Étude hors ligne d'une application temps réel à contraintes strictes

Annie Choquet-Geniet — Emmanuel Grolleau — Francis Cottet

LISI- ENSMA
Site du Futuroscope BP 109
86960 Futuroscope Cedex
{ageniet, grolleau, cottet}@ensma.fr

RESUME. *Nous présentons une méthodologie d'étude hors ligne d'applications temps réel, constituées de tâches interagissantes, soumises à des contraintes temporelles strictes. A l'aide d'une modélisation par réseau de Petri avec ensemble terminal, fonctionnant sous la règle de tir maximal, nous calculons l'ensemble de toutes les séquences valides et nous présentons des techniques d'extraction de séquences optimales au vu de certains critères de performances de l'application (temps de réponse...). Notre approche permet de traiter des systèmes de tâches à départs différés, de prendre en compte des ressources de type lecteur-écrivain. De plus, les tâches se synchronisant peuvent avoir des périodes différentes. Ceci constitue la classe la plus large de systèmes temps réel étudiée jusqu'alors dans les approches hors ligne.*

ABSTRACT. *We present a methodology of off line analysis for hard real time applications, constituted of interacting tasks, submitted to hard temporal constraints. By means of a Petri net with terminal markings, under the maximal firing rule, we compute all the feasible schedules and we present extraction techniques of optimal schedules, for some performance criteria (e.g. optimal response time of a subset of tasks..). Our approach takes into account asynchronous task systems, read and write resources. Furthermore, communicating tasks do not have to share the same period provided the emission and reception rates are the same. The considered class is the wider which has been considered for off line approaches.*

MOTS-CLÉS. *ordonnancement – analyse hors ligne – séquences optimales - réseaux de Petri – règle de tir maximal – marquages terminaux*

KEYWORDS. *Scheduling – off line analysis – optimal schedules – Petri nets – maximal firing rule – terminal markings*

Introduction

1.1. *L'ordonnancement temps réel*

La spécificité des applications temps réel provient de ce qu'elles sont soumises à des contraintes temporelles, dues à la criticité de certaines actions qui lui permettent d'interagir avec l'environnement d'un procédé à contrôler [STA 88, ELL 97]. Afin de garantir la sûreté de fonctionnement, exigence première de telles applications, il est indispensable de s'assurer non seulement de sa correction algorithmique, mais aussi de sa correction temporelle, en vertu du principe fondamental qui veut qu'un résultat juste mais fourni hors délai soit assimilable à un résultat erroné.

Dans cette optique, l'application est modélisée par un ensemble de tâches périodiques, apériodiques ou sporadiques [STA 98], pouvant communiquer et partager des ressources, et caractérisées par des paramètres temporels [LIU 73]. Cet ensemble de tâches est obtenu à partir d'une étude (non formelle) du cahier des charges de l'application. Chaque tâche est implémentée à l'aide d'une part des instructions classiques d'un langage de haut niveau, et d'autre part de primitives fournies par un noyau temps réel qui assurent les interactions entre les tâches.

Dans cet article, nous ne prenons en considération que les tâches périodiques, les autres pouvant être prises en compte par des serveurs périodiques ou ordonnancées en arrière plan [BUT 97, DEL 99]. Nous nous intéressons au problème de la validation temporelle de l'application ainsi qu'à la construction effective de séquences temporellement **valides** qui sont les séquences dans lesquelles aucune tâche ne commet de faute temporelle. Nous nous plaçons dans un cadre monoprocesseur, préemptif, et nous considérons uniquement les systèmes déterministes : les paramètres temporels sont connus a priori, puisque ce sont les seuls pour lesquels on puisse garantir le respect des contraintes temporelles. Chaque tâche est caractérisée par une **période**, une **date de premier réveil**, une **charge**, qui représente sa pire durée d'exécution, et un **délai critique**, qui est la durée maximale tolérée entre l'activation de la tâche et sa terminaison. La date à laquelle l'instance en cours doit être terminée est son **échéance**. Si les dates de premier réveil sont toutes les mêmes, on parle de tâches à **départs simultanés**, sinon, on parle de tâches à **départs différés**. Enfin, si le délai critique d'une tâche est égal à sa période, on dit que la tâche est à **échéance sur requête**. La validation temporelle de l'application repose alors sur la possibilité de choisir une politique adéquate d'attribution du processeur. Deux approches peuvent être envisagées, l'ordonnancement **en ligne** : la politique d'ordonnancement est implémentée au sein de l'ordonnanceur, et l'ordonnancement **hors ligne** : cette fois, une séquence prédéfinie est fournie au séquenceur, avec l'avantage d'éviter ainsi le surcoût lié à l'exécution de l'algorithme d'ordonnancement.

