ANALYSE HORS LIGNE D'APPLICATIONS TEMPS REEL FORTEMENT COUPLEES COMPORTANT DES TACHES A DUREES VARIABLES

Stéphane Pailler,
Doctorant en Informatique
stephane.pailler@ensma.fr, + 33 5 49 49 83 36

Annie Choquet-Geniet,
Maître de conférences HDR en Informatique
ageniet@ensma.fr + 33 5 49 49 80 68

Adresse professionnelle
LISI - ENSMA ★ Téléport 2 – 1 Avenue Clément Ader
BP 40109 ★ F-86961 Futuroscope Cedex

Résumé : Nous proposons une méthodologie d'ordonnancement hors ligne d'applications temps réel fortement couplées. Nous mettons en évidence l'importance des variations de la durée d'exécution des tâches qui peuvent mettre en péril la sûreté de fonctionnement du procédé contrôlé. Nous proposons alors une méthode pour tenir compte de ces variations en intégrant explicitement dans le processus de validation du système les instructions conditionnelles du code des tâches. Après avoir adapté le modèle temporel de tâches à ce contexte, nous modélisons ces applications à l'aide de réseaux de Petri autonomes fonctionnant sous la règle de tir maximal et munis d'ensemble terminaux. Nous définissons deux concepts d'ordonnançabilité : l'ordonnançabilité locale et l'ordonnançabilité globale d'où découle la notion de graphe d'ordonnancement. Puis, nous montrons comment extraire ces graphes d'ordonnancement à partir du graphe d'ordonnançabilité globale.

Summary : We propose an off-line scheduling methodology for highly coupled real time applications. We show that variations in the computation time of tasks may hazard the safeness of the controlled process. Thus, we have chosen to take conditional instructions of tasks' code explicitly into account, in order to reduce the potential failures. After the adaptation of the task's temporal model to this context, we model these applications using autonomous Petri nets which run under the earliest firing rule with terminal marking set. We define two concepts of schedulability: the local schedulability and the global one and we define the concept of scheduling graph. Finally, we show how to obtain a scheduling graph from the graph of global schedulability.

Mots clés : Ordonnancement hors ligne, réseau de Petri, durées variables, graphe d'ordonnancement, instructions conditionnelles.

Analyse hors ligne d'applications temps réel fortement couplées comportant des tâches à durées variables

Un nombre sans cesse croissant de procédés autonomes ou assistés (système embarqué des voitures, centrale nucléaire, aéronautique, sonde spatiale...) sont contrôlés par des systèmes temps réel interagissant avec leur environnement. Pourtant, ces systèmes ne sont pas dépourvus d'erreurs qui peuvent mettre en danger des vies humaines ou occasionner une perte d'une valeur économique conséquente. C'est pourquoi, il est nécessaire de développer des méthodes permettant de minimiser ces risques en validant les applications temps réel qui contrôlent ces procédés critiques.

Dans la plupart des cas, ces erreurs proviennent d'un non-respect des contraintes temporelles. En effet, une application temps réel se définit comme une application multi-tâches, chacune d'elle étant assujettie à des contraintes temporelles inhérentes au procédé contrôlé. Par conséquent, le problème majeur consiste en un choix d'une politique d'ordonnancement permettant de distribuer au(x) processeur(s) les instances des tâches de façon à respecter chacune des contraintes temporelles.
Il existe deux méthodes pour y parvenir: la première consiste en l'implantation d'un algorithme d'ordonnancement qui choisira à chaque instant la tâche qu'il est nécessaire d'exécuter, c'est l'ordonnancement en ligne. La seconde consiste en une analyse préalable de l'application et des séquences d'ordonnancement possibles sur un intervalle de temps approprié: c'est l'ordonnancement hors ligne. L'analyse d'ordonnançabilité s'appuie sur le modèle temporel de tâche [LIU,1973],[STANKOVIC, 1998] permettant de caractériser chaque tâche à l'aide des paramètres temporels suivants: R_i, la date de première activation de la tâche i, D_i son échéance relative de fin d'exécution, T_i sa périodicité d'activation, C_i sa durée d'exécution calculée au pire cas sur l'architecture cible. Nous adopterons par la suite, la notation $<R_i, C_i, D_i, T_i>$ pour définir une tâche τ_i.

Beaucoup de travaux ont permis la réalisation d'outils de validation dans des contextes d'ordonnancement hétérogènes: contraintes temporelles strictes ou souples, applications fortement couplées (présence de primitives temps réel comme le partage de ressources et les communications), réduction du temps de réponse.... Pourtant, rares sont les outils effectivement utilisés dans le monde industriel. En effet, les variations de la durée d'exécution d'un tâche dues principalement à la structure du code de la tâche mais également au non déterministe du processeur qui l'exécute (en raison des pipelines, caches, exécutions spéculatives etc.), remet en cause bien trop souvent la pertinence des outils d'analyse d'ordonnançabilité classique. La durée d'exécution C_i habituellement prise en compte dans le modèle temporel provient du calcul d'une borne supérieure de durée WCET (Worst Case Execution Time) qui doit répondre à deux principaux critères: la fiabilité, ce qui certifie qu'aucune exécution de la tâche considérée ne sera de durée supérieure au WCET, et la précision qui permet réduire sa surestimation [PUSCHNER, 2000], [PUSCHNER, 1989], [HEALY, 1998]. Malgré cela, dans un contexte avec primitives temps réel, il peut arriver que l'exécution d'une tâche soit très inférieure au WCET, ce qui peut conduire à une faute temporelle. L'exemple de la figure 1 illustre ce type d'anomalie mise en évidence par [Graham, 1969].

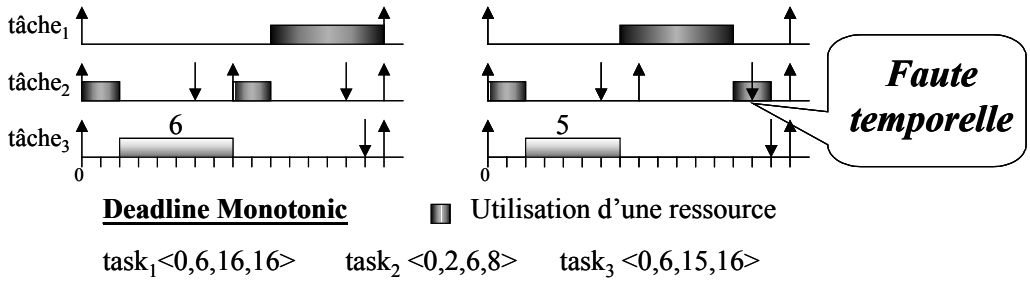

Figure 1 : Exemple d'une anomalie d'ordonnancement en raison d'une réduction de la durée d'exécution de la tâche τ_3.

De plus, le fait de considérer uniquement le chemin d'exécution de durée maximale, oblige également à prendre en compte toutes les primitives temps réel présentes dans chacun des comportements de la tâche. Par conséquent, les tests d'ordonnançabilité sont effectués sur un modèle sur-contraint.
Pour palier ce problème, nous suggérons une méthode qui étend le modèle temporel de Liu Layland pour permettre d'isoler chaque chemin d'exécution des tâches, chacun d'eux pouvant comporter des primitives temps réels. Puis, nous utilisons une modélisation par réseaux de Petri pour analyser l'ordonnançabilité de ces applications fortement couplées.

Dans une première partie, nous introduisons un nouveau modèle temporel et nous redéfinissons le problème d'ordonnancement associé à cette extension. Puis dans une seconde partie nous réutilisons la modélisation par réseaux de Petri autonomes, fonctionnant sous la règle de tir maximal et ensembles terminaux de [CHOQUET-GENIET, 1996], [GROLLEAU, 2000]. Enfin, dans une dernière partie, nous montrons comment extraire un graphe d'ordonnancement décrivant un comportement valide de l'application à partir du graphe des marquages.

1 – EXTENSION DU MODELE DE TACHE

Nous considérons des applications fortement couplées composées de tâches périodiques à départ simultané (les dates de réveil R_i sont toutes égale à 0). Notre objectif consiste à isoler les différents chemins d'exécution possibles de chaque tâche. Pour cela, nous prenons en compte explicitement les instructions conditionnelles (IF THEN ELSE) du code des tâches et associons à chaque comportement induit une pire durée d'exécution. Nous pouvons alors étendre le modèle de Liu Layland en considérant non pas une durée d'exécution C_i mais un *multi-ensemble* de durée E_i pour chaque tâche. Ce multi-ensemble s'apparente au modèle multiframe de [MOK, 96] sans toutefois la connaissance préalable de la durée d'exécution de la tâche pour chaque instance successive. Nous proposons ici une méthode permettant de tenir compte de toutes ces durées pour chaque instance.
On peut noter que dans le cas où une tâche ne possède pas d'instruction conditionnelle, ce multi-ensemble est réduit à une unique durée qui correspond au modèle temporel classique.

Comme nous nous sommes placés dans le cadre d'une étude hors ligne, nous devons tenir compte de l'indécision de chaque test conditionnel pour l'analyse d'ordonnançabilité. Pour cela nous introduisons la notion de *graphe d'ordonnancement*: un graphe d'ordonnancement est un graphe où chaque branche correspond à un ordonnancement de l'application en ne considérant pour chaque instance de tâche qu'un seul chemin d'exécution. Nous devons reformuler le problème d'ordonnancement d'une application comportant des durées d'exécution variables, en définissant les deux concepts suivants:

- une application est dite *localement ordonnançable* si tous les chemins d'exécution de chaque tâche sont indépendamment ordonnançables.
- une application est dite globalement ordonnançable s'il existe au moins un graphe d'ordonnancement valide. On constate dans ce cas, qu'à chaque instruction conditionnelle d'une tâche à un instant t, il existe une branche d'ordonnancement valide à partir de t, quel que soit le résultat du test conditionnel.

Nous pouvons noter ici que l'ordonnançabilité globale implique l'ordonnançabilité locale mais l'inverse est vrai uniquement dans un contexte sans primitives temps réel. Pour notre méthodologie d'analyse d'ordonnançabilité hors ligne, nous devons donc considérer uniquement l'ordonnançabilité globale qui permet de s'abstraire de l'incertitude des tests conditionnels. Nous cherchons donc à obtenir un graphe d'ordonnancement.

2 MODELISATION PAR RESEAU DE PETRI

Le méthode d'analyse d'ordonnançabilité que nous proposons reprend celle présentée dans [GROLLEAU, 2000], [CHOQUET-GENIET, 1996], [GROLLEAU, 1999]. Elle consiste en une modélisation d'une application par réseau de Petri coloré fonctionnant avec la règle de tir maximal. On peut ainsi obtenir tous les ordonnancements valides grâce à la construction du graphe des marquages.

Cette modélisation se décompose en deux parties:

- Un système de tâche qui est construit de façon classique à partir des blocs fonctionnels du code des tâches. On peut noter qu'une transition correspond à une action de durée unitaire et que chacune d'elles est en concurrence pour obtenir le jeton de la place processeur, nécessaire à leur franchissement. Par conséquent à chaque temps, une unique transition du système de tâches est franchie.
- Un système d'horlogerie discrétisé basé sur [FOHLER, 1994], [KOPETZ, 1992] qui comprend une horloge externe (RTC) qui capte le temps dans chaque place $Time_i$, cette dernière agissant comme une horloge locale pour réactiver de façon périodique la tache i correspondante.

La figure 2 résume les différentes caractéristiques de cette modélisation. La règle de tir maximal impose au réseau de Petri le franchissement à chaque unité de temps, d'une transition du système de tâches (sauf si la charge total est nulle). Il apparaît immédiatement que seules les séquences d'ordonnancement conservatives peuvent être produites. Or, pour augmenter les capacités d'ordonnancement de notre modèle, nous devons également considérer les séquences non conservatives. Pour cela, nous introduisons une tâche supplémentaire nommée *tâche oisive (τ_0)* qui simulera l'inactivité du processeur. Ainsi, à chaque franchissement d'une transition de la tâche oisive, le processeur reste inactif: les séquences d'ordonnancement non conservatives sont ainsi produites.

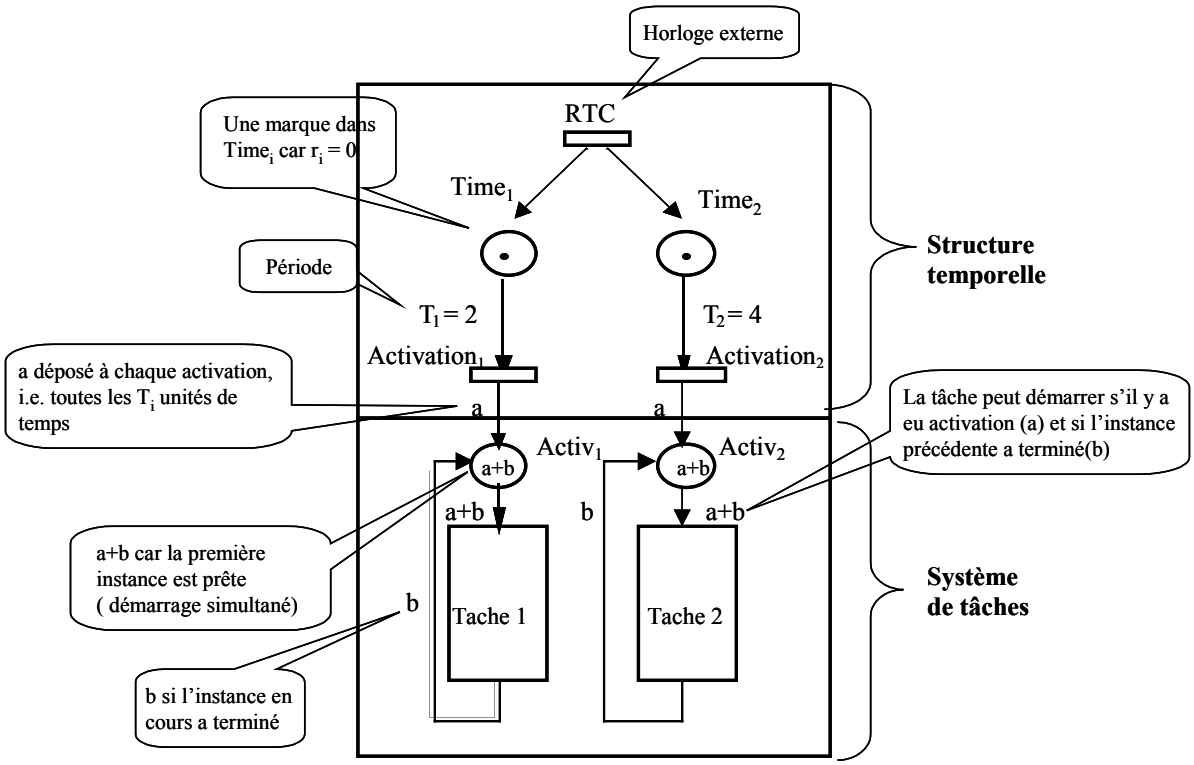

Figure 2 : Modélisation par réseau de Petri pour une application comportant 2 tâches avec respectivement des périodes de T_1=2 et T_2=4. La place processeur n'est pas représentée.

Les paramètres temporels de la tâche oisive dépendent directement du facteur de charge processeur (noté U). Considérons la metapériode P comme le plus petit multiple commun des périodes des tâches alors, puisque les séquences d'ordonnancement sont cycliques et de période P (on rappelle que les tâches démarre à l'instant 0), on peut définir le nombre exacte de temps d'inactivité processeur: *P(1-U)*. On peut donc définir la tâche oisive τ_0 par $<R_i=0, C_i=P(1-U), D_i=T_i=P>$.

Toutefois, l'utilisation du modèle temporel étendu rend le facteur de charge processeur indéterministe puisqu'il dépend des chemins d'exécution empruntés et donc des choix des instructions conditionnelles. Pour calculer la durée de la tâche oisive, nous avons choisi de considérer les chemins d'exécution les plus longs pour chaque tâche, ce qui maximise le facteur de charge processeur et par conséquent réduit la durée de la tâche oisive. Lorsqu'une tâche s'exécute en passant par un chemin de plus courte durée, la durée de la tâche oisive est alors incrémentée de la différence de durées. Pour permettre ce dynamisme de la durée de la tâche oisive, nous avons choisi de la modéliser par: une seule transition en concurrence avec les transitions des autres tâches, pour représenter un temps d'inactivité processeur à chaque franchissement et, une place comportant autant de jetons qu'il doit y avoir de temps oisif sur P. La figure 3 illustre la modélisation d'une tâche oisive.

Nous nous intéressons ici à l'exploitation du graphe des marquages induit par la modélisation par réseau de Petri, et plus particulièrement à l'extraction d'un graphe d'ordonnancement valide. Dans un premier temps, l'exploitation directe du réseau de Petri permet d'obtenir un graphe des marquages de profondeur P (le système est dans le même état à

une profondeur de t et à $t + k \times P$ (k>0) du fait de la cyclicité des séquences d'ordonnancement).

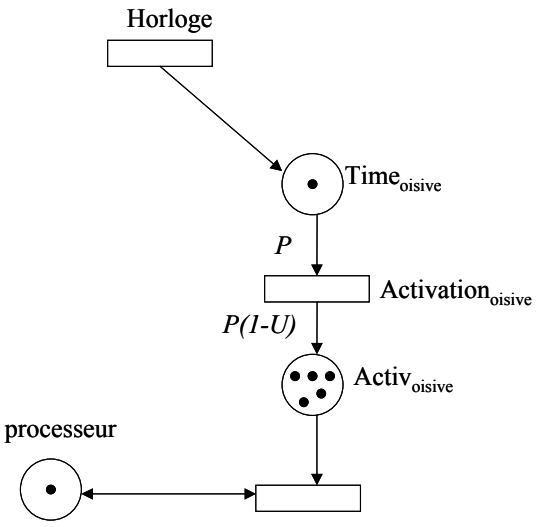

Figure 3 : Exemple d'une modélisation par réseau de Petri d'une tâche oisive

3 - ANALYSE HORS LIGNE DU GRAPHE DES MARQUAGES

Ce graphe des marquages comporte des états sans successeurs, c'est à dire des branches ne permettant pas d'arriver à la hauteur P en raison d'une erreur d'ordonnancement (échéance non respectée). Il est donc nécessaire d'épurer le graphe pour obtenir au final un *graphe d'accessibilité* qui contient toutes les séquences d'ordonnancement valides pour chaque chemin d'exécution des tâches: nous sommes alors dans un contexte d'ordonnancement local. Or, nous nous intéressons à l'ordonnancement global pour extraire un graphe d'ordonnancement certifiant l'existence d'un chemin d'exécution quelque soit le résultat des instructions conditionnelles. Nous devons donc une nouvelle fois épurer le graphe d'accessibilité pour supprimer toutes les branches d'ordonnancement issues d'un test conditionnel à une hauteur h dans le graphe qui ne permettent pas d'emprunter une succession d'états jusqu'à la hauteur P correspondant à l'alternative de ce test de hauteur h. A chaque instant où un choix est fait sur le chemin d'exécution à emprunter, le chemin correspondant à l'alternative de ce choix doit être possible au même instant. Nous obtenons alors un *graphe d'accessibilité global*.

La plus grande difficulté réside ensuite dans l'extraction d'un graphe d'ordonnancement à partir du graphe d'accessibilité global. En effet, la taille d'un graphe d'ordonnancement est d'autant plus grande qu'il y a de chemins d'exécution possibles pour chaque tâche.

Propriété : Considérons n tâches définies par
$<R_i, E_i, D_i, T_i>$ pour $1 \le i \le n$,
ne comportant pas de primitives temps réel :

- Le nombre de comportements possibles pour une tâche τ_i sur la metapériode P est

$$\langle E_i \rangle^{\frac{P}{T_i}},$$

- Le nombre de comportements possibles dans un graphe d'ordonnancement est

$$\prod_{1}^{n} \langle E_i \rangle^{\frac{P}{T_i}},$$

$\langle \cdot \rangle$ représentant le nombre d'éléments de E_i.

Le graphe d'accessibilité globale est composé de l'ensemble des graphes d'ordonnancement possibles, on se rend donc facilement compte de l'explosion combinatoire rencontrée lors de l'énumération des graphes d'ordonnancement valides. Malgré la présence de primitives temps réel qui diminue fortement le nombre de graphes d'ordonnancement dans le graphe d'accessibilité global, il est de toute évidence nécessaire d'éviter sa construction dans son ensemble. C'est pourquoi, nous adoptons des techniques d'optimisation par contraintes [GROLLEAU, 1999] permettant de réduire la construction du graphe d'accessibilité globale pour la recherche d'un arbre d'ordonnancement. On peut citer les contraintes de successeurs qui empêche la préemption des blocs indépendants des tâches, ainsi que des contraintes *a priori* qui permettent de ne pas construire des branches d'ordonnancement qui ne vérifient pas certains critères de qualités d'ordonnancement comme un seuil maximum de temps de réponse ou de gigue. Toutefois, ce dernier type de contrainte nécessite d'être redéfini pour s'appliquer non pas à une séquence mais à un graphe d'ordonnancement. En effet, en considérant une contrainte de temps de réponse pour une tâche qui comprend plusieurs chemins d'exécution, le temps de réponse peut s'appliquer soit à l'un des chemins (le plus court ou le plus long par exemple) soit à la moyenne des différents chemins. Le résultat obtenu sera bien évidemment totalement différent. Par défaut nous considérons uniquement le temps réponse maximal de tous les chemins d'exécution de la tâche, mais nous travaillons actuellement sur les différentes possibilités qu'offre ce type d'optimisation sur le choix des graphes d'ordonnancement.

Une autre voie que nous explorons également consiste à s'intéresser plus particulièrement aux placements des temps d'inactivités processeur. En effet, il pourrait être avantageux de les regrouper pour offrir une période de traitement indépendante au processeur, comme par exemple la simulation d'une tâche serveur pour le traitement de tâches apériodiques, ou pour un traitement de fond non critique.

La figure 4 montre un exemple de graphe d'accessibilité globale sur lequel est appliqué une contrainte de successeur, puis un graphe d'ordonnancement extrait.

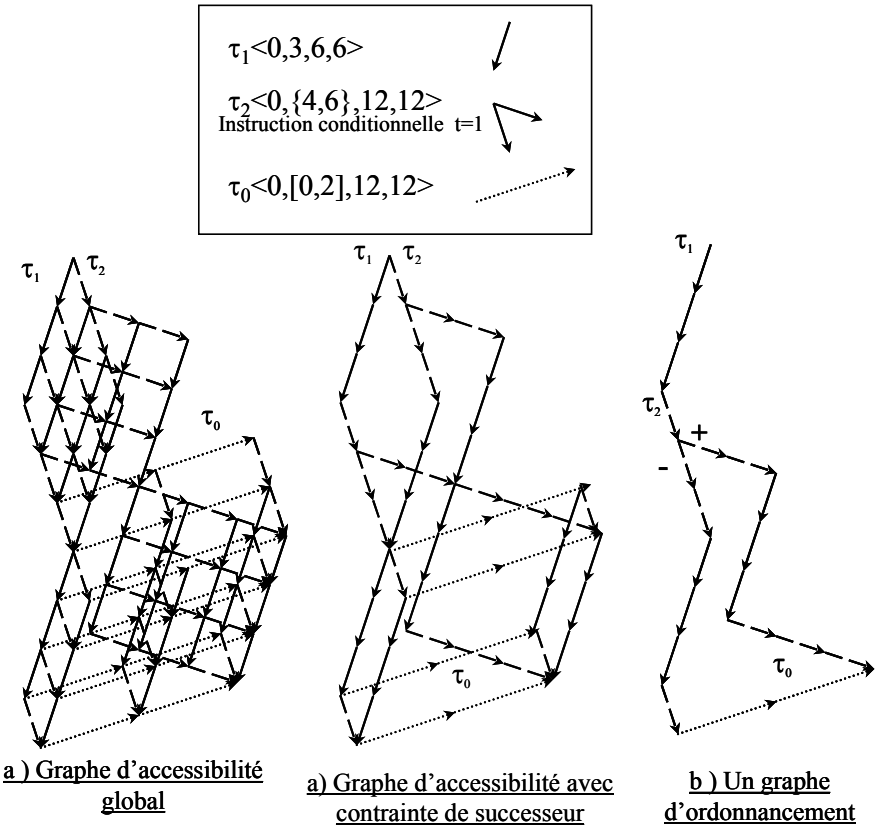

Figure 4 : Le graphe d'accessibilité (a) étant de taille très importante il est nécessaire de lui appliquer des contraintes lors de sa construction, comme des contraintes de successeur(b), pour pouvoir en extraire un graphe d'ordonnancement(c).

4 – CONCLUSION

Nous avons proposé une méthodologie pour l'analyse d'ordonnançabilité d'applications temps réel fortement couplées, qui utilise une modélisation par réseau de Petri. Nous avons montré que les variations de la durée d'exécution doivent être prises en compte pour éviter certains problèmes d'instabilité, malgré les avancées dans le domaine de l'estimation des WCET. L'indéterminisme des processeurs, l'utilisation de compilation optimisée et le mode programmation actuelle augmente le nombre de comportement d'exécution pour chaque tâche et rend impossible la mesure exacte des WCET [PUSCHNER, 2002]. C'est pourquoi nous proposons une méthodologie qui tire avantage des études du WCET en intégrant dans la modélisation par réseaux de Petri, les instructions conditionnelles du code des tâches. Après avoir étendu le modèle temporel de tâche de Liu Layland, nous avons déduit deux concepts d'ordonnançabilité : l'ordonnançabilité locale et globale. Nous avons par la suite introduit la notion de tâche oisive à durée dynamique qui est directement dépendante de la fluctuation des différents chemins d'exécution des tâches. Après avoir montré comment modéliser ce type d'applications par réseaux de Petri, nous en avons extrait un graphe des marquages. Il a été ensuite nécessaire de filtrer ce graphe pour supprimer les séquences d'ordonnancement non valides de façon à ne garder qu'un graphe d'accessibilité globale formé de l'ensemble des graphes d'ordonnancement valides. Nous avons montré par la suite comment en extraire un graphe d'ordonnancement en appliquant des contraintes permettant de réduire la construction du graphe d'accessibilité.

L'étape suivante de cette étude consistera à étendre les possibilités d'extraction des graphes d'ordonnancement tout en optimisant leur recherche.

BIBLIOGRAPHIE

[CHOQUET-GENIET, 1996] A. CHOQUET-GENIET, D. GENIET et F. COTTET, "*Exhaustive computation of the scheduled task execution sequences of a real-time application*". Proc. of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems p. 246–262 (Uppsala, Sweden), 1996.

[FOHLER, 1994] G. FOHLER, *"Flexibility in statically scheduled hard real-time systems*". Thèse, University of Wien, April 1994.

[GROLLEAU, 2000] E. GROLLEAU et A. CHOQUET-GENIET, "*Off line computation of real-time schedules by means of Petri nets*". Workshop On Discrete Event Systems WODES2000, Discrete Event Systems: Analysis and Control, Kluwer Academic Publishers, p. 309–316, Ghent, Belgium, 2000.

[GRAHAM, 1969] R. GRAHAM, "*Bounds on multiprocessing timing anomalies*". SIAM Journal of Applied Mathematics 17, no. 2, p.416-429, 1969.

[GROLLEAU, 1999] E. GROLLEAU, "*Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de Petri en environnement monoprocesseur et multiprocesseur*". Thèse, ENSMA – Université de Poitiers, Novembre 1999.

[HEALY, 1998] C. HEALY, M. SJÖDIN, V. RUSTAGI et D. WHALLEY, "*Bounding loop iterations for timing analysis*". 4th IEEE Real-Time Technology and Applications Symposium (RTAS '98), Discrete Event Systems: Analysis and Control, IEEE, p. 12–21, 1998.

[KOPETZ, 1992] H. KOPETZ, "*Sparse time versus dense time in distributed real-time systems*". 12th Int. Conf. On Distributed Computing Systems, p.460–467, Japon, June 1992.

[LIU, 1973] C. LIU et J. LAYLAND, "*Scheduling algorithms for multiprogramming in real-time environnement*". Journal of the ACM 20(1), p.46–61, (1973).

[MOK, 1996] A. MOK et D. CHEN, " *A multiframe Model for Real Time Tasks*". in Proceedings of the 17th Real-Time System Symposium, pp. 22-- 29, Dec. 1996.

[PUSCHNER, 2002] P. PUSCHNER, "*Is worst-case execution-time analysis a non problem? - towards new software and hardware architectures*". 2nd Intl. Workshop on WCET Analysis 14th Euromicro Conference on Real- Time Systems, June 2002.

[PUSCHNER, 2000] P. PUSCHNER et A. BURNS, "*Guest editorial: A review of worst-case execution-time analysis*". Real-Time Systems 18, p.115–128, 2000.

[PUSCHNER, 1989] P. PUSCHNER et C. KOZA, "*Calculating the maximum execution time of real-time programs*". Journal of Real-Time Systems 1(2) p. 159–176, 1989.

[STANKOVIC, 1998] J. STANKOVIC, M. SPURI, K. RAMAMRITHAM et G. C. BUTTAZZO, "*Deadline scheduling for real-time systems*". vol. ISBN 0 7923 8269 2, 1998.