Validation temporelle de systèmes de tâches temps-réel strictes à durée variables à l'aide de langages rationnels

Dominique Geniet, Gaëlle Largeteau-Skapin

Validation Temporelle de Systèmes de Tâches Temps-Réel Strictes à Durées Variables à l'Aide de Langages Rationnels[1]

Dominique Geniet* — **Gaëlle Largeteau***

** Laboratoire d'Informatique Scientifique et Industrielle*
Université de Poitiers & E.N.S.M.A.
Téléport 2, 1 av. C. Ader, BP 40109
F-86961 Futuroscope Chasseneuil Cédex
Tel:33+(0) 549 498 062 Fax:33+(0) 549 498 064
mailto:{dgeniet,glargeteau}@ensma.fr

RÉSUMÉ. *Nous montrons, dans ce papier, que les systèmes temps-réel constitués de tâches périodiques et sporadiques interdépendantes à durée d'exécution variables peuvent être validées temporellement à partir d'une méthodologie à base de langages rationnels. Ceci établit la cyclicité des séquences d'ordonnancements pour ces systèmes de tâches.*

ABSTRACT. *We show that real-time systems composed of both periodic and sporadic interdependent tasks which CPU time is not fixed can be validated from a methodology based on rational languages. This property leads to the cyclicity of the schduling sequences for such systems.*

MOTS-CLÉS : *Temps-réel, Automates finis, Langages rationnels, Cyclicité, Validation temporelle, Ordonnançabilité*

KEYWORDS: *Real-time, Finite automata, Rational languages, Cyclicity, Temporal validation, Feasibility*

1. À paraître dans les actes de **Modélisation des Systèmes Réactifs, Toulouse, Septembre 2001, France**

1. Introduction

1.1. *Les systèmes temps-réel*

Par essence même, un système temps-réel est d'une part **réactif** (car devant prendre en compte, à la volée, les informations entrantes qui lui indiquent l'évolution du procédé dans son environnement physique), et d'autre part **concurrent** (car l'ensemble des opérations relatives au pilotage –observations de l'environnement et actes de conduite– doivent être réalisées simultanément). Il est donc naturellement composé d'un ensemble de **tâches** élémentaires, chacune d'elles implémentant une réaction (ou une partie de réaction) que le système devra fournir à une sollicitation extérieure (qui s'exprime sous la forme d'un **événement**).

Certaines informations entrant dans le système suivent des flots réguliers : ce sont, généralement, des données transmises par des capteurs périodiques. Les tâches de lecture et de traitement de ces informations doivent donc être synchronisées avec les capteurs en question. Ce sont des tâches **périodiques**. Si le système comporte n tâches périodiques, nous les appelons ici $(\tau_i)_{i\in[1,n]}$. Dans le modèle classique de tâches temps-réel [LIU 73], la spécification temporelle de τ_i est constituée par la donnée de quatre caractéristiques :

- La **date de première activation** (notée $\mathbf{r_i}$) est l'instant de création de τ_i (l'origine des temps est $\min_{i\in[1,n]}(r_i)$)
- Le **période** (notée $\mathbf{T_i}$) est le délai séparant deux activations successives de τ_i
- Le **délai critique** (noté $\mathbf{D_i}$) est le délai séparant la date d'activation d'une occurrence de τ_i de la date à laquelle l'exécution de cette occurrence doit être terminée (cette date est nommée **échéance** de l'occurrence de τ_i)
- La **durée d'exécution** (notée $\mathbf{C_i}$) est la durée pendant laquelle chaque occurrence de τ_i devra disposer d'un processeur pour pouvoir terminer son exécution

Notons que *date de première activation*, *période* et *délai critique* sont imposés par le procédé à piloter. À ce titre, ce sont des paramètres hérités. À l'opposé, la *durée d'exécution* est un paramètre synthétisé à partir des caractéristiques du processeur cible et du corps de τ_i.

Les signaux d'alarmes, ou les interventions d'un éventuel superviseur, constituent les flots d'informations entrantes non-périodiques. Les tâches de prise en compte de ces informations, activées uniquement lorsque c'est nécessaire, sont appelées tâches **sporadiques** ou **apériodiques**. Une tâche sporadique est caractérisée par la connaissance d'un délai minimum séparant deux de ses occurrences successives. Dans le cas où un tel délai ne peut être garanti, la tâche est dite apériodique. Si le système comporte p tâches sporadiques[1], nous les appelons ici $(\alpha_i)_{i\in[1,p]}$.

1. Nous nous focalisons ici sur les systèmes constitués de tâches périodiques et sporadiques

1.2. *Techniques de validation*

Avant sa mise en exploitation, tout système doit être validé, de façon à garantir que son comportement correspond aux besoins affichés. Ceci est particulièrement vrai pour les applications de contrôle-commande. Pour les applications temps-réel, la validation comporte deux aspects. D'une part la classique validation fonctionnelle, mais également la validation temporelle[2] : il s'agit de garantir que, quelle que soit l'évolution du procédé, le système temps-réel est apte à traiter correctement (c'est à dire *en respectant ses contraintes temporelles*) l'ensemble des signaux susceptibles de se présenter. Le pilotage d'un système temps-réel est classiquement abordé suivant deux approches différentes (voir une synthèse fournie dans [GRO 99]) : l'ordonnancement **en ligne** et l'ordonnancement **hors-ligne**.

1.2.1. *Politiques en ligne*

Le pilotage en ligne consiste à implanter dans la machine cible un ordonnanceur qui s'appuie sur une politique spécifique (basée sur des priorités, où sur les paramètres temporels des tâches, par exemple) : chaque événement (signal ou commutation) voit l'ordonnanceur élire la prochaine tâche en fonction de sa politique. Cette approche a motivé un grand nombre de travaux ([CHO 99], par exemple, pour les systèmes périodiques, [BAK 91] propose des politiques adaptées à la gestion des ressources critiques, on trouvera dans [DEL 99] une synthèse sur l'intégration des sporadiques dans cette approche). Si l'approche en ligne est souple, elle est toutefois limitée par sa non-optimalité[3] dès que les tâches sont interdépendantes (c'est à dire à peu près tout le temps, dans les cas réels !).

1.2.2. *Politiques hors-ligne*

En toute généralité, le problème de décision de l'ordonnançabilité d'un système temps-réel est NP-difficile [BAR 90]. Il est donc illusoire de rechercher une politique en ligne optimale dans le cas général. Cet état de fait a motivé la mise en place de stratégies basées sur l'énumération exhaustive des ordonnancements possibles d'une configuration donnée : les stratégies hors ligne. Ici, l'énumération permet, dès lors que la configuration est ordonnançable, d'exhiber une séquence valide qui, implantée dans un séquenceur, pilotera l'application.

Pour les systèmes périodiques[4], la production d'une telle séquence suppose la connaissance de la durée minimale de simulation. [LEU 80] en fournit une borne, et [GRO 99] généralise ce résultat.

Les méthodes hors-ligne s'appuient sur des modèles de tâches, à base de systèmes à états (automates et réseaux de Petri). Certains sont à temps explicite [ALU 94]

2. Un *bon* résultat rendu hors délai est faux
3. Un algorithme d'ordonnancement est **optimal** si il n'échoue que sur les configurations de tâches qui ne peuvent être ordonnancées (voir [GRO 99] pour une définition formelle de l'optimalité)
4. i.e. uniquement constitués de tâches périodiques

[FLO 85] [RAM 74] (on parle de modèles **temporisés**), d'autres à temps implicite [CHO 96]. Les modèles à temps explicite sont exploités via des techniques de simulation, ceux à temps implicite via des techniques d'analyse. Dans les deux cas, un premier axe d'étude consiste à produire des techniques de modélisation exhaustives (i.e. qui prennent en compte l'ensemble des configurations réelles possibles). Par ailleurs, les complexités temporelles de décision étant exponentielles, un axe majeur des travaux est la mise en œuvre de techniques d'amélioration des temps de calcul (essentiellement par la détection précoce de branches correspondant à des cas d'échec). Les systèmes périodiques interdépendants pour lesquels les tâches ont une durée d'exécution fixée sont pris en compte. Évidemment, ce cas de figure est tout à fait irréaliste, puisque dès que le corps d'exécution d'une tâche intègre un test du type *If…Then…Else…*, on sort du modèle. Dans ce cas, on valide usuellement en pratiquant la technique de la *pire durée d'exécution*. Les matériels jugés aptes à piloter l'application étudiée sont, en général, très fortement sur-évalués. Mais à ce problème d'*inefficacité* s'ajoute un problème de fond, qui invalide totalement l'approche, dans le cas d'interdépendence entre tâches (ressources, communication) : l'instabilité des algorithmes d'ordonnancement [5].

1.2.3. *Validation*

Valider une application temps-réel (i.e. un systême de tâches) consiste à prouver que, quelle que soit l'évolution du procédé qu'elle pilote, elle ne se trouvera jamais en situation de violation de ses contraintes temporelles. Dans le cas où l'objectif est un pilotage en ligne, la validation s'obtient à l'aide de techniques de simulation. Dans le cas hors-ligne, on utilise des techniques de model-checking pour produire une séquence d'ordonnancement en accord avec les contraintes spécifiées.

Les méthodes de validation temporelle basées sur l'approche *pire cas* sont utilisées dans tous les ateliers industriels existants, et supposées valides dans tous les travaux scientifiques que nous connaissons à ce jour. Pourtant, le problème de l'instabilité des algorithmes d'ordonnancement montre que ces approches sont en fait invalides dans la quasi-totalité des cas réels.

Notre propos est ici de combler ce vide en matière de validation temporelle de systèmes critiques. Dans ce papier, nous établissons une technique de décision de la validité temporelle pour des systèmes de tâches tmpes-réel comportant des surées d'exécution variables, et nous étendons le résultat de la cyclicité des ordonnancements valides à de tels systèmes de tâches, en environnement multi-processeur.

---

5. Le problème de l'instabilité s'exprime ainsi : un ordonnancement temporellement valide pour un système $(\tau_i)_{i\in[1,n]}$ de tâches de durées d'exécution $(C_i)_{i\in[1,n]}$ peut être non valide si la durée de l'une des occurrences de l'une des τ_i est strictement inférieure à C_i. On trouvera une description détaillée de ce problème –en général non abordé– et de ses conséquences, dans [CHO 00], pp 73 & 74.

1.3. *Cadre du travail*

Nous avons introduit dans [GEN 00] une technique de modélisation de tâches temps-réel à base d'automates finis, et nous avons montré qu'elle prend en compte les configurations de tâches périodiques interdépendantes à durée d'exécution fixe. Dans [GEN 01], nous prenons en compte les tâches sporadiques (qui peuvent être interdépendantes avec les périodiques), et nous introduisons une méthode à base de séries génératrices, pour décider a priori de l'ordonnançabilité de certaines sporadiques.

Le temps y étant implicite, notre modèle présente, par rapport aux autres approches orientées modèles (automates temporisés [ACE 98], réseaux de Petri [GRO 99]) un certain nombre d'avantages. D'une part, il permet de disposer de processus de décision basés sur les techniques d'analyse des automates finis, dont la puissance, l'efficacité et la modularité sont établies depuis longtemps. Par exemple, le résultat central de [GRO 99], qui établit la cyclicité des ordonnancements en environnement mono processeurs de systèmes de tâches interdépendants, se retrouve, dans notre approche, comme un corollaire de la rationnalité de l'ensemble des comportements valides d'une application temps-réel. La rationnalité de cet ensemble persistant en multi-processeur, nous pouvons d'ores et déjà affirmer que, pour les tâches à durées fixes, la cyclicité des ordonnancements persiste en multi-processeurs. D'autre part, l'aspect comportemental de notre approche permet l'expression de propriétés assez fines, plus difficilement exprimables par d'autres approches, soit parce qu'elles nécessitent des manipulations équationnelles élaborées (approches *temporisées*), soit parce qu'elles imposents des définitions structurelles assez lourdes (approches *réseaux de Petri*, par exemple).

Ici, après une synthèse de la modélisation et une réflexion sur les méthodes de calcul les plus adaptées, nous montrons que la modélisation s'adapte bien aux tâches à durées variables, et nous généralisons le résultat de la cyclicité à ce cas d'étude. Nous sommes donc en mesure de prendre en compte le problème de l'instabilité des algorithmes d'ordonnancement : c'est là un apport central de notre méthode.

Ce résultat montre que ce modèle possède une expressivité suffisante pour l'étude de systèmes temps-réel quelconques **et** réels. Par rapport aux modèles temporisés [RAM 74][ACE 98], il présente l'intérêt d'impliciter le temps, et donc d'utiliser les techniques de model checking les plus performantes qui soient : celles du modèle des *automates finis déterministes*.

2. Modélisation de tâches à contraintes strictes

Nous nous intéressons ici à la validation temporelle d'applications. Notre modèle doit donc être à même de répondre à des questions centrées sur l'aspect opérationnel (respect des échéances, donc), et non sur l'aspect fonctionnel (ce que fait le programme). Dans un premier temps, nous présentons succinctement la modélisation temporelle [GEN 00] pour les tâches à durées fixes. Nous étendons ensuite cette modélisation aux tâches à durées variables.

2.1. *Tâches périodiques à durée d'exécution fixe*

Intéressons nous ici à la tâche[6] τ_i, dont les paramètres temporels sont r_i, C_i, D_i et T_i. À partir de sa date d'activation (un entier de $r_i + T_i\mathbb{N}$), chaque occurrence de τ_i doit impérativement se voir allouer un processeur pendant exactement C_i unités de temps sur sa période. En représentant par le symbole a_i l'activité de τ_i, et par le symbole $\bullet$ l'inactivité de τ_i, tout mot de $a_i^{C_i} \text{ш} \bullet^{D_i - C_i}$ correspond, sur tout intervalle temporel du type $[r_i + k.T_i, r_i + k.T_i + D_i[$, à une configuration d'allocation d'un processeur à τ_i compatible avec ses contraintes temporelles. Cet ensemble est rationnel. Prendre en compte le fait que, sur l'intervalle $[r_i + k.T_i + D_i, r_i + (k+1).T_i[$, τ_i est toujours inactive, revient à suffixer tout mot de $a_i^{C_i} \text{ш} \bullet^{D_i - C_i}$ par $\bullet^{T_i - D_i}$. La concaténation de ces deux langages est le langage $a_i^{C_i} \text{ш} \bullet^{D_i - C_i} \bullet^{T_i - D_i}$, également rationnel. Sur le plan opérationnel, τ_i est la suite $(\tau_{ij})_{j \in \mathbb{N}}$ de ses occurrences. Donc, allouer globalement à τ_i un processeur de manière compatible avec ses contraintes temporelles revient à allouer à chacune des occurrences τ_{ij} un processeur de manière compatible avec ses contraintes temporelles. Chaque mot ω de $a_i^{C_i} \text{ш} \bullet^{D_i - C_i} \bullet^{T_i - D_i}$ est de longueur T_i : il décrit une configuration d'allocation **valide**[7] du processeur à une occurrence de τ_i. Donc, si $(\omega_j)_{j \in \mathbb{N}} \in \left(\left(a_i^{C_i} \text{ш} \bullet^{D_i - C_i}\right) \bullet^{T_i - D_i}\right)^{\mathbb{N}}$, le mot $\omega_0.\omega_1 \ldots \omega_n$ modélise une configuration d'allocation du processeur valide pour n+1 occurrences de τ_i. D'une façon générale, tout mot ω de $\left(\left(a_i^{C_i} \text{ш} \bullet^{D_i - C_i}\right) \bullet^{T_i - D_i}\right)^*$ modélise une configuration d'allocation de processeur valide pour τ_i sur n'importe quel intervalle temporel du type $[r_i + k.T_i, r_i + k.T_i + |\omega|[$, et donc, notamment, sur l'intervalle $[r_i, r_i + |\omega|[$. Étant donné que τ_i est inactive sur l'intervalle $[0, r_i[$, tout mot $\bullet^{r_i}\omega$, où $\omega \in \left(\left(a_i^{C_i} \text{ш} \bullet^{D_i - C_i}\right) \bullet^{T_i - D_i}\right)^*$, modélise une configuration d'allocation de processeur valide pour τ_i sur l'intervalle $[0, r_i + |\omega|[$. ω pouvant de longueur arbitrairement grande (le langage est une étoile de langage rationnel), il modélise les configurations d'allocation valides pour n'importe quelle durée de vie de la tâche. Le langage $\bullet^{r_i}\left(\left(a_i^{C_i} \text{ш} \bullet^{D_i - C_i}\right) \bullet^{T_i - D_i}\right)^*$ est rationnel. Le problème de l'ordonnancement temps-réel consiste, à un instant t donné de la vie de la tâche, à pouvoir prédire ses possibilités d'évolution dans un environnement donné. Bien entendu, le passé de la tâche est connu. Pour ce qui nous intéresse ici, ce passé est simplement l'historique des allocations CPU de τ_i un mot ω de $\{\bullet, a_i\}^*$. Par construction, ω est de la forme $\omega_1.\mu$, où $\omega_1 \in \bullet^{r_i}\left(\left(a_i^{C_i} \text{ш} \bullet^{D_i - C_i}\right) \bullet^{T_i - D_i}\right)^*$ et

$$\exists \nu \in \{\bullet, a_i\}^* \text{ tel que } \mu\nu \in \left(a_i^{C_i} \text{ш} \bullet^{D_i - C_i}\right) \bullet^{T_i - D_i}$$

Donc, ω est un préfixe d'un mot de $\bullet^{r_i}\left(\left(a_i^{C_i} \text{ш} \bullet^{D_i - C_i}\right) \bullet^{T_i - D_i}\right)^*$. Par ailleurs, par construction également, quel que soit l'instant f appartenant au futur $]t, +\infty[$, il existe un mot ω_1 dans $\left(\left(a_i^{C_i} \text{ш} \bullet^{D_i - C_i}\right) \bullet^{T_i - D_i}\right)^*$, et tel que $\omega\mu\nu\omega_1$ soit de longueur

6. Nous ne considérons que le cas des tâches non réentrantes.
7. c'est à dire compatible avec les contraintes temporelles de la tâche

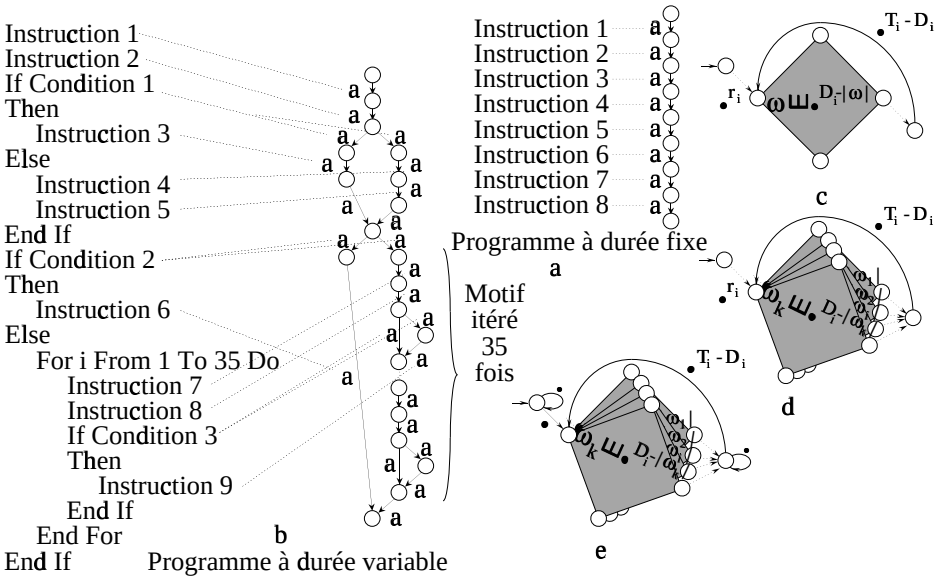


FIG. 1. *Mots associés à une tâche à durée variable*

supérieure à f. À tout instant t, le passé ω de τ_i est donc un mot du centre[8] de $\bullet^{r_i}\left(\left(a_i^{C_i} \text{ш} \bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\right)^*$. Réciproquement, par définition, tout mot du centre est le passé d'une configuration valide d'allocation de processeur à la tâche.

Définition 1 *On appelle* **comportement temporel valide** *de la tâche τ_i tout mot ω appartenant au langage* $Centre\left(\bullet^{r_i}\left(a_i^{C_i} \text{ш} \bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\right)^*$

Un comportement temporel valide de τ_i modélise donc une configuration d'allocation de processeur à τ_i pour laquelle on est en mesure de garantir que τ_i dispose d'au moins une possibilité de continuer son exécution en respectant ses contraintes temporelles. Dans la suite, nous notons $L(\tau_i)$ ou, plus succinctement, L_i, l'ensemble des comportements temporels valides de τ_i. Ce langage est rationnel. .

2.2. *Tâches à durée d'exécution variable*

Dans le cas général, les tâches temps-réel s'appuient sur des programmes dont les comportements sont variables. C'est le cas dès que le programme intègre une instruction conditionnelle ou de boucle. La durée d'exécution de la tâche dépend alors du comportement fonctionnel du programme. Pour intégrer ce genre de tâches dans le modèle temporel, il faut modéliser des durées d'occupation CPU variables, mais également que les durées d'occupation CPU permises pour τ_i correspondent ex-

8. Le centre de L est l'ensemble des préfixes de L indéfiniment prolongeables dans L.

actement aux comportements fonctionnels engendrés par le programme sous-jacent. Considérons, par exemple, le programme présenté en Figure 1. On lui associe sa représentation canonique sous forme d'automate fini, en appliquant le morphisme $Instruction \longrightarrow a_i$ à l'ensemble des étiquettes. Tout chemin dans cet automate est étiqueté par un mot de la forme a^j, où j est la durée d'allocation CPU nécessaire pour l'exécution du comportement correspondant de la tâche. Dans le cas de cet exemple, l'ensemble est $a^2\left\{a^2,a^3\right\}\left\{a^2,a\left\{a^2\left\{a,a^2\right\}\right\}^{35}a\right\}$, qui s'écrit aussi $\left\{a^6,a^7,\left(a^{6+3\times35+i}\right)_{i\in[0,36]}\right\}$. D'une manière générale[9], cet ensemble est fini. Notons le L_{C_i}. Pour $\omega \in L_{C_i}$, $Centre\left(\bullet^{r_i}\left(\left(\omega \text{Ш} \bullet^{D_i-|\omega|}\right)\bullet^{T_i-D_i}\right)^*\right)$ collecte l'ensemble des comportements temporels valides de τ_i pour lesquels toute occurrence de τ_i possède un comportement de durée d'exécution $|\omega|$. Ce langage est accepté par l'automate dont l'allure est présentée sur la Figure 1.c. Nommons $(\omega_k)_{k\in I}$ les mots de l'ensemble L_{C_i}. Étant donné que L_{C_i} est fini, I l'est également. Les comportements des différentes occurrences de τ_i correspondent à des ω_k non nécessairement égaux. Un comportement temporel valide de τ_i est donc, globalement, élement de $Centre\left(\bullet^{r_i}\left(\bigcup_{k\in I}\left(\omega \text{Ш} \bullet^{D_i-|\omega|}\right)\bullet^{T_i-D_i}\right)^*\right)$. I étant un ensemble fini, ce langage est rationnel (l'allure de l'automate associé est présenté en Figure 1.d).

Les tâches sporadiques étant également modélisables par des langages rationnels, elles s'intègrent parfaitement dans cette étude. Le procédé de traitement est le même.

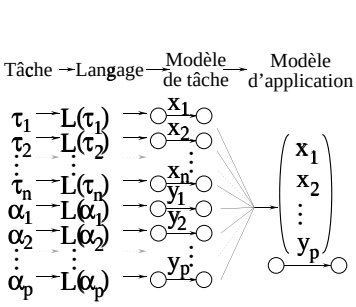
3. Modélisation de systèmes de tâches

Une application temps-réel est constituée d'un ensemble de tâches, indépendantes ou interdépendantes. Nous nous intéressons ici aux applications temps-réel à contraintes strictes, donc uniquement constituées de tâches soumises à des contraintes temporelles strictes. Une telle application est définie par la donnée de deux familles de tâches $(\tau_i)_{i\in[1,n]}$ et $(\alpha_i)_{i\in[1,p]}$: les α_i sont des sporadiques, et les τ_i des périodiques.

3.1. *Fonctionnement simultané de l'ensemble des tâches*

Un système temps-réel étant réactif, les tâches qui le composent fonctionnent simultanément. Cette simultanéïté est modélisée par des **produits de Hadamard** de langages : si $(A_i)_{i\in[1,n]}$ sont des alphabets, le produit de Hadamard est le morphisme

9. Notre hypothèse restrictive est ici de considérer que les programmes ne comportent, comme boucles, que des instructions de type *For…Do*, pour lesquelles le nombre d'itérations est décidable statiquement : dans le cadre de systèmes temps-réels à contraintes strictes, il s'agit une restriction raisonnable…



		Corps	Mod. temp.
Système de tâches	Tâche τ_1	Activité Attente Sec. Crit. Dem. P V	a P a V
	Tâche τ_2	Activité Attente Sec. Crit. Dem. P V	a P a V
	Tâche τ_3	Activité Attente Sec. Crit. Dem. P V	a P a V
Ress.	Tâche R_1	* P * V	a P a V

FIG. 2. *Des produits de Hadamard pour modéliser les systèmes de tâches*

de concaténation $(A_i)_{i\in[1,n]} \longrightarrow \prod_{k=1}^{k=n} A_i : (a_i)_{i\in[1,n]} \longmapsto \begin{pmatrix} a_1 \\ : \\ a_n \end{pmatrix}$. La sémantique associé au produit de Hadamard d'une famille de lettres (chaque lettre représentant l'état d'activité d'une tâche) est la simultanéïté[10](Voir Figure 2.Gauche). Dans la suite, nous notons $\underset{i=1}{\overset{i=n}{\Omega}} L_i$ le produit de Hadamard des langages $(L_i)_{i\in[1,n]}$. Notons que le produit de Hadamard d'une famille de centres de langages rationnels reste un centre de langages rationnels [EIL 76]. La notion de comportement temporel valide, définie plus haut pour les tâches, s'étend naturellement aux systèmes de tâches.

Définition 2 *On appelle comportement temporel valide d'une application temps-réel à contraintes strictes constituée de deux familles de tâches* $(\tau_i)_{i\in[1,n]}$ *et* $(\alpha_i)_{i\in[1,p]}$ *indépendantes tout mot de* $\left(\underset{i=1}{\overset{i=n}{\Omega}} L(\tau_i)\right) \Omega \left(\underset{i=1}{\overset{i=p}{\Omega}} L(\alpha_i)\right)$

Dans la suite de ce travail, nous notons indifféremment $L((\tau_i)_{i\in[1,n]}, (\alpha_i)_{i\in[1,p]})$ ou $\left(\underset{i=1}{\overset{i=n}{\Omega}} L(\tau_i)\right) \Omega \left(\underset{i=1}{\overset{i=p}{\Omega}} L(\alpha_i)\right)$ cet ensemble.

3.2. *Interdépendance des tâches*

Lorsque les tâches communiquent ou partagent des ressources critiques, l'ensemble des comportements temporels valides de l'application est un sous-ensemble de $L\left((\tau_i)_{i\in[1,n]}, (\alpha_i)_{i\in[1,p]}\right)$: certaines périodes d'activité de certaines tâches se trouvent retardées, du fait de l'exclusion mutuelle sur l'accès aux ressources, où du fait de l'attente de messages. Certaines comportements temporels valides dans le cas de l'indépendance des tâches ne sont plus valides. Pour détecter ces comportements, nous

10. Dans le cadre de la modélisation de systèmes concurrents, cette approche a été introduite par [ARN 94].

utilisons une tâche virtuelle associée à la ressource, et dont l'objet est de tracer ses états. Par ailleurs, nous modifions le morphisme $instruction \rightarrow a_i$ de façon à ce qu'il laisse inchangées les instructions critiques. Comme il garde sa propriété de *morphisme*, les résultats décrits plus haut restent valides.

Nous utilisons la technique conçue par Arnold et Nivat [ARN 94] pour la modélisation de la synchronisation de processus concurrents. Formellement, le calcul du langage des comportements valides s'exprime de la façon suivante. Soit $(R_i)_{i\in[1,r]}$ l'ensemble des ressources (resp. messages) partagées (resp. transmis ou reçus) par les tâches $(\tau_i)_{i\in[1,n]}$. À chaque R_i, nous associons le langage $L(R_i)$ destiné à tracer son historique. R_i peut donc apparaître à un ou plusieurs exemplaires, à condition que le nombre d'exemplaires soit fini[11]. Nous construisons ensuite le langage $L_E = \left(\underset{i=1}{\overset{i=n}{\Omega}} L(\tau_i)\right) \Omega \left(\underset{i=1}{\overset{i=r}{\Omega}} L(R_i)\right)$. Soit maintenant S le sous-ensemble de l'alphabet support de L_E dont chaque élément v vérifie la propriété

$$\forall (i,x) \in [1,r] \times \{P,V\}, v_{n+i} = x \Longleftrightarrow \exists! j \in [1,n] \text{ tel que } v_j = x$$

S contient donc l'ensemble des configurations d'activité instantannée valides du système : lorsqu'une tâche effectue une instruction critique (P ou V, lorsqu'il s'agit d'une ressource, S ou R lorsqu'il s'agit d'une transmission), la tâche virtuelle qui trace l'état de la ressource correspondante la prend effectivement en compte. On appelle ce langage le **produit synchronisé** du système de tâches.

3.3. *Application au calcul d'ordonnançabilité*

3.3.1. *Cas des tâches à durées fixes*

L'intersection de langages n'est pas interne dans la classe des centres de langages rationnels. Dans notre étude, cette propriété traduit le fait que le partage de ressources (par exemple) entre n tâches soumises à des contraintes strictes peut amener certaines d'entre elles en situation de faute temporelle. Ce résultat est bien connu, et a motivé dans le passé un grand nombre de travaux [BAK 91], l'objet étant de décider de l'ordonnançabilité du système de tâches dans ce contexte d'interdépendance. Nous allons voir que notre approche permet une résolution extrêmement efficace de ce problème.

Le produit synchronisé $L\left((\tau_i)_{i\in[1,n]}\right)$ est partitionné en

$$Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right) \cup \left(L\left((\tau_i)_{i\in[1,n]}\right)\right) \setminus Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right)$$

La composante $Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right)$ collecte l'ensemble des comportements temporels valides, alors que la composante $\left(L\left((\tau_i)_{i\in[1,n]}\right)\right) \setminus Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right)$

11. Ce n'est pas vraiment une restriction, pour un système réel.

collecte les comportements temporels non valides : n'appartenant pas au centre du langage, ils modélisent les séquences d'allocation CPU aux différentes tâches telles qu'il est certain que, dans le futur, l'une au moins des tâches ne sera pas en mesure de respecter ses échéances temporelles. Le centre du produit synchronisé fournit donc exactement l'ensemble des comportements valides. L'ordonnançabilité de la configuration est donc obtenue à partir du prédicat $Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right) \neq \emptyset$.

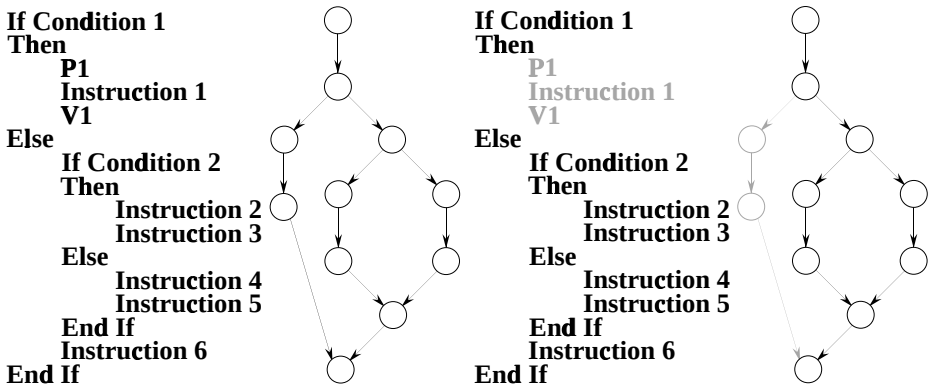
3.3.2. *Cas des tâches à durées variables*

Lorsque les tâches sont à durées variables, le calcul du centre ne suffit plus car, pour une tâche donnée, la synchronisation par la technique d'Arnold-Nivat peut interdire à la tâche certains de ses comportements individuels (par exemple les plus longs), et en autoriser d'autres. Prise tel quel, la méthode de validation temporelle décrite ci-dessus décide la propriété *pour chaque tâche, il existe au moins un chemin d'exécution toujours valide*. Cette propriété est, bien entendu, totalement insuffisante : il n'est (fonctionnellement) pas tolérable que le champ de fonctionnement d'une tâche soit restreint par le système. Il est donc nécessaire de décider si une tâche conserve, dans un modèle temporel synchronisé d'application, la possibilité d'exécuter l'ensemble de ses chemins. Pour ce faire, il va suffire de renforcer la notion de validité temporelle des comportements.

Pour simplifier le discours, nous raisonnons ici sur des tâches périodiques uniquement. toutefois, comme tout le raisonnement repose sur la rationnalité des modèles temporels de tâches, il reste valide pour les systèmes intégrant des sporadiques. Considérons, par exemple, le cas d'une système de tâches $(\tau_i)_{i\in[1,n]}$, chaque τ_i pouvant (ou non) être à durée variable. Les tâches τ_1 et τ_2 sont amenées à partager une ressource critique. Le programme associé à τ_1 est donné en Figure 3.Gauche[12]. Mais leurs programmes respectifs sont conçus de elle sorte que, lorsqu'il y a partage, aucune des deux tâches n'est à même de respecter ses échéances. Le résultat normalement attendu est que le système n'est pas ordonnançable. Or, pour chacune des deux tâches, il existera une possibilité de fonctionnement compatible avec ses contraintes temporelles. Pour la tâche τ_1, ce sont l'ensemble des chemins excluant le partage de ressource avec τ_2. La partie considérée comme temporellement valide du programme de τ_1 est donnée en noir dans le Figure 3.Droite, la partie en grisé, considérée comme non temporellement valide, étant simplement ignorée par l'analyse d'ordonnançabilité. Dans ce cas précis, le résultat d'ordonnançabilité fourni par la méthode est faux : le système est diagnostiqué comme étant ordonnançable, alors qu'il ne l'est pas. Cette propriété doit être caractérisée, pour déterminer si la configuration est acceptable : c'est le cas si chaque tâche dispose de la possibilité de suivre l'ensemble de ses chemins. Ceci nous amène à la définition suivante :

Définition 3 *Un système de tâches $(\tau_i)_{i\in[1,n]}$ est* **globalement ordonnançable** *si tout chemin de toute tâche est possible pour chacune de ses occurrences.*

12. Dans la suite, on appellera **chemin d'exécution** –ou simplement *chemin*– de la tâche toute trace d'un chemin dans l'automate canoniquement associé à ce programme. On notera $X(\tau_i)$ l'ensemble fini de ces chemins.



Corps de la tâche τ_1 Partie valide du corps de la tâche τ_1

FIG. 3. *Comportements non globalement valides*

Un mot ω du produit est de la forme $\begin{pmatrix} \omega_{1_1} \\ : \\ \omega_{1_n} \end{pmatrix} \begin{pmatrix} \omega_{2_1} \\ : \\ \omega_{2_n} \end{pmatrix} \dots \begin{pmatrix} \omega_{|\omega|_1} \\ : \\ \omega_{|\omega|_n} \end{pmatrix}$. Projeté sur sa $i^{\text{ème}}$ composante, ω fournit $\pi_i(\omega) = \begin{pmatrix} \omega_{1_i} \end{pmatrix} \begin{pmatrix} \omega_{2_i} \end{pmatrix} \dots \begin{pmatrix} \omega_{|\omega|_i} \end{pmatrix}$, élément de $L(\tau_i)$. Si, dans ce comportement, on supprime l'ensemble des $\bullet$ (correspondant aux instants d'inactivité de τ_i, on obtient un chemin d'exécution de τ_i. La configuration est valide si, dans $Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right)$, ce chemin peut être couplé avec l'ensemble des chemins d'exécution de l'ensemble des autres tâches, dans le respect des contraintes temporelles de l'ensemble de l'application. C'est ce que nous exprimons dans la propriété suivante, en préalable de laquelle nous posons quelques notations.

Notation

Dans la suite, pour tout $i \in [1,n]$, on note π_i le morphisme $(x_j)_{i\in[1,n]} \to x_i$. Pour tout alphabet Σ et tout $A \subset \Sigma$, on note π_A le morphisme $\begin{cases} x \in A \Leftrightarrow \pi_A(x) = x \\ x \notin A \Leftrightarrow \pi_A(x) = \varepsilon \end{cases}$ et par $\pi_{\neg A}$ le morphisme $\begin{cases} x \in A \Leftrightarrow \pi_{\neg A}(x) = \varepsilon \\ x \notin A \Leftrightarrow \pi_{\neg A}(x) = x \end{cases}$

Le résultat suivant permet de décider si le centre du produit synchronisé collecte exactement l'ensemble des comportements temporels globalement valides d'un système de tâches :

Théorème 4 *Un ensemble de tâches $(\tau_i)_{i\in[1,n]}$ est globalement ordonnançable sur une configuration matérielle donnée si et seulement si*

$$\forall (x_i)_{i\in[1,n]} \in \prod_{i=1}^{i=n} (X(\tau_i)), \exists \omega \in Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right) \text{ tel que}$$

$$\forall i \in [1,n], \pi_{\neg\bullet}(\pi_i(\omega)) = x_i$$

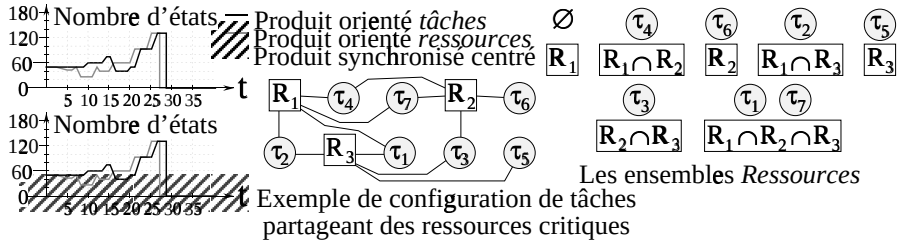


FIG. 4. *Une configuration de tâches*

L'ordonnançabilité globale est donc une caractéristique du centre du produit synchronisé : soit le système est globalement ordonnançable et, dans ce cas, le centre fournit l'ensemble des séquences d'ordonnancement valides, soit il ne l'est pas, et la spécification doit être revue en amont.

Si le système de tâches est ordonnançable, l'ensemble de ses séquences d'ordonnancement valides est un langage rationnel. Le lemme de l'étoile permet donc, à partir du théorème 4, d'obtenir le résultat suivant, qui généralise celui de [GRO 99] au cas d'une part des systèmes de tâches à durées variables, et d'autre part au cas des ordonnancements en environnement multi-processeurs (sous l'hypothèse de migration totale des tâches) :

Corollaire 5 *Soit $(\tau_i)_{i\in[1,n]}$ un ensemble de tâches interdépendantes, à durées variables, à ordonnancer sur une configuration matérielle mono-processeur ou multi-processeurs. Les séquences d'ordonnancement valides pour ce système sont cycliques.*

4. Recherche d'une stratégie efficace pour le produit synchronisé

Notre but est ici d'étudier les stratégies de calcul permettant de limiter la taille des automates résultant des calculs intermédiaires intervenant lors de la construction du modèle temporel $Centre\left(L\left((\tau_i)_{i\in[1,n]}\right)\right)$.

Considérons, par exemple, une configuration de sept tâches périodiques, possédant chacune les caractéristiques $r_i = 0$, $C_i = 6$, $D_i = 9$, $T_i = 10$, le système partageant trois ressources (voir figure 4). Nous appelons *ensemble ressource* les ensembles R_i contenant les tâches τ_i qui utilisent la ressource R_i (Voir Figure 4). Sur cet exemple, le produit de Hadamard et la synchronisation correspondante sont réalisés de la manière suivante (les opérations sont réalisées dans l'ordre d'écriture) :

$$(L(\tau_1)\,\Omega L(\tau_2)\,\Omega L(\tau_3)\,\Omega L(\tau_4)\,\Omega L(\tau_5)\,\Omega L(\tau_6)\,\Omega L(\tau_7)) \cap S$$

Les automates associés aux τ_i comportent chacun une cinquantaine d'états. Le calcul du modèle temporel passe par un produit de Hadamard (concernant trois tâches) de $1,25 \times 10^4$ états, le produit de sept tâches en comporte $7,8125 \times 10^{11}$ et enfin après la synchronisation, le modèle temporel obtenu comme résultat contient 131 états.

La méthode naïve consiste à calculer d'abord le produit de Hadamard des $L(\tau_i)$, puis à synchroniser. La complexité est en $O\left(\underset{i\in[1,n]}{ppcm}(T_i)\right)$.

Le nombre d'états de l'automate résultat est nettement plus petit que celui des automates intermédiaires. En fait, les synchronisations (partage de ressources), implémentées sous la forme d'intersections de langages, ont pour effet de supprimer les parties inutiles des automates. En les effectuant le plus tôt possible, on retarde l'explosion combinatoire, voire même on l'évite, si le système est suffisamment contraint. Nous intercalons donc les synchronisations avec les ressources entre les produits de tâches (i.e. on fait systématiquement un produit de deux automates que l'on synchronise immédiatement).

Soient S_p l'ensemble des configurations instantanées dans lesquelles le nombre de tâches actives est inférieur ou égal au nombre de processeurs disponibles, R_i l'automate virtuel associé à la ressource R_i [ARN 94], et S_i l'ensemble des n-uplets correspondant aux configurations instantannées conformes au protocole d'utilisation de R_i. Sur l'exemple, la forme du produit obtenu en intercalant les synchronisations est : $((L(\tau_1)\,\Omega L(\tau_2))\cap S_p)\,\Omega S_1$ etc.

Nous avons testé différentes approches :
- Un calcul orienté *tâches*, qui consiste à faire le produit de Hadamard des tâches utilisant la même ressource puis à synchroniser avec S_i, ceci successivement pour toutes les ressources
- Un calcul orienté *ressources*, qui traite d'abord les ensembles de tâches qui sont soumises aux plus fortes contraintes pour maximiser la suppression de transitions due à la synchronisation sur les ressources

Pour le calcul orienté *tâches*, nous prenons les ensembles ressources un par un et calculons d'abord le produit de Hadamard (synchronisé sur le nombre de tâches actives) de toutes les tâches d'un ensemble ressource R_i puis nous synchronisons avec la ressource correspondante S_r (les R_i sont traités dans l'ordre croissant des cardinaux). La figure 4 présente l'évolution de l'occupation mémoire en fonction du temps, pour notre exemple test. La croissance du nombre d'états en fonction du temps traduit la bonne qualité de la stratégie de coupure des automates.

Pour le calcul orienté *ressources*, nous commençons par les tâches qui utilisent le plus de ressources. On calcule les ensembles R_i, puis les classes d'équivalence $[R_1]$, $[R_2]$, …, $[R_1, R_2]$, $[R_1, R_3]$, …, $[R_1, R_2, R_3]$, …, qui vérifient[13]

$$\forall R \in \wp\left(\left\{(R_i)_{i\in[1,n]}\right\}\right), [R] = \left\{\tau_j \text{ tels que } \tau_j \to R \text{ et } \tau_j \not\to \left(\left\{(R_i)_{i\in[1,n]}\right\} \setminus R\right)\right\}$$

Chacun de ces ensembles contient les tâches qui utilisent les ressources indiquées et uniquement celles là. Ainsi, chaque tâche n'apparaît que dans un de ces ensembles. Sur la figure 4, on constate que le produit orienté ressources est meilleur (au sens du

13. On note $\tau \to R$ le fait que la tâche τ utilise la ressource R, et par $\tau\neg \to R$ le fait qu'elle ne l'utilise pas.

nombre d'états) que le produit orienté tâches pendant les quatorze premières secondes. À partir de cet instant, la courbe *ressources* se trouve au dessus de la courbe *tâches*. Celà traduit l'avance du produit orienté *ressources* sur le produit orienté *tâches*.

Si le centre du langage associé à une configuration de tâches est vide, alors cette configuration n'est pas ordonnançable, il en est donc de même pour toute extension de cette configuration. Il faut donc essayer de décider de cette ordonnançabilité le plus tôt possible, c'est à dire sur le plus petit sous-ensemble possible de la configuration de tâches. La décision est alors acquise pour l'ensemble de système, et la suite du calcul est inutile. Aussi, nous entrelaçons synchronisations et calculs de centre[14]. Cette technique améliore le produit orienté ressources, dans le sens où, pour une configuration donnée, elle permet d'obtenir la décision d'ordonnançabilité précocément. Sur notre exemple, notamment, elle permet une détection bien plus précoce de la non ordonnançabilité (voir Figure 4). Nous appelons cette technique *produit synchronisé centré*. L'outil développé repose sur cette stratégie de calcul.

5. Conclusion

L'utilisation des langages rationnels pour l'analyse d'ordonnançabilité s'est donc révélée très fructueuse. D'une part, le modèle est adapté aux tâches à durées variables, pour lesquelles les résultats établis dans [GEN 00] restent valides, et d'autre part nous ètendons le critère d'ordonnançabilité pour les systèmes constitués de telles tâches. À notre connaissance, à ce jour, un tel critère n'a été obtenu par aucune autre approche. Par ailleurs, nous étendons le résultat de la cyclicité des ordonnancements d'un système temps-réel (établi par [GRO 99] pour les systèmes périodiques en environnement mono-processeur) au cas des systèmes de tâches à durées variables en environnement multi-processeur, sous l'hypothèse de migration totale.

Pour les systèmes intégrant des tâches sporadiques, notre méthodologie fournit des résultats similaires. Ces tâches pouvant se modéliser à l'aide de langages rationnels, les résultats de décision d'ordonnançabilité et de cyclicité obtenus pour les systèmes périodiques persistent pour les systèmes intégrant des sporadiques.

Actuellement, nous étudions d'une part les capacités d'analyse de notre modèle pour les systèmes distribués, et d'autre part les méthodes effectives d'évaluation de la propriété caractéristique du théorème 4. Une étude en cours de démarrage est centrée sur l'utilisation du modèle temporel comme outil de pilotage effectif d'applications. Pour les systèmes de tâches à durées variables, en effet, même si l'ordonnançabilité globale est acquise, chaque commutation de contexte pose le problème du choix de la tâche à sélectionner, choix pour l'instant fortement non déterministe. Les fonctions génératrices introduites dans [GEN 01] constituent vraissemblablement une piste intéressante pour la levée de ce non déterminisme. Par ailleurs, les études comportementales du procédé de calcul semblent montrer qu'une explosion combinatoire reflète

14. Le calcul de centre est linéaire en le nombre d'états de l'automate, un produit de Hadamard $A\Omega B$ est proportionnel à $Taille(A) \times Taille(B)$.

souvent une sous-charge du système. Nous cherchons à raffiner le modèle pour limiter cette explosion, tout en conservant la puissance des outils d'analyse de langages rationnels, et la compatibilité avec l'enrichissement du modèle [GEN 01].

6. Bibliographie

[ACE 98] ACETO L., BOUYER P., NO A. B., LARSEN K. G., The Power of Reachability Testing for Timed Automata, *Proc. of* 18^{th} *Conf. Found. of Software Technology and Theor. Comp. Sci.*, LNCS 1530, Springer-Verlag, December 1998, p. 245-256.

[ALU 94] ALUR R., DILL D., A Theory of Timed Automata, *Theoretical Computer Science*, vol. 126, 1994, p. 183-235.

[ARN 94] ARNOLD A., *Finite transition systems*, Prentice Hall, 1994.

[BAK 91] BAKER T., Stack-Based Scheduling of Real-Time Processes, *the Journal of Real-Time Systems*, vol. 3, 1991, p. 67-99.

[BAR 90] BARUAH S., ROSIER L., HOWELL R., Algorithms and Complexity Concernig the Preemptive Scheduling of Periodic, Real-Time Tasks on one Processor, *Real-Time Systems*, , 1990, p. 301-324.

[CHO 96] CHOQUET-GENIET A., GENIET D., COTTET F., Exhaustive Computation of the Scheduled Task Execution Sequences of a Real-Time Application, *Proc. of FTRTFT'96*, October 1996.

[CHO 99] CHOQUET-GENIET A., Ordonnancement des Applications Temps-Réel, *Actes de l'école d'éte E.T.R'99*, 1999, p. 53-68.

[CHO 00] CHOQUET-GENIET A., *Systèmes parallèles et Temps-Réel : Analyse à l'Aide de Modèles Formels*, Mémoire d'Habilitation à Diriger les Recherches, Univ. Poitiers, December 2000.

[DEL 99] DELACROIX J., Ordonnancement de Tâches Apériodiques, *Actes de l'école d'éte E.T.R'99*, 1999, p. 69-82.

[EIL 76] EILENBERG S., *Automata Languages and machines*, vol. A, Academic Press, 1976.

[FLO 85] FLORIN G., NATKIN S., Les Réseaux de Petri Stochastiques, *Techniques et Sciences Informatiques*, vol. 4, n° 1, 1985, p. 143-160.

[GEN 00] GENIET D., Validation d'Applications Temps-Réel à Contraintes Strictes à l'Aide de Langages Rationnels, *RTS'2000*, 2000, p. 91-106.

[GEN 01] GENIET D., DUBERNARD J., Ordonnancement de Tâches Sporadiques à Contraintes Strictes à l'Aide de Séries Génératrices, *Proc. of RTS'01*, 2001.

[GRO 99] GROLLEAU E., Ordonnancement Temps-Réel Hors-Ligne Optimal à l'Aide de Réseaux de Petri en Environnement Monoprocesseur et Multiprocesseur, PhD thesis, Univ. Poitiers, 1999.

[LEU 80] LEUNG J., MERILL M., A Note on Preemptive Scheduling of Periodic Real-Time Tasks, *Information Processing Letters*, vol. 11, n° 3, 1980, p. 115-118.

[LIU 73] LIU C., LAYLAND J., Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *Journal of the ACM*, vol. 20, n° 1, 1973, p. 46-61.

[RAM 74] RAMCHANDANI C., Analysis of Asynchronous Concurrent Systems by Petri Nets, rapport, 1974, MIT, project MAC.