Les algorithmes d'ordonnancement utilisés en ligne reposent sur la notion de priorité et fonctionnent **au plus tôt** : le processeur ne peut pas rester inactif s'il y a des tâches prêtes, et il est attribué à la tâche de plus forte priorité parmi les tâches prêtes. Les priorités peuvent être définies arbitrairement par le concepteur, en fonction de contraintes de temps à respecter, ou bien être dérivées des paramètres temporels. Les algorithmes les plus connus de cette catégorie sont : Rate Monotonic, RM, qui attribue la plus forte priorité à la tâche de plus petite période, Earliest Deadline, ED, qui attribue la plus forte priorité à la tâche dont la prochaine échéance

est la plus proche, Least Laxity, LL, qui attribue la plus forte priorité à la tâche dont la date correspondant à l'échéance moins la charge restant à traiter est la plus proche [LIU 73, LEU 80, MOK 78]).

Sous certaines hypothèses, par exemple, pour ED, si les tâches sont indépendantes ou liées par des contraintes de précédence ; ou pour RM, si les tâches sont indépendantes à échéance sur requête et à priorités statiques, certains de ces algorithmes sont **optimaux**, en ce sens que soit ils ordonnancent correctement l'application : aucune faute temporelle n'est commise, soit l'application n'est pas ordonnançable [LIU 73, DER 74, BLA 76].

Ces algorithmes sont de complexité polynomiale, et dans le cas de tâches indépendantes à départs simultanés, on dispose de critères analytiques d'ordonnançabilité ne nécessitant pas la construction effective de séquences d'ordonnancement. Mais, dès lors que les tâches interagissent (contraintes de précédence, utilisation de ressources), ou même si elles sont indépendantes, mais à départs différés et de délai critique plus petit que la période, le problème de l'ordonnancement devient NP-difficile [MOK 83, BAR 90, LEU 80]. De plus, en présence de ressources, il n'existe plus d'algorithme optimal d'ordonnancement et donc le problème de l'ordonnancement ne peut plus être résolu en un temps raisonnable [MOK 83]. Outre le problème classique de possibles interblocages, la difficulté majeure lorsque l'on ordonnance des tâches utilisant des ressources provient du phénomène d'inversion de priorité : une tâche de faible priorité détient une ressource dont a besoin une tâche de plus forte priorité. Survient une tâche de priorité intermédiaire : elle va préempter la tâche de plus faible priorité, et de manière transitive, retarder ainsi la tâche de plus forte priorité. A priori, la durée des inversions de priorité n'est pas bornée, car plusieurs tâches intermédiaires peuvent retarder alternativement une tâche de plus forte priorité. Afin de contrôler l'inversion de priorité, plusieurs protocoles de gestion de ressources ont été développés : protocole à priorité héritée (PHP) [KAI 82, SHA 90], protocole à priorité plafond (PCP) [SHA 90, CHE 90], protocole à priorité de pile (SRP) [BAK 91]. Ces protocoles ont comme vocation de supprimer l'inversion de priorité et de borner la durée maximale de blocage sur attente de ressources par la durée maximale d'une section critique pour les plus puissants d'entre eux. Certains d'entre eux permettent également d'éviter les interblocages. Afin de réaliser l'étude d'ordonnançabilité de l'application, la durée de chaque tâche doit être augmentée de la pire durée de blocage dont elle peut souffrir. Les paramètres peuvent alors cesser d'être réalistes, ce qui signifie que plus il y aura d'utilisations de ressources partagées, moins l'analyse d'ordonnançabilité sera performante. Notons enfin que l'on ne dispose plus que de conditions suffisantes d'ordonnançabilité.

1.2. *L'ordonnancement hors ligne*

Des méthodes d'ordonnancement hors-ligne ont été développées, permettant l'étude d'applications fortement interagissantes. Elles sont en règle générale exponentielles en temps. La plupart de ces méthodes reposent sur des techniques de branch-and-bound ou d'énumération. Elles s'adressent soit à un ensemble spécifique d'instances de tâches, soit à des systèmes à départs simultanés de tâches périodiques. [ZAM 97] traite de systèmes de tâches indépendantes et procède de manière

