AFADL'98, Approches Formelles dans l'Assistance au Développement de Logiciels, pp. 137-148, Futuroscope, 30/09-1/10/1998

Validation de Systèmes Temps Réel à l'Aide de Réseaux de Petri

Emmanuel GROLLEAU, Annie CHOQUET–GENIET, Francis COTTET
LISI–ENSMA
Site du FUTUROSCOPE BP 109
86960 FUTUROSCOPE Cedex
E–mail grolleau@ensma.fr, ageniet@ensma.fr, cottet@ensma.fr

Résumé : Nous présentons un outil d'aide au choix de séquences d'ordonnancement, optimales au vu de certains critères, de systèmes de tâches temps réel. La méthodologie d'ordonnancement sous-jacente permettant la détermination de l'ensemble des séquences optimales est basée sur une modélisation par réseaux de Petri colorés à contraintes de successeur avec ensemble terminal. Les systèmes de tâches considérés peuvent partager des ressources et se synchroniser.

Mots clés : Application temps–réel, réseau de Petri, ordonnancement optimal

1. Introduction

Un système temps réel se distingue des autres systèmes informatiques par le fait qu'il est soumis à des contraintes temporelles fortes, dues à la criticité de certaines actions qui lui permettent d'interagir avec l'environnement d'un procédé à contrôler [15]. Le modèle classiquement utilisé pour implémenter un système temps réel est celui présenté par Liu et Layland [11]. Il décrit une application temps réel à l'aide d'un système de tâches cycliques définies par quatre paramètres temporels : leur date de première activation, leur durée (au pire, bien que celle-ci soit non triviale à obtenir), leur délai critique (temps imparti pour s'exécuter) et leur période d'activation. La période, le délai critique et la date de première activation sont choisis pour satisfaire la partie temporelle du cahier des charges. Ces paramètres ne sont pas figés mais doivent être choisis pour permettre au système de satisfaire les contraintes temporelles qui sont imposées au procédé. Des études ont montré qu'il pouvait être intéressant de faire varier les paramètres pour obtenir de meilleurs comportements des systèmes [8].

La validation d'un système de tâches temps réel nécessite un processus de développement-validation plus long que les applications « classiques ». En effet, une fois la validation logicielle effectuée, il reste à montrer la validité temporelle de l'application. Cette validation ne peut s'effectuer qu'en fin de cycle de développement, car la durée des tâches dépend de leur codage. On peut déplorer le manque de portabilité des applications temps réel puisque la durée dépend aussi de l'architecture matérielle utilisée.

Valider le comportement temporel d'un système de tâches obtenu à partir d'un cahier des charges, c'est valider l'ordonnancement des tâches sur le ou les processeurs. On dira donc qu'un ordonnancement est valide s'il permet à toutes les tâches de respecter leurs contraintes temporelles. L'ordonnancement de systèmes de tâches est donc une problématique majeure du temps réel.

Lorsque les tâches sont indépendantes, des algorithmes polynomiaux permettent de trouver des ordonnancements valides pour peu que le système ne charge pas le processeur outre mesure. La plupart d'entre eux sont basées sur l'affectation de priorités (statiquement ou dynamiquement) aux tâches. Il existe des techniques analytiques [10][11][16] (i.e. ne nécessitant pas la construction de séquences d'ordonnancement) permettant de conclure quant à l'ordonnançabilité des systèmes de tâches indépendantes.

Cependant, un modèle ne permettant d'ordonnancer que des tâches indépendantes est limitatif. En effet, dans la plupart des systèmes de tâches, des communications induisent des contraintes de précédence entre certaines parties des tâches, mais surtout, celles-ci sont amenées à partager des ressources critiques. La prise en compte des contraintes de précédence peut être faite, sans modification du modèle classique, en découpant les tâches au niveau des points de communication, puis en modifiant les dates de première activation et les délais critiques des tâches ainsi obtenues [5]. Cette méthode a été étendue aux systèmes de tâches partageant des ressources critiques dans [14], elle permet d'ignorer les contraintes de précédence, mais pas les exclusions mutuelles engendrées par le

partage de ressources. En effet, dès lors que des ressources doivent être partagées, le problème de l'ordonnancement devient NP-difficile [12].

Un certain nombre d'algorithmes approchés ont été mis en oeuvre pour remédier à ce problème. En dehors du phénomène connu d'interblocage, l'un des phénomènes les plus caractéristiques engendré par l'adjonction de ressources critiques à un système est l'inversion de priorité. Une tâche peu prioritaire détenant une ressource s'exécute, elle est interrompue pour laisser le processeur à une tâche de priorité moyenne, et lorsqu'une tâche de haute priorité, nécessitant la ressource de la tâche de basse priorité, s'éveille, celle-ci doit attendre la terminaison de la tâche de priorité intermédiaire, puis la libération de la ressource par la tâche de basse priorité, avant de pouvoir s'exécuter. Dans ce cas, le temps perdu par la tâche de haute priorité à cause de l'inversion de priorité n'est 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 [13], le protocole à priorité plafond [4][13] ou le protocole de gestion des ressources par piles [2]. La propriété de tels protocoles est de limiter, voire de borner la durée des inversions de priorité.

Cependant, ces algorithmes restent approchés et nécessitent une simulation hors-ligne (i.e. construction de la séquence d'ordonnancement à priori) avant d'êtres implantés en-ligne, c'est à dire à l'aide d'un ordonnanceur implanté dans le système d'exploitation cible.

Puisqu'une simulation du système conduisant à la construction d'une séquence d'ordonnancement est indispensable à la validation temporelle d'un système de tâches temps réel, plutôt que de mettre en oeuvre une politique d'affectation de priorités et de gestion des ressources afin d'implanter un ordonnanceur, on peut utiliser des techniques d'ordonnancement hors-ligne. Cette approche a plusieurs avantages sur la première : diminution de la surcharge due au calcul des priorités, simplicité de mise en oeuvre d'un séquenceur. Un de ses principaux inconvénients est son manque de souplesse. En effet, les durées d'exécution des tâches sont souvent surestimées lors de la spécification des tâches, et on peut estimer qu'un ordonnanceur mettra à profit le temps gagné pour exécuter d'autres tâches. Nous montrons en section 3 qu'un tel inconvénient est tout à fait relatif dès lors qu'il existe des ressources critiques.

La littérature propose différents algorithmes d'ordonnancement hors-ligne qui sont soit approchés (algorithmes génétique, recuit simulé [3]), soit de complexité exponentielle (logique temporelle [1]). Notre approche [6][9] se trouve dans ce cas, mais elle a l'originalité de proposer la construction de séquences optimales au vu de certains critères, et elle peut être intégrée à un outil d'aide au choix de séquences d'ordonnancement optimales au vu de certains critères. Elle est basée sur la construction de réseaux de Petri à contraintes terminales dont le langage, lorsqu'il fonctionne à vitesse maximale, est l'ensemble de toutes les séquences d'ordonnancement valides du système de tâches modélisé.

Dans la section 2, nous présentons brièvement le modèle de tâches classique. En section 3, nous justifions l'intérêt porté à l'approche hors-ligne par rapport à l'approche en-ligne. En section 4 nous présentons le modèle utilisé. En section 5, nous montrons comment exploiter ces séquences pour n'obtenir que des séquences optimales, puis nous présentons l'outil mettant ces techniques en oeuvre en section 6.

2. Définitions et notations usuelles

Le modèle classique représentant les tâches temps réel a été brièvement présenté en introduction. Il modélise chaque tâche τ_i à l'aide de quatre paramètres temporels :

- r_i, la date de réveil de τ_i est la date à laquelle τ_i est activée pour la première fois.
- C_i, la charge de τ_i est sa pire durée d'exécution.
- R_i, le délai critique de τ_i est le temps qui lui est imparti pour s'exécuter à chacune de ses activations.
- T_i est la période d'activation de τ_i qui est donc réactivée toutes les T_i unités de temps.

Ces paramètres sont présentés par un diagramme de Gantt sur la figure 1.

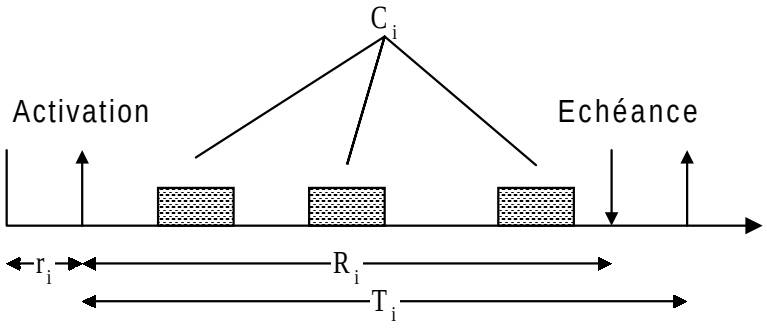

Figure 1 : Paramètres temporels des tâches temps réel

Une tâche est donc caractérisée temporellement par $\tau_i = <r_i, C_i, R_i, T_i>$. 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 $\frac{C_i}{T_i}$ représente le taux d'utilisation processeur requis par la tâche τ_i, on voit que U représente le taux d'utilisation requis par l'ensemble des tâches. Si la charge U d'un système est supérieure au nombre de processeurs, il est certain que ce système ne peut pas être ordonnancé.

3. Ordonnancement et séquencement

Lorsqu'il est confronté à un système de tâches temps réel partageant des ressources critiques, le programmeur a le choix entre deux approches fondamentales :

- L'approche en-ligne, qui consiste à implanter sur la machine cible de l'application un ordonnanceur, avec une politique d'ordonnancement choisie.
- L'approche hors-ligne, qui consiste à construire une séquence d'ordonnancement valide du système étudié, puis à implanter un séquenceur sur la machine cible.

L'avantage indéniable de l'approche en-ligne est sa souplesse d'utilisation. On pourra par exemple, si toutes les contraintes temporelles ne sont pas strictes (i.e. certaines tâches peuvent dépasser leur échéance), prévoir des ajouts dynamiques de tâches (périodiques ou apériodiques) avec des méthodes de garantie d'échéance. L'avantage de l'approche hors-ligne est sa simplicité lors de sa mise en oeuvre et l'efficacité du séquenceur qui a un moindre coût processeur qu'un ordonnanceur.

3.1 Ordonnancement

Nous nous intéressons aux systèmes de tâches pouvant communiquer et partager des ressources. Dans le cadre de l'ordonnancement, on choisira un protocole d'allocations de ressources, garantissant de préférence la bornitude des inversions de priorité, et donc celle des durées de blocage des tâches lors de l'attente d'une ressource critique. Ainsi la charge initiale des sections critiques devra être modifiée pour intégrer la durée de blocage maximale lors de l'attente de la ressource. Cette démarche, la plus communément utilisée, pose d'une part des problèmes de mise en oeuvre au niveau de l'ordonnanceur, qui devra gérer les accès aux ressources et les communications, et d'autre part l'intégration de la durée de blocage maximale dans les sections critiques augmente sensiblement la charge d'un système de tâches.

Par exemple, considérons une application S composée de trois tâches, $S=\{\tau_1<0,4,16,16>, \tau_2<0,2,2,8>, \tau_3<0,6,15,16>\}$ où τ_1 est une tâche destinée à l'affichage de fenêtres graphiques sur une console, et utilise pendant toute son exécution la ressource critique console, τ_2 est une tâche de type alarme, avec un délai critique très court, qui affiche une fenêtre d'erreur et qui nécessite aussi la ressource console pendant son exécution, et τ_3 est une tâche de scrutation/calcul indépendante.

La charge de S est de 87,5%, et si l'on intègre la durée de blocage maximale de τ_2 par τ_1 (la charge de τ_2 devient 1+4=5 unités de temps) à cause de la console, la charge du système passe, quel que soit

le protocole de gestion de ressources utilisé, à 137,5%, ce qui rend le système non ordonnançable sur un processeur. Pourtant, sur la figure 2, le système semble ordonnançable par *Earliest Deadline*, dans lequel la priorité la plus grande est affectée à la tâche dont l'échéance est la plus proche.

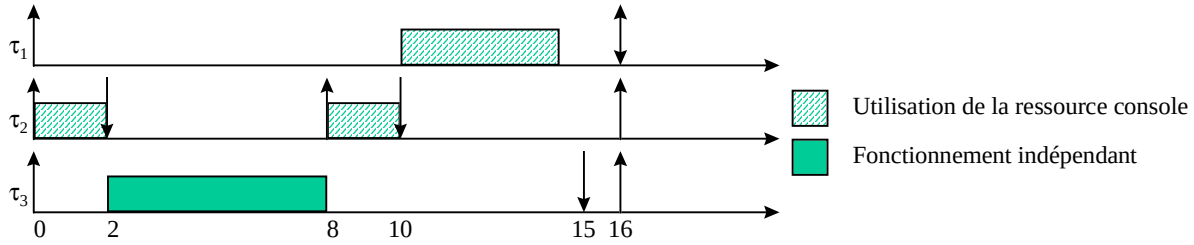

Figure 2 : Etude hors-ligne du système S avec Earliest Deadline.

La séquence produite est valide (i.e. toutes les échéances sont respectées). Cependant, déduire que le système est ordonnançable par *Earliest Deadline* n'est pas tout à fait exact. En effet, rappelons que la charge C_i des tâches est calculée au pire lors de l'étude théorique. Dès lors que des ressources critiques sont en jeu, le pire cas n'est pas la pire durée. Pour exemple, nous considérons sur la figure 3 un comportement possible de S pendant son fonctionnement réel, où la durée de τ_3 n'est que de 5 unités de temps au lieu de 6.

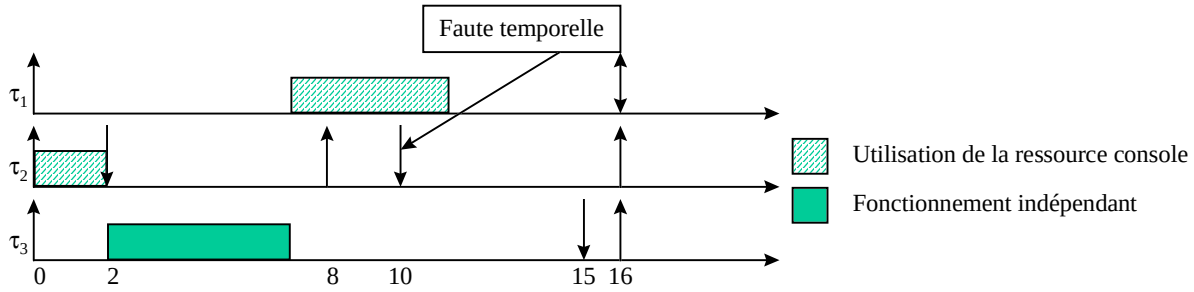

Figure 3 : Un comportement indésirable du système S.

Cet exemple montre que lorsque le système à ordonnancer doit garantir des exclusions mutuelles, si les durées de blocage ne peuvent pas être intégrées dans la durée des sections critiques, on ne peut pas garantir une politique d'ordonnancement, mais seulement une séquence produite.

Lorsque les tâches sont très contraintes en terme de ressources critiques, les techniques d'ordonnancement entraînent une augmentation des charges des tâches qui peut être fatale à l'ordonnançabilité du système. Il est donc moins contraignant de garantir une séquence qu'une politique d'ordonnancement.

3.2 Séquencement

Plusieurs approches ont été menées dans le domaine du séquencement. Ces approches sont soit basées sur des algorithmes aléatoires, soit basées sur des algorithmes non polynomiaux en temps. Cette catégorie regroupe principalement les approches utilisant la logique temporelle, les automates temporisés, et les réseaux de Petri temporisés ou non. Notre approche se base sur l'utilisation de réseaux de Petri non temporisés, mais fonctionnant à vitesse maximale, c'est à dire qu'à chaque tir de transition, c'est un ensemble maximal de transitions franchissables au même instant qui est tiré.

4. Modélisation de systèmes de tâches temps réel à l'aide des réseaux de Petri

Dans ce paragraphe nous présentons la modélisation de systèmes de tâches temps réel à l'aide des réseaux de Petri colorés avec marquages terminaux proposée dans [6]. Pour une définition des réseaux de Petri, le lecteur peut se reporter à [7]. P est l'ensemble des places du réseau, T l'ensemble des transitions, et $M : P \rightarrow N \cup \{a\} \cup \{b\}$ la fonction de marquage qui donne les jetons contenus dans une place où N est l'ensemble des entiers naturels, et a et b sont des couleurs de jetons. Notons qu'ici, les

couleurs sont utilisées uniquement dans un but de lisibilité, et les règles de franchissement des transitions sont intuitivement les mêmes que pour les réseaux de Petri non colorés.

Ce modèle permet d'obtenir toutes les séquences d'ordonnancement possibles d'un système de tâches modélisé.

4.1 Modèle initial

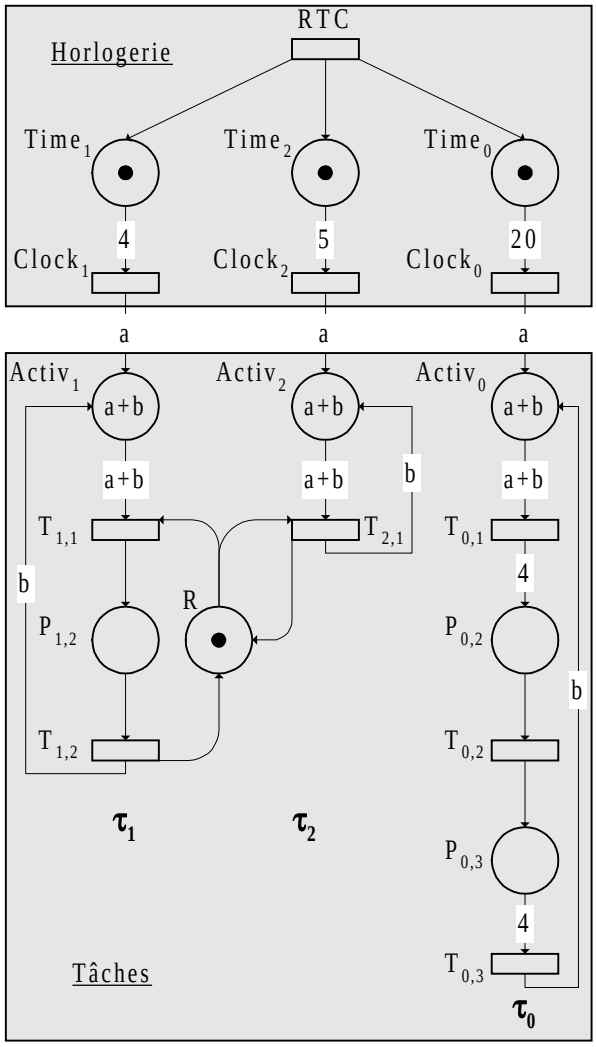

Figure 4: Modélisation d'un système de 3 tâches à l'aide d'un réseau de Petri

Nous donnons sur la figure 4 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 à vitesse maximale du réseau de Petri, cette transition est tirée à chaque tir de transitions. C'est elle qui va ainsi temporiser, en produisant à chaque fois un jeton dans chaque horloge locale, le comportement du réseau. Dans la suite on associera unité de temps à tir d'un ensemble de transitions.

- les horloges locales des tâches qui permettent de réactiver les tâches à chaque fin de période de celles-ci. 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 *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, qui sont représentées de manière classique. 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 que soit elle reçoit un jeton *b* (la tâche a eu une latence de 0 pendant sa dernière période), soit elle en contient déjà un (sa latence était plus grande que 0). Si la tâche n'est pas à échéance sur requête, lorsque l'horloge locale de la tâche dépasse le délai critique R_i, la tâche doit avoir terminé son exécution, ce qui veut dire que $M(Time_i)>R_i \Rightarrow M(Activ_i)\geq b$. 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*. Donc, comme $M(Time_i)=1$ uniquement lorsque la tâche τ_i vient d'être (ré)activée, il est nécessaire, pour que la tâche respecte son

échéance, que $M(Time_i)=1 \Rightarrow M(Activ_i)=a+b$. Les échéances des tâches, qu'elles soient à échéance sur requête ou non, nécessitent donc que

$(M(Time_i)>R_i \Rightarrow M(Activ_i)=b)$ (1) et $(M(Time_i)=1 \Rightarrow M(Activ_i)=a+b)$ (2)

pour un marquage donné M.

Cette contrainte est utilisée pour définir le marquage terminal du modèle. Un marquage n'est valide que si il fait partie de l'ensemble terminal. Il doit donc respecter les contraintes (1) et (2).

Les transitions du système de tâches sont mises en exclusion mutuelle à l'aide d'une place processeur qui n'est pas représentée graphiquement par raison de lisibilité. Cela impose qu'une seule tâche peut avancer à un instant donné puisque nous faisons fonctionner le réseau en parallèle à vitesse maximale.

On étiquette les transitions d'horlogerie par le mot vide, et celles du système de tâches par le nom des tâches correspondantes. Ainsi, le langage terminal (i.e. chaque marquage obtenu est terminal) du réseau de Petri fonctionnant à vitesse maximale est exactement l'ensemble de toutes les séquences valides au plus tôt du système de tâches modélisé. Une séquence d'ordonnancement au plus tôt est une séquence dans laquelle on alloue à chaque instant le processeur à une tâche, sauf si aucune tâche n'est active. Chaque unité de temps du corps d'une tâche est représentée par une place et une transition 1-1, toutes les places-transitions d'une tâche sont mises en séquence. Ensuite, les accès aux ressources sont ajoutés de manière classique, et les communications sont modélisées par boîtes aux lettres. Afin de réduire la taille du réseau de Petri, une séquence de plus de trois transition 1-1 et autant de places est représentée de façon équivalente par une séquence de trois places et trois transitions. C'est la cas sur la figure 4 pour la tâche τ_0 : trois places et trois transitions remplacent six places et six transitions.

4.2 Importance des temps creux

Rappelons que nous nous intéressons à l'ensemble de toutes les séquences d'ordonnancement du système de tâches modélisé. En effet, certains systèmes de tâches nécessitent à certains moments de choisir de ne rien faire alors que des tâches sont actives. Par exemple, le système $S=\{\tau_1=<0,2,4,4>,\tau_2=<0,1,1,5>\}$, avec τ_1 et τ_2 qui partagent la même ressource critique pendant toute leur exécution, ne possède aucune séquence d'ordonnancement valide au plus tôt, alors qu'elle en possède 54 si l'on autorise des temps creux à avoir lieu alors même que la tâche τ_1 est active.

Il est donc indispensable que le modèle produise aussi des séquences d'ordonnancement qui ne sont pas au plus tôt. Une tâche oisive, nommée τ_0, qui modélise l'occurrence de temps creux dans les séquences d'ordonnancement est ajoutée au système. Cette tâche a une période et une échéance égales à la métapériode H du système, une durée d'exécution égale à $H\times(1-U)$, U étant la charge du système de tâches. La tâche oisive τ_0 est donc caractérisée par $<0,H\times(1-U), H, H>$, et les systèmes augmentés de la tâche oisive ont une charge de 100%.

Sur la figure 4, τ_0 est la tâche oisive associée au système $S'=\{ \tau_1 <0,2,4,4>, \tau_2 <0,1,1,5>\}$. Sans l'adjonction de cette tâche, le système S' n'est pas ordonnançable par le réseau de Petri car S' n'est pas ordonnançable au plus tôt. En effet pour les 54 ordonnancements possibles de ce système, lors de son premier réveil, τ_1 doit être prioritaire sur τ_0 alors qu'à son second réveil, c'est τ_0 qui doit être prioritaire sur elle : la tâche τ_1 doit être mise en attente bien qu'elle soit la seule tâche active pour permettre à la tâche τ_2 de respecter son échéance au temps 6. En effet l'utilisation de la ressource R rend ces tâches non préemptives l'une par l'autre. Ceci est en contradiction avec le fonctionnement au plus tôt des algorithmes classiques. S' n'est donc ordonnançable par aucun algorithme classique.

4.3 Obtention de toutes les séquences d'ordonnancement

L'ensemble des séquences d'ordonnancement est donné par la construction du graphe des marquages du réseau de Petri. Lorsque toutes les tâches sont activées au même instant, la profondeur du graphe des marquages est donnée par la métapériode H du système modélisé. En effet, à l'instant 0, toutes les tâches sont activées. A l'instant H, elles sont toutes réactivées et doivent avoir terminé leur exécution précédente. L'état du réseau de Petri est donc le même à l'instant H qu'à l'instant 0. Il suffit donc d'étudier les ordonnancements sur une période de H unités de temps, puisque l'état du système boucle sur cette période.

4.3.1 Quelques indications sur la complexité

Dans [9], il est montré que le nombre de sommets du graphe des marquages du réseau de Petri est borné par $\left(1+\prod_{i=1}^{n} Ti\right)^{n+1}$. Il y a, évidemment, une borne très large et exponentielle à la taille du graphe des marquages. Cependant, à cause des synchronisations et des exclusions mutuelles, la taille des graphes diminue. De plus, les échéances et les périodicités des tâches font que le graphe forme souvent une *grappe* autour de la diagonale de l'hyperpavé, allant du noeud initial au noeud de final.

Pour exemple nous donnons en figure 5a (resp. 5b) le graphe des ordonnancements du système de tâches S={t1<0,6,14,14>, τ_2<0,4,7,7>} (resp. {t1<0,9,21,21>, τ_2<0,4,7,7>}). Bien que très artificiel, pour entraîner une bonne lisibilité du graphe, cet exemple est typique des ordonnancements.

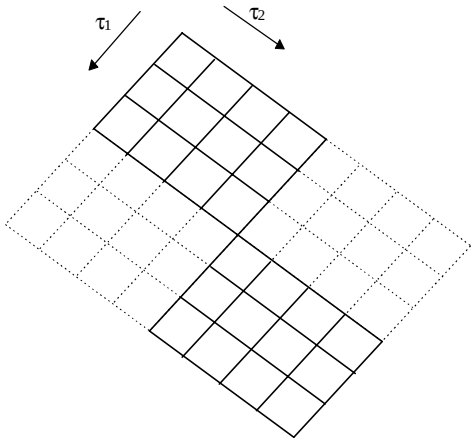

Figure 5a : Un graphe d'ordonnancement, la moitié de l'hyperpavé est utilisé

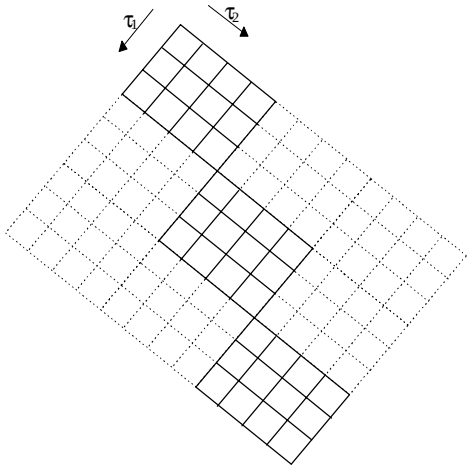

Figure 5b : Exemple de graphe d'ordonnancement, 1/3 de l'hyperpavé est utilisé

4.3.2 Optimisations lors de la construction

L'algorithme basique de construction du graphe des marquages d'un réseau de Petri à contraintes terminales développe chaque séquence d'ordonnancement possible jusqu'à ce que :

(i) arrivé à la fin de la simulation (i.e. au temps H) il valide la séquence

ou

(ii) arrivé à un marquage non terminal M' (dépassement d'une échéance) il déclare la séquence non valide. Dans ce cas, la construction de la séquence est avortée, le noeud père de M', s'il ne fait partie d'aucune séquence valide, est supprimé, et ainsi de suite.

ou

(iii) blocage du réseau de Petri. Même façon de procéder qu'en *(ii)*.

La taille du graphe des marquages du réseau de Petri est exponentielle en nombre de noeuds, et sa construction entraîne du retour arrière dans les cas *(ii)* et *(iii)*.

Pour limiter l'impact de ce retour arrière sur la taille du graphe et sur le temps perdu à explorer des cas, nous utilisons conjointement au réseau de Petri une table dans laquelle des informations permettant de dépister au plus tôt le fait qu'un noeud ne puisse mener qu'à des marquages non valides. Cette table contient pour chaque tâche le nombre d'unités de temps qu'elle requière pour se terminer, ainsi que sa latence (temps restant avant échéance). Cette table, présentée en figure 6, est ordonnée suivant les latences croissantes. Les indices t,i de $\tau_{t,i}$ représentent respectivement la date t considérée et la position de $\tau_{t,i}$ dans la table.

Nom	$\tau_{t,0}$	$\tau_{t,1}$	...	$\tau_{t,n}$

Travail restant	Travail $\tau_{t,0}$	Travail $\tau_{t,1}$	...	Travail $\tau_{t,n}$
Latence	Latence $\tau_{t,0}$	Latence $\tau_{t,1}$	...	Latence $\tau_{t,n}$

Figure 6 : Table exprimant une condition nécessaire d'ordonnançabilité

La condition nécessaire exprime la propriété suivante : lorsque l'échéance d'une tâche $\tau_{t,i}$ expirera, les échéances des tâches $\tau_{t,j}$ telles que $j \leq i$ auront également expiré. Donc les tâches $\tau_{t,0}$, $\tau_{t,1}$, ..., $\tau_{t,i}$ doivent pouvoir terminer leur exécution avant que l'échéance de $\tau_{t,i}$ ne tombe, et si il existe i tel que :

$$\sum_{j=0}^{i} \text{Travail } \tau_{t,j} \geq \text{Latence } \tau_{t,i}$$

alors une des tâches de $\{\tau_{t,0}, \tau_{t,1}, ..., \tau_{t,i}\}$ ne pourra pas respecter sa prochaine échéance. Ce qui est intéressant, c'est que ce test, ainsi que la mise à jour de la table ont une complexité linéaire en fonction du nombre de tâches [9].

Pour limiter la taille du graphe des marquages, en diminuant les entrelacements entre les tâches qui se préemptent, des contraintes de successeur [9] peuvent être imposées lors de la construction des graphes des marquages. Ces contraintes traduisent l'interdiction à toute tâche d'interrompre la tâche travaillant sauf dans les cas suivants :

1.Une tâche qui vient d'être (ré)activée peut préempter la tâche travaillant

2.Une tâche qui vient d'être débloquée par la réception d'un message ou la libération d'une ressource peut préempter la tâche travaillant

3.Si la tâche travaillant demande son entrée en section critique, toute tâche peut la préempter.

L'ensemble des ordonnancements du système représenté figure 5a est ainsi fortement réduit. Il est donné sur la figure 7.

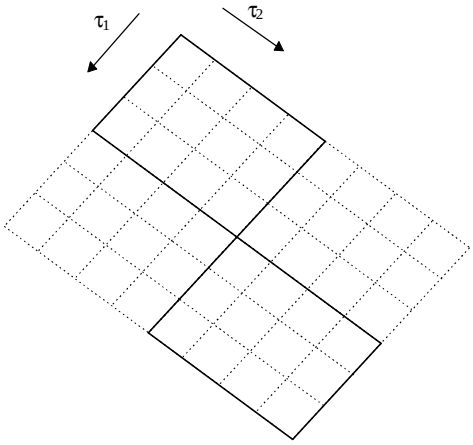

Figure 7 : Exemple de graphe des marquages soumis aux contraintes de successeur

4.3.3 Contraintes absolues sur les séquences

Pour diminuer de façon plus significative la taille du graphe des marquages, des contraintes absolues, portant sur le comportement du système, peuvent être utilisées. Chaque contrainte absolue est exprimée sous la forme d'une équation mathématique, dont les opérantes sont :

- soit fonction d'une tâche considérée
- soit fonction d'une tâche et du noeud sur lequel l'équation est évaluée.

Par exemple, la fonction RT, qui renvoie le temps de réponse d'une tâche, est bien entendu dépendante de la séquence et de l'endroit de la séquence sur laquelle elle est évaluée.

Les fonctions dépendantes du noeud du graphe sur lequel on les évalue sont les suivantes :

- RT : nom_de_tâche×noeud→entier, est le temps de réponse de τ_i dans sa période active sur le noeud.
- L : nom_de_tâche ×noeud →entier, est le temps de retard admissible $L(\tau_i,N)=(R(\tau_i)-RT(\tau_i,N))$ dans sa période active sur le noeud.

•RR : nom_de_tâche ×noeud →entier, est le taux de réaction de τ_i donné par $RR(\tau_i,N)=RT(\tau_i,N)/R(\tau_i)$ dans sa période active sur le noeud.

Les autres fonctions sont les suivantes :
•C : nom_de_tâche →entier, est la charge de τ_i.
•r : nom_de_tâche →entier, est la date de première activation de τ_i.
•R : nom_de_tâche →entier, est le délai critique de τ_i.
•T : nom_de_tâche →entier, est la période de τ_i.

Exemple : $\forall N \in G$, $RT(\tau_1, N)<=1+C(\tau_1)$, avec G l'ensemble des noeuds du graphe, est une contrainte imposant que le temps de réponse de chaque occurrence de la tâche τ_1 ne soit pas plus grand que 1 plus sa charge.

La grammaire de description des contraintes en syntaxe BNF, où les mots clés sont en caractères gras et les commentaires sont donnés entre /* et */, est donnée ci-dessous.

```
contrainte ::= ( contrainte ) | contrainte & contrainte | équation
/* L'opérateur & est l'opérateur logique « et » */
équation ::= entier comparateur entier | rationnel comparateur rationnel
/* Les équations sont soit entières, soit rationnelles */
entier ::= fonction_entière | nombre_entier | (floor) rationnel
          | (ceil) rationnel | entier opérateur entier | ( entier )
/* floor (resp. ceil) est la partie entière supérieur (resp. inférieure)*/
rationnel ::= fonction_rationnelle | nombre_rationnel | (rat) entier
          | rationnel opérateur rationnel | ( rationnel )
/* rat est la fonction de coercion entier → rationnel */
comparateur ::= > | >= | < | <= | == | !=
opérateur ::= + | - | * | /
fonction_entière ::= ∀N.RT ( nom_de_tâche,N ) | ∀N.L ( nom_de_tâche,N )
                   | r ( nom_de_tâche ) | C ( nom_de_tâche )
                   | R ( nom_de_tâche ) | T ( nom_de_tâche )
fonction_rationnelle ::= ∀N.RR (nom_de_tache,N )
nombre_entier ::= [0–9]+
nombre_rationnel ::= [0–9]+.[0–9]+
```

Les contraintes sont évaluées pour chaque noeud construit. Si l'une d'entre elles n'est pas respectée, alors le noeud est déclaré non valide et abandonné.

Si l'ensemble des contraintes ne contient aucune expression dépendante des noeuds, alors le graphe est soit vide (contrainte toujours fausse), soit identique à ce qu'il serait sans contrainte (contrainte toujours vérifiée). Dans ce cas, il est évident que les contraintes ne servent à rien.

Comme précédemment pour limiter les retours en arrière, il est préférable de déterminer au plus tôt la non validité d'un noeud. A l'aide de la table décrite en figure 6, il est possible, à tout moment (i.e. à chaque noeud du graphe des marquages), de déterminer un intervalle dans lequel le prochain temps de réponse d'une tâche correspondant au noeud courant se trouvera. Sa borne supérieure est donnée par son délai critique, et sa borne inférieure est calculée de la façon suivante :

```
FONCTION TempsDeRéponseMinimal(T:Table, N:Noeud, τi:nom_de_tâche)
                                                    RENVOIE entier
  Min: entier
  τj: nom_de_tâche
DÉBUT
  Min←Travail_Restant(T,τi)
  τj←Premier_Nom_de_Tâche(T)
  TANT QUE Min+Travail_Restant(T,τj) > Latence(T,τj) FAIRE
    Min←Min+Travail_Restant(T,τj)
```

```
    τj←Nom_De_Tâche_Suivant(T,τj)
  FAIT
  RENVOYER Min+Temps_Déjà_Ecoulé(N,τi)
FIN
```

T est la table présentée figure 6, associée au noeud courant N. Les fonctions prenant T en paramètre permettent d'accéder au contenu de T. La boucle se termine obligatoirement car la table T satisfait la condition nécessaire d'ordonnançabilité donnée en 4.3.2. L'algorithme se base sur les propriétés suivantes :

Au début, Min reçoit la charge restant à traiter pour τ_i, cela correspond au cas où τ_i est exécutée à partir du noeud N sans être préemptée jusqu'à sa terminaison. Puis on parcourt la table dans l'ordre croissant des latences. Considérons $\tau_{t,0}$ la tâche de latence minimale. Si c'est la tâche τ_i, alors puisque la table respecte la condition nécessaire (§ 4.3.2), la tâche τ_i peut être, hors réveil d'autres tâches, exécutée immédiatement, et son temps de réponse minimal est donc sa charge restant à traiter plus le temps déjà écoulé depuis son réveil. Si $\tau_{t,0}$ n'est pas τ_i, alors si les deux tâches peuvent être exécutées avant l'échéance de $\tau_{t,0}$, τ_i peut être exécutée immédiatement, sinon elle devra attendre la terminaison de $\tau_{t,0}$. Le même raisonnement s'applique pour toutes les tâches de la table jusqu'à ce que les latences soit suffisantes pour permettre d'exécuter τ_i, ce qui sera le cas au pire à la position de la tâche τ_i.

La complexité du calcul de l'intervalle possible du temps de réponse d'une tâche en un noeud N est linéaire en fonction du nombre de tâches, ce qui donne la complexité du calcul des fonctions dépendantes du noeud. Toutes les autres fonctions son évaluées en temps constant.

Ces contraintes permettent de limiter la taille du graphe des marquages, tout en coupant les mauvaises branches le plus tôt possible. Elles permettent au concepteur d'un système temps réel d'exprimer des contraintes avant construction du graphe, le plus souvent sur le plus petit temps de réponse possible, mais elles peuvent permettre de minimiser une gigue (voir [8] pour une définition de la gigue).

Il reste maintenant à exprimer des contraintes qualitatives. Celles-ci sont appliquées après la construction du graphe des marquages du réseau de Petri.

5. Extraction de séquences optimales

Etant donné un graphe des marquages de taille $\aleph$, et de profondeur P, le nombre de séquences possibles peut être exponentiel en $\aleph$. Nous montrons ici une méthode d'extraction d'une séquence d'ordonnancement en fonction de différents critères (importance, minimisation du temps de réponse, maximisation du temps de retard admissible, de certaines tâches ou ensembles de tâches). L'ensemble des séquences extraites par cette méthode contient toutes les séquences optimales vis à vis des critères choisis. Chaque séquence ϕ de cet ensemble est de longueur H. Comme le système est dans le même état au début qu'à la fin de la séquence ϕ, ϕ^* (ϕ répétée à l'infini) est le meilleur ordonnancement pour les critères choisis.

5.1 Critères d'optimisation

Nous présentons ici la liste des critères que nous avons étudiés. Soit E un ensemble de tâches d'un système de tâches S :

Maximisation de l'importance : les tâches de E sont exécutées le plus tôt possible.

Minimisation du temps de réponse maximal (ou moyen) : le temps maximal (ou moyen) entre l'activation et la terminaison des tâches de E est minimisé.

Maximisation du temps de retard admissible minimal (ou moyen) : le temps minimal (ou moyen) entre la terminaison et l'échéance des tâches de E est maximisé.

Minimisation du taux de réaction maximal (ou moyen) : le temps maximal (ou moyen) du temps de réponse divisé par l'échéance des tâches de E est minimisé.

5.2 Algorithmes de choix de séquence

La méthode consiste à valuer le graphe à l'aide d'une fonction de valuation qui dépend du critère à optimiser, puis à y effectuer une recherche des plus courts chemins.

Par exemple, pour maximiser l'importance d'un ensemble de tâches, les arcs du graphe sont valués implicitement de la façon suivante : le coût d'un arc situé entre la profondeur n-1 et la profondeur n est n si il est étiqueté par une tâche importante et 0 sinon. Cette valuation est effectuée en temps constant pour chaque arc. Les séquences optimales sont données par l'ensemble des plus courts chemins allant du noeud initial au noeud terminal.

La méthode utilisée est appliquée en deux phases.

1. Chaque noeud est pondéré de telle sorte que sa pondération donne le coût minimal des chemins allant de ce noeud au noeud terminal. Cette opération est effectuée à l'aide d'un parcours topologique allant du noeud final au noeud initial. Elle a une complexité de l'ordre de $\aleph(n+1)$, n étant le nombre de tâches du système considéré, et $\aleph$ le nombre de noeuds.

2. Une fois les noeuds pondérés, l'ensemble des plus courts chemins est donné par un parcours du graphe. Ce parcours part du noeud initial N_0, et ne choisit dans ses fils que les noeuds N_i de pondération minimale. Il procède récursivement de la même façon sur chacun des noeuds N_i choisis.

Les fonctions de pondération des arcs sont détaillées dans [9].

6. Présentation de PeNSMARTS

Un outil, nommé PeNSMARTS pour *Petri Net Scheduling, Modeling & Analyses of Real Time Systems*, met en oeuvre toutes les techniques présentées depuis la section 4. Cet outil a été présenté à ETR97.

A partir d'une abstraction d'un système de tâches donné, il construit automatiquement le modèle de réseau de Petri. L'outil propose d'utiliser ou non les contraintes de successeur, en effet celles-ci peuvent ne pas être avantageuses, en particulier lorsque l'on souhaite minimiser la gigue de certaines tâches. Lorsqu'il construit le graphe des marquages, une barre de progression permet d'annuler la construction à tout instant, pour le cas où le graphe s'avérerait trop long à construire. Si le graphe des marquages a une taille trop importante, l'utilisateur peut à loisir définir des contraintes absolues.

Lorsque le graphe des marquages est construit, l'utilisateur peut le raffiner, en optimisant les critères présentés en section précédente. Ce raffinage du graphe, qui peut se faire en phases successives suivant plusieurs critères à optimiser, est traduit par la suppression logique des noeuds ne faisant pas partie de séquences d'ordonnancement optimales. Ainsi, chaque optimisation peut être annulée, de même que les contraintes absolues qui peuvent être appliquées sur le graphe.

Pour exemple d'application, nous reprenons le système $S=\{\tau_1=<0,2,4,4>,\tau_2=<0,1,1,5>\}$ présenté en section 4.2. Les résultats de l'utilisation de PeNSMARTS sont présentés sur la table suivantes :

	Nombre de séquences	Nombre de noeuds
Construction sans (resp. avec) contraintes de successeur	54 (2)	29 (24)
Minimiser le temps de réponse maximale de τ_1	3	23
Puis Maximiser l'importance de τ_0	1	21
Deux fois annuler	54	29
Appliquer la contrainte $\forall N.RT(\tau_0,N)==R(\tau_0)-2$		

7. Conclusion

Après avoir justifié l'intérêt de l'approche hors-ligne de l'ordonnancement de systèmes de tâches temps réel, nous avons proposé une approche permettant l'ordonnancement optimal d'un système de tâches temps–réel à partir d'un modèle basé sur les réseaux de Petri. Dans une première phase, le graphe d'accessibilité est construit en temps et en espace NP. Conscients de cet inconvénient, nous utilisons des techniques permettant de limiter la taille du graphe ainsi que les retours arrières inhérents à sa construction : contraintes de successeur, table de prévision, contraintes absolues. Puis nous avons montré qu'un algorithme du type recherche de plus court chemin pouvait être mis en place sur ce graphe pour extraire les séquences optimales au vu de certains critères en un temps linéaire du nombre de noeuds plus le nombre d'arcs du graphe d'accessibilité.

Les critères que nous avons choisis d'optimiser sont la minimisation du temps de réponse, la minimisation du taux de réaction, la maximisation de la latence , tous trois en moyenne ou au pire d'un ensemble de tâches, ainsi que l'importance d'un ensemble de tâches par rapport aux autres.

Les techniques présentées ont été mises en oeuvre dans un outil d'aide au choix de séquence d'ordonnancement nommé PeNSMARTS.

8. Bibliographie

1. R. Alur, T.A. Henzinger, « Real Time Logics: Complexity and Expressiveness », 5th annual IEEE Symposium on Logic in Computer Science, 1990, pp 390-401.
2. T.P. Baker, « Stack-Based Scheduling of Realtime Processes », Journal of Real-Time systems, 3, 1991.
3. J.P. Beauvais, A.M. Déplanche « Affectation de Tâches dans un Système Temps Réel Réparti », TSI, vol. 17, n°1, janvier 1998, Hermes, pp 87-106.
4. M.Chen, K. Lin, « Dynamic Priority Ceilings : A Control Protocol for Real-Time Systems », Journal of Real-Time Systems, 2, 1990.
5. H. Chetto, M. Silly, T. Bouchentouf, « Dynamic scheduling of Real-Time Tasks Under Precedence Constraints », Journal of Real-Time Systems, 2, 1990.
6. A. Choquet–Geniet, D. Geniet, F. Cottet, « 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, 11–13/09/96.
7. A. Choquet-Geniet, G. Vidal-Naquet, « Réseaux de Petri et systèmes Parallèles », Editions Armand Colin, 1993.
8. F. Cottet, M. Courtes, M. Holle, « Traitement de la Gigue Temporelle pour les Ordonnancements Temps Réel par echéance », actes de conférence RTS'98, 14-16/01/1998, Paris.
9. E. Grolleau, A. Choquet-Geniet, F. Cottet, « Ordonnancement Optimal de systèmes de Tâches Temps Réel à l'Aide de réseaux de Petri », actes de conférence AGIS'97, 9-11/12/97, Angers.
10. J.Y.T. Leung, M.L. Merill, « A Note on Preemptive Scheduling of Periodic Real–Time Tasks », Information Processing Letters 11 (3), pp 115–118, 1980.
11. C.L. Liu, J.W. Layland, « Scheduling Algorithms for Multiprogramming in a Hard Real–Time Environment », journal of the ACM, 20 (1), pp 46–61, 1973.
12. A.K. Mok, « 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.
13. L. Sha, R. Rajkumar, J.P. Lehoczky, « Priority Inheritance Protocols : An Approach to Real–Time Synchronisation », IEEE Transactions on Computers, Vol. 39, No. 9, pp. 1175–1185, September 1990.
14. M. Spuri, J.A. Stankovic, « How to Integrate Precedence Constraints and Shared Resources in Real-Time Scheduling », IEEE Transactions on Computers, vol. 43, n°12, pp1407-1412, 1994.
15. J.A. Stankovic, « Misconception about Real–Time Computing », IEEE Computer Magazine, 21 (10), 1pp 0–19, 1988.
16. J.A. Stankovic, M. Spuri, M. Di Natale, G. Buttazzo, « Implications of Classical Scheduling Results for Real-Time Systems », IEEE Computer vol.28 n°6, pp 1-25, 1995