., « Stack-based scheduling of realtime processes », *Journal of Real-Time Systems*, n°3, 1991.

[BEA 98] BEAUVAIS J.P., DÉPLANCHE A.M., « Affectation de tâches dans un système temps réel réparti », *Technique et Science Informatiques*, vol. 17, n°1, 1998, p. 87-106, Hermès, Paris.

[BER 91] BERTHOMIEU B., DIAZ M., « Modeling and verification of time dependent systems using time Petri nets », *IEEE Transactions on Software Engineering*, vol. 17, n°3, p. 259-273, 1991.

[CAR 84] CARTENZEN H., VALK R., « Infinite behaviour and fairness in Petri nets », *Advances in Petri Nets*, n°188, p. 83-100, 1984.

[CHEN 90] CHEN M., LIN K., « Dynamic priority ceilings : a control protocol for real-time systems », *Journal of Real-Time Systems*, n°2, 1990.

[CHE 90] CHETTO H., SILLY M., BOUCHENTOUF T., « Dynamic scheduling of real-time tasks under precedence constraints », *Journal of Real-Time Systems*, n°2, 1990.

[CHO 93] CHOQUET-GENIET A., VIDAL-NAQUET G., *Réseaux de Petri et systèmes parallèles*, Editions Armand Colin, 1993.

[CHO 96] CHOQUET–GENIET A., GENIET D., COTTET F., « Exhaustive computation of the scheduled task execution sequences of a real–time application », *actes du 4th International Symposium on Formal Techniques in Real Time and Fault Tolerant Systems*, Uppsala, Suède, 11–13 septembre 1996.

[CLA 86] CLARKE E.M., EMERSON E.A., SISTLA A.P., « Automatic verification of finite-state concurrent systems using temporal logic specifications », *ACM Transactions on Programming Languages and Systems*, vol. 2, n°8, p. 244-263, 1986.

[GROa 97] GROLLEAU E., « Projet PeNSMARTS, Petri Net Scheduling, Modeling & Analyses of real-time Systems », *actes de l'école d'été temps réel, ETR'97*, p. 235, Poitiers-Futuroscope, 22-26 septempre 1997.

[GROb 97] GROLLEAU E., CHOQUET-GENIET A., COTTET F., « Cyclicité des ordonnancements au plus tôt des systèmes de tâches temps réel à contraintes strictes », rapport de recherche n°97007, 1997, LISI/ENSMA.

[GROc 97] GROLLEAU E., CHOQUET-GENIET A., COTTET F., « Ordonnancement optimal de systèmes de tâches temps réel à l'aide de réseaux de Petri », *actes de conférence Automatique, Génie Informatique, Signal, AGIS*, Angers, 9-11 décembre 1997.

[GROa 98] GROLLEAU E., CHOQUET-GENIET A., COTTET F., « Cyclicité des ordonnancements au plus tôt des systèmes de tâches temps réel », *actes de conférence Rencontres Francophones du Parallélisme, RenPar'10*, Strasbourg, 9-12 juin 1998.

[GROb 98] GROLLEAU E., CHOQUET-GENIET A., COTTET F., « Validation de systèmes temps réel à l'aide de réseaux de Petri », *actes de conférence Approches Formelles dans l'Assistance au Développement de Logiciels, AFADL'98*, Futuroscope, 30 sept- 1er oct 1998.

[JUA 97] JUANOLE G., « Réseaux de Petri temporisés et/ou stochastiques », *actes de l'école d'été temps réel, ETR'97*, p. 174-180, Poitiers-Futuroscope, 22-26 septempre 1997.

[LEU 80] LEUNG J.Y.T., MERILL M.L., « A note on preemptive scheduling of periodic real–time tasks », *Information Processing Letters*, n°11, vol. 3, p. 115–118, 1980.

[LIU 73] LIU C.L., LAYLAND J.W., « Scheduling algorithms for multiprogramming in a hard real–time environment », *journal of the ACM*, n°20, vol. 1, p. 46–61, 1973.

[MOK 83] MOK A.K., « Fundamental design problems of distributed systems for the hard real-time environment », Ph. D. Thesis, Department of Electrical Engineering and Computer Science, Massachussets Institute of Technologie, Cambridge, Massachussets, May 1983.

[SHA 90] SHA L., RAJKUMAR R., LEHOCZKY J.P., « Priority inheritance protocols : an approach to real–time synchronisation », *IEEE Transactions on Computers*, n°9, vol. 39, p. 1175–1185, Septembre 1990.

[SPU 94] SPURI M., STANKOVIC J.A., « How to integrate precedence constraints and shared resources in real-time scheduling », *IEEE Transactions on Computers*, n°12, vol. 43, p. 1407-1412, 1994.

[STA 88] STANKOVIC J.A., « Misconception about real–time computing », *IEEE Computer Magazine*, n°21, vol. 10, p. 0–19, 1988.

[STA 95] STANKOVIC J.A., SPURI M., DI NATALE M., BUTTAZZO G., « Implications of classical scheduling results for real-time systems », *IEEE Compute*, n°6, vol.28, p. 1-25, 1995.

[STA 90] STARKE P.H., « Some properties of timed nets under the earliest firing rule », Lecture Notes in Computer Science, vol. 424, p. 418-432, 1990.

[VAL 81] VALK R., VIDAL-NAQUET G., « Petri nets and regular languages », *Journal of Computer and System Sciences*, n°3, vol. 23, Academic Press, 1981.