récurrente. Une fois construites toutes les séquences valides pour un ensemble de k tâches, une (k+1)-ème tâche est ajoutée au système et son placement dans les séquences précédemment construites est étudié. [XUP 90] présente la méthode la plus générale, et traite des systèmes de tâches non périodiques, avec ressources critiques mono instance, et contraintes de précédence. La méthode s'étend aux systèmes de tâches périodiques à départ simultanés, en considérant toutes les instances des tâches entre les instants 0 et PPCM des périodes. La séquence initialement utilisée est celle produite par l'algorithme ED. Ensuite, afin de corriger celle-ci si des dépassements d'échéance apparaissent, les tâches sont découpées et de nouvelles contraintes de précédence ou de préemption sont ajoutées. L'ensemble est itéré jusqu'à obtention d'une séquence valide, ou jusqu'au constat d'échec. Des méthodes arborescentes ont été proposées dans [BAK 74, BRA 75], dans des cadres respectivement mono et multi processeur, dans un contexte non préemptif, pour des tâches indépendantes. Dans les deux cas, il s'agit de produire toutes les séquences valides, en engendrant pas à pas toutes les permutations possibles, et en éliminant celles qui engendrent un dépassement d'échéance. Dans le cas multi processeur, le placement est conjointement envisagé. Des méthodes d'analyse utilisant des réseaux temporisés [RAM 74] ou temporels [MER 76] ont également été développées. Elles ont pour principal objectif d'établir des diagnostics ([TSA 93, MEN 83, BER 91]). Enfin, des méthodes approchées de moindre coût ont été proposées : recuit simulé, algorithme génétique...[BEA 96, BEA 98] envisage conjointement le problème du placement et celui de l'ordonnancement selon LL, dans un cadre multi processeur.

Notre approche s'inscrit dans le cadre de l'étude exacte hors-ligne des applications fortement couplées, à départs simultanés ou différés, traitant aussi bien le problème des ressources exclusives, que les problèmes de type lecteur/écrivain, avec des entrelacements de sections critiques non restreints aux imbrications (une telle hypothèse est utilisées dans les travaux présentant les protocoles de gestion de ressources) et comportant des communications, ce qui est plus général que la prise en compte de précédence entre tâches. Notre objectif est : - de fournir un diagnostic d'ordonnançabilité, y compris (et surtout) dans les cas où les critères analytiques ne s'appliquent pas, et dans les cas où les algorithmes classiques faillissent, ceci parce que nous sommes à même de produire l'ensemble exhaustif des séquences valides ; - de permettre la construction d'une séquence valide, pouvant de plus intégrer des critères non exprimables à l'aide des paramètres temporels, mais permettant de guider le concepteur dans le choix de la séquence qu'il fournira au séquenceur. On peut par exemple choisir les séquences permettant d'obtenir un temps de réponse moyen optimal pour un sous-ensemble de tâches, ou bien minimiser le taux de réaction de certaines tâches... Une autre caractéristique des séquences valides concerne la répartition des temps creux. Celle-ci est particulièrement importante si l'on veut pouvoir ordonnancer des tâches apériodiques en arrière plan, en couplant un ordonnancement hors ligne des tâches périodiques avec un ordonnancement en ligne des tâche apériodiques. Notre méthode permet de les localiser a priori et de choisir un arrangement spécifique souhaité des temps creux, par exemple, une répartition la plus uniforme possible, ou au contraire où les temps creux sont placés le plus tard possible (cas des ordonnancements au plus tôt).

Notons que même si l'un des algorithmes classiques ordonnance l'application, il se peut qu'il ne soit pas le meilleur au vu des critères retenus.

La méthode utilisée est basée sur une modélisation de l'application par réseau de Petri : l'ensemble des tâches est modélisé par un réseau de Petri autonome classique [CHO 93, PET 81], auquel est adjointe une structure temporelle, constituée d'une horloge globale et d'horloges locales aux tâches. Cette structure temporelle est elle aussi modélisée par un réseau Place/Transition. La puissance du réseau ainsi obtenu est augmentée d'une part par un fonctionnement sous la règle de tir maximal [STA 90] qui implémente une horloge logique, et d'autre part par l'adjonction d'un ensemble de marquages terminaux qui permet de prendre en compte les délais critiques des tâches. A partir de ce réseau, nous pouvons construire un graphe des marquages qui représente l'ensemble exhaustif des séquences valides, et nous pouvons extraire les séquences optimales au vu de certains critères de performances de l'application. Nous ne possédons qu'une borne supérieure de la taille du graphe des marquages ainsi obtenu, qui est exponentielle en le nombre des tâches. Mais cette borne est très pessimiste dans la plupart des cas : les contraintes temporelles et les interactions entre tâches restreignent fortement cette taille. Afin de la limiter encore plus, nous utilisons diverses heuristiques, comme par exemple la limitation du nombre de changements de contextes en interdisant les préemptions multiples entre portions de tâches indépendantes.

La suite de cet article est organisée de la façon suivante : en section 2, nous présentons le modèle formel de tâches et nous précisons nos hypothèses. En section 3, nous présentons la modélisation utilisée. En section 4, nous décrivons les étapes successives de notre méthodologie d'analyse d'une application et nous donnons quelques heuristiques permettant de limiter l'explosion combinatoire. En section 5, nous présentons une courte étude de cas.

2. Le modèle de tâches

2.1. *Le modèle temporel*

Nous utilisons le modèle classique [STA 98] de décomposition d'une application temps réel en un ensemble fini $\{\tau_1,\tau_2,\ldots, \tau_n\}$ de tâches périodiques, issues d'une conception préliminaire basée sur l'étude du cahier des charges.

Nous supposons que le système comporte des ressources non préemptibles. Une ressource R possède un nombre N_R d'instances, et une tâche τ peut demander m instances de la ressource R ($1 \leq m \leq N_R$). Nous considérons aussi bien les accès exclusifs que les accès de type lecteur/écrivain. Enfin, les sections critiques peuvent s'entrelacer (l'entrelacement n'étant pas limité à la seule imbrication).

Par ailleurs, les tâches peuvent communiquer entre elles par le bais de primitives d'émission/réception de messages, qui permettront d'exprimer des relations de précédence entre certaines parties du corps des tâches. Notre modèle est plus général que le modèle classiquement utilisé pour les approches en ligne en ce que les émission/réception sont distribuées dans le corps des tâches, et pas seulement localisées en fin/début de tâches. Nous ne supposons donc pas que les tâches sont données sous forme normale [COT 94]. Par contre, nous supposons qu'il n'y a pas d'attente de message à l'intérieur d'une section critique.

Le squelette d'une tâche va donc être constitué :

— de blocs fonctionnels écrits dans un langage de haut niveau

— de primitives temps réel :

– Prise et libération de sémaphores

– Émission / réception de messages

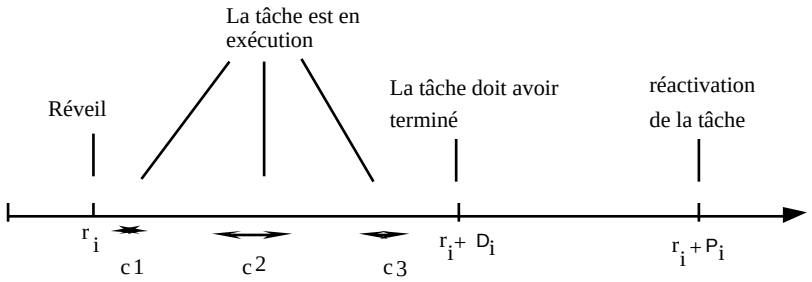
A ce modèle fonctionnel est adjoint un modèle temporel, qui est le modèle classique de Liu-Layland [LIU 73]. Une tâche τ_i est caractérisée par (cf fig. 1) :

– une date de première activation r_i

– une pire durée d'exécution C_i

– un délai critique relatif D_i

– une période P_i



$$C_i = c_1 + c_2 + c_3$$

Figure 1. *Les différentes caractéristiques temporelles d'une tâche : date de réveil, durée de la tâche, délai critique et période*

Pour calculer C_i, une pire durée est associée à chaque bloc fonctionnel (voir [BAB 96a, SHA 89, PAR 93, PUS 89]). Nous ne traitons pas explicitement les durées des primitives temps réel, mais nous les intégrons dans les durées des blocs avoisinants.

Nous dirons que le système peut être à départs simultanés si les r_i sont tous égaux ou à départs différés si les r_i ne sont pas tous égaux.

La k-ième instance de la tâche τ_i sera activée à l'instant $r_i + (k-1)P_i$.

Les délais critiques sont supposés stricts, et ils indiquent la durée maximale tolérée entre l'activation d'une instance de τ_i et sa terminaison. La k-ième instance de τ_i aura donc l'échéance absolue stricte $d_{i,k} = r_i + (k-1)P_i + D_i$. Nous ne prenons pas en compte la réentrance, ce qui se traduit par $D_i \leq P_i$: une instance doit être terminée avant l'activation de la suivante.

Enfin, si deux tâches se synchronisent par émission/réception, leurs périodes doivent être corrélées, afin d'éviter les fautes temporelles et les pertes de messages. Nous notons $e_{i,j}$ le nombre de messages émis par une instance de la tâche τ_i à destination de la tâche τ_j et $a_{i,j}$ le nombre de messages qu'une instance de la tâche τ_j attend de la tâche τ_i. Les fréquences d'émission et de réception doivent être les mêmes. En effet, si la fréquence d'émission est plus faible, la tâche réceptrice se trouvera retardée, et finira par commettre une faute temporelle. Si elle est plus

grande, une partie des messages ne sera pas prise en compte, et le terme de synchronisation perdra sa signification. Cela se traduit par la contrainte :

$$\frac{e_{i,j}}{P_i} = \frac{a_{i,j}}{P_j}$$

Nous définissons la **méta-période** de l'application par P = PPCM(P_1,…,P_n). Une grandeur caractéristique d'une application est sa **charge processeur** définie par

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i}.$$

U représente le taux d'utilisation du processeur requise par l'application. Une condition nécessaire immédiate d'ordonnançabilité de l'application est que U soit inférieur ou égal à 1 (on considère un système mono processeur).

2.2. *La tâche oisive*

Si U est strictement inférieur à 1, le processeur restera inactif pendant U(1-P) unités de temps toutes les méta-périodes. Afin de pouvoir modéliser ces temps creux dits **cycliques** qui reviennent périodiquement, nous introduisons une tâche appelée **tâche oisive** τ_0 dont les paramètres temporels sont : C_0 = U(1-P), D_0 = P_0 = P. La date de réveil sera précisée ultérieurement. De cette façon, nous travaillons toujours avec une charge processeur égale à 1, et nous pouvons localiser les temps creux dans une séquence. Lorsque l'on considère les algorithmes classiques d'ordonnancement en ligne, une telle tâche n'a que peu d'intérêt, puisqu'elle n'est jamais prioritaire, et ne s'exécute que lorsqu'elle est la seule prête. Par contre, si l'on souhaite élargir le cadre de l'étude hors ligne aux algorithmes non au plus tôt, ce qui peut permettre de parer à certaines situations de blocage ou d'inversion de priorité, elle devient nécessaire, et peut être exécutée même si d'autres tâches sont prêtes. Un autre avantage de l'introduction explicite de cette tâche est qu'elle permet de contrôler le nombre de temps creux placés au sein d'une séquence, ce qui garantit qu'il y en aura le nombre requis exactement.

Si l'application est à départs simultanés, il n'y aura pas d'autres temps creux que ceux intégrés par la tâche oisive. Par contre, dans le cas d'applications à départs différés, il peut y en avoir d'autres, en nombre fini, qui résultent des perturbations liées à la montée en charge progressive du système [GRO 99, GRO 00]. Les figures 2 et 3 illustrent ce phénomène au travers d'un exemple. Soit le système constitué de trois tâches indépendantes τ_1(r_1 = 0, C_1 = 1, D_1 = T_1 = 4), τ_2 (r_2 = 1, C_2 = 3, D_3 = P_3 = 6), τ_3 (r_3 = 3, C_3 = 1, D_3 = P_3 = 4), et la séquence fournie par l'algorithme au plus tôt ED (cf fig. 2). Ce système a une charge égale à 1, donc il n'y a pas de temps creux cycliques. Cependant, on constate la présence d'un temps creux entre les instants 6 et 7. Ce temps creux, qui ne réapparaît plus par la suite est appelé temps creux acyclique. La figure 3 présente le diagramme des charges de l'application calculé à partir de la séquence fournie par ED. A chaque instant, le diagramme donne le nombre total d'unités de temps d'exécution en attente, c'est-à-dire la somme des charges correspondant aux différentes instances en cours de tâches actives. Les

temps creux acycliques apparaissent lorsque la charge à traiter est nulle. Ceci peut être généralisé à tous les algorithmes au plus tôt.

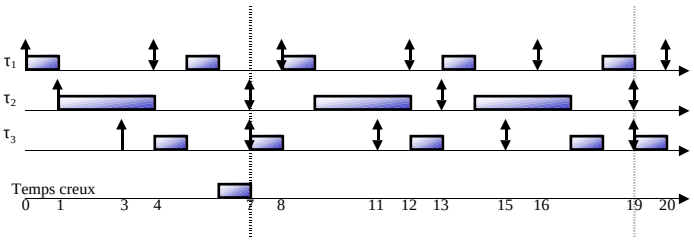


Figure 2. *Séquence ED pour le système de tâches : $\tau_1(r_1 = 0, C_1 = 1, D_1 = T_1 = 4)$, τ_2 $(r_2 = 1, C_2 = 3, D_3 = P_3 = 6)$, τ_3 $(r_3 = 3, C_3 = 1, D_3 = P_3 = 4)$*

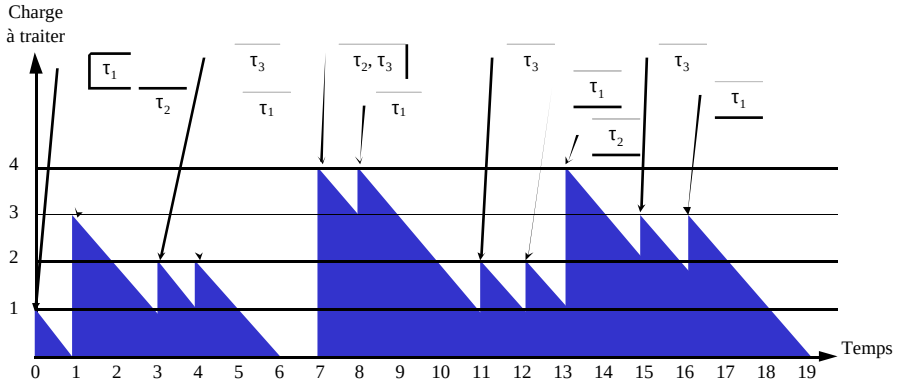


Figure 3. *Le diagramme des charges*

Il a été montré dans [GRO 99], [GRO 00] que :

1 - le nombre n_c de temps creux acycliques est borné et peut être déduit du diagramme de charges

2 - le dernier temps creux acyclique se produit avant l'instant $Max(r_i) + P$

3 - si le dernier temps creux se produit à l'instant t_C (égal à -1 s'il n'y a pas de temps creux), l'état du système est le même aux instants $t_C + 1$ et $t_C + P + 1$.

4 - la date du dernier temps creux est indépendante du choix de la stratégie d'ordonnancement (pourvu que ces temps creux soient ordonnancés au plus tard).

5 - la date de réveil de la tâche oisive τ_0 est choisie de manière à ce que le système entre le plus vite possible dans son comportement cyclique : $r_0 = t_c + 1$, et t_c est déduite du diagramme des charges.

Ces temps creux acycliques sont pris en compte dans une tâche non périodique, τ_c, appelée **tâche creuse**, de paramètres temporels : $<r_c = 0, C_c = n_c, D_c = t_c>$.

3. Modélisation des applications temps réel par réseaux de Petri

Nous présentons dans cette section une modélisation des applications temps réel utilisant des réseaux de Petri, intégrant des places colorées, avec ensemble terminal et fonctionnant sous la règle de tir maximal. Dans un premier paragraphe, nous rappelons les principales notions utilisées. Le lecteur pourra se reporter à [CHO 93], [PET 81] pour davantage de détails concernant les réseaux de Petri.

3.1. *Définitions et notations*

Un **réseau de Petri marqué** est un couple (N, M_o) où $N = (Q, T, W)$ avec :
Q : ensemble fini de places,
T : ensemble fini de transitions,
W : QxT TxQ → N la fonction de valuation.
et M_o : Q → N est le marquage initial, $M_o(p)$ étant le nombre de marques contenues initialement dans la place p

N fournit une description statique d'un système, par exemple une configuration de tâches pouvant interagir, M_o décrit son état initial. La dynamique du système est décrite par la **règle de tir** des transitions.

Une transition t est **valide** pour un marquage M ssi p Q, $M(p) \geq W(p, t)$. Son **tir** conduit au marquage M' défini par : p Q, $M'(p) = M(p) - W(p, t) + W(t, p)$.

Cette notion s'étend de manière naturelle aux séquences de transitions. L'ensemble des séquences valides depuis le marquage initial constitue le **langage** du réseau marqué. L'ensemble des marquages atteints par le tir de l'une de ces séquences est **l'ensemble d'accessibilité** du réseau marqué.

Un réseau fonctionne sous **la règle de tir maximal** si et seulement si à chaque tir, on franchit un ensemble maximal, au sens de l'inclusion, de transitions simultanément valides. La puissance d'expression des réseaux fonctionnant sous la règle de tir maximal est la même que celle des réseaux temporisés [RAM 74] fonctionnant au plus tôt [STA 90].

Une place est **colorée** [JEN 97] si elle contient des marques de différentes couleurs. Un ensemble C (fini) de couleurs est associé à chaque place colorée. Pour ces places, la fonction de valuation est définie par W : QxT xC Tx Q x C → N et la règle de tir par : une transition est **valide** à partir du marquage M ssi p Q et c C, $M(p, c) \geq W(p, t, c)$. Son **tir** conduit au marquage M' défini par :
p Q, c C, $M'(p, c) = M(p, c)-W(p, t, c)+W(t, p c)$.

Afin de contrôler l'évolution du réseau, on peut le contraindre par un **ensemble terminal** [CAR 84, VAL 81] : on adjoint au réseau un ensemble i de marquages, le plus souvent défini à l'aide d'une propriété sur les marquages, et on ne prend en compte que les séquences de tir pour lesquelles les marquages successifs atteints restent dans l'ensemble terminal. A tout instant donc, le marquage courant doit vérifier la propriété caractérisant l'ensemble terminal. L'ensemble de ces séquences est appelé **centre du langage terminal**.

Enfin, une autre technique de contrôle du réseau consiste à interdire certains séquencements de transitions, ce qui revient à interdire certains motifs dans les

séquences de transitions. Cela se traduit par des **contraintes de successeurs** : on adjoint au réseau une fonction S : T → p (T). On n'accepte que les séquences
u = $u_1 \ldots u_n$ (n N, u_i T) telles que i {1, …, n-1}, u_{i+1} S(u_i).

3.2. *Modélisation des applications temps réel*

Le modèle adopté comporte deux parties (cf fig.4) :

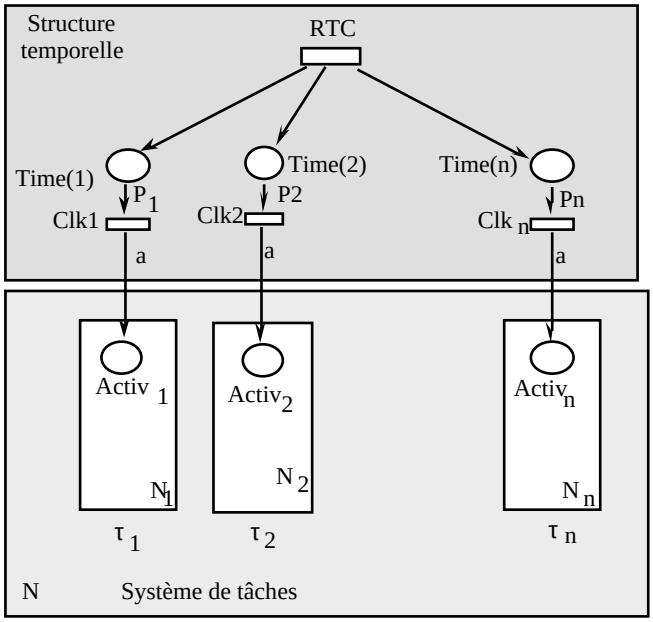


Figure 4. *Modélisation d'une application par un réseau Place/transition comportant deux composantes : un réseau modélisant le système de tâches, et une structure temporelle intégrant une horloge temps réel globale RTC et des horloges locales aux tâches (Time(i) et Clk$_i$))*

− Un réseau modélisant le système de tâches, où chaque transition est associée à une action. Le système étant mono processeur, une seule tâche s'exécute à la fois. Il y a donc une place processeur, contenant un unique jeton, en entrée/sortie de chacune des transitions, qui assure l'exclusion mutuelle entre les différentes actions effectuées par les tâches. Pour plus de clarté dans les schémas, cette place reste implicite. Chaque tâche est représentée de manière classique par un ensemble de transitions mises en séquence (cf fig.7). Il y a une transition par unité de temps d'exécution des tâches. De plus, afin de limiter la taille du modèle, les blocs dont la durée est supérieure à 3 sont condensés en un ensemble de 3 transitions (cf fig.5). Les communications sont représentées à l'aide de place de type boite aux lettres (cf la place MB en fig.7), et il y a une place par ressource (cf la place R en fig.7).

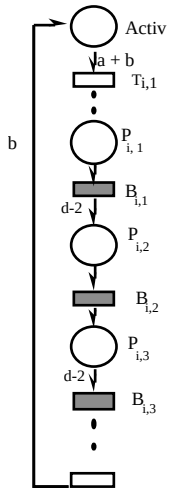


Figure 5. *Bloc de durée d > 2 : $B_{i,1}$ est tirée une fois, puis $B_{i,2}$ (d – 2) fois et enfin $B_{i,3}$ est tirée une fois*

La première place de chaque tâche τ_i, nommée $Activ_i$, est une place colorée associée à un ensemble de deux couleurs notées « a » et « b ». Cette place colorée représente de manière plus concise un ensemble de deux places non colorées A et B. Un jeton de couleur « a » est déposé à chaque activation de la tâche, et un jeton de couleur « b » indique qu'aucune instance de la tâche n'est en cours d'exécution. Lorsque la dernière action de la tâche s'exécute, la transition associée tire et dépose un jeton de couleur « b » dans la place $Activ_i$.

Une tâche ne peut commencer son exécution que si la place $Activ_i$ contient un jeton de couleur « a » et un jeton de couleur « b » : $W(Activ_i, T_{i,1}) = a + b$, où a+b signifie a ET b. Cela traduit le fait qu'une instance a été activée et qu'il n'y a pas d'autre instance en cours d'exécution, puisque nous avons supposé qu'il n'y avait pas de réentrance.

− Un réseau modélisant la structure temporelle. Ce réseau comporte :

- Une horloge globale notée RTC (Real Time Clock), modélisée par une transition sans place source. Comme le réseau fonctionne sous la règle de tir maximal, cette transition sera tirée à chaque tir de transitions. Elle va ainsi temporiser le fonctionnement du réseau, en produisant à chaque tir un jeton dans les horloges locales aux tâches. Cela crée une représentation logique du temps, une unité de temps d'exécution étant représentée par un tir de la transition RTC.

- Des horloges locales aux tâches. Il en existe une par tâche. Elles sont composées d'une place (Time(i)) permettant de comptabiliser le temps écoulé depuis la dernière activation de la tâche, et d'une transition Clk_i, qui tire toutes les P_i unité de temps (i.e. $W(Time(i),Clk_i) = P_i$ si P_i est la période de la tâche). Elle produit alors un jeton de couleur « a » dans la place $Activ_i$.

Les marquages initiaux des places $Activ_i$ et Time (i) permettent de prendre en compte la date de première activation de la tâche, c'est-à-dire d'intégrer le paramètre r_i. Nous supposerons que $r_i <= P_i$. Les autres cas induisent une légère modification du réseau que nous ne présentons pas ici, car elle n'a pas d'incidence sur la suite des résultats présentés, et n'est qu'instrumentale. Ces marquages sont définis par :

$$M_0(\text{Time(i)}) = \begin{cases} 1 \text{ si } r_i = 0 \\ P_i - r_i + 1 \text{ si } r_i > 0 \end{cases}$$

$$\text{et } M_0(\text{Activ}_i) = \begin{cases} a + b \text{ si } r_i = 0 \\ b \text{ si } r_i > 0 \end{cases} .$$

Si $r_i = 0$, la tâche à l'instant initial est : - activée (présence d'un jeton de couleur « a » dans la place $Activ_i$) et - au repos (présence d'un jeton de couleur « b » dans cette même place). A l'instant 0, la place Time(i) contient une marque, correspondant au premier top de l'horloge déclenchée à l'instant 0.

Si r_i est non nul, à l'instant initial, la tâche est au repos ce qu'exprime la présence d'un jeton de couleur « b » dans la place $Activ_i$, mais n'est pas activée, il n'y a donc pas de jeton de couleur « a ». Le déclenchement se fera au bout de r_i unités de temps. Comme les déclenchement ont lieu toutes les P_i unités de temps, on suppose que l'horlogerie a été déclenchée à l'instant - (P_i - r_i), ce qui se traduit par la présence de (P_i - r_i + 1) marques dans la place Time(i).

Reste à prendre en compte les délais critiques, grâce à l'introduction d'un ensemble terminal i défini par :

$\mathbf{M \in i \Leftrightarrow M(Time(i)) > D_i \Rightarrow M(Activ_i) = b \text{ et}}$

$\mathbf{M(Time(i)) = 1 \Rightarrow M(Activ_i) = a + b \text{ ou } M(Activ_i) = b}$

La première équation, qui intervient pour les tâches qui ne sont pas à échéance sur requête ($D_i < P_i$), indique que lorsque survient l'échéance de l'instance en cours, celle-ci doit avoir totalement terminé son exécution : il n'y a plus de marque « a », c'est à dire qu'elle a commencé son exécution depuis sa dernière activation, et il y a une marque « b », qui indique que cette exécution est terminée. La seconde équation intervient pour les tâches à échéance sur requête($D_i = P_i$). Juste après la réactivation de la tâche, la place Activ doit contenir « a + b », i.e. la tâche doit avoir terminé complètement l'exécution de l'instance précédente. Le cas $M(Activ_i) = b$ n'intervient qu'au tout début, dans le cas où $r_i = P_i$.

Le réseau fonctionne sous la règle de tir maximal : à chaque instant, RTC tire, toutes les horloges locales valides tirent, et une unique transition du sous-réseau modélisant le système des tâches tire, l'unicité est due à la place implicite Processeur. En particulier, il se peut que ce soit une transition issue de la tâche oisive ou de la tâche creuse, auquel cas il y a un temps creux. Notons qu'il est préférable d'avoir intégré les temps creux dans les tâches oisive et creuse plutôt que de travailler avec le système de tâches initial, et de choisir à certains moments de ne pas tirer de transitions, car dans notre cas, le nombre de temps creux est contrôlé, et on ne risque pas d'en introduire trop, ce qui conduirait à une faute temporelle.

Afin d'illustrer notre méthode, considérons une application, composée de trois tâches périodiques :

- La tâche τ_1 est composée d'un bloc de durée 2, puis émet un message en direction de τ_2 et enfin termine par un bloc de durée 1.
- La tâche τ_2 commence par la réception du message émis par τ_1, puis comporte un bloc de durée 1, et enfin termine par une section critique de durée 1 utilisant une ressource R.
- τ_3 commence par un bloc de durée 1, puis une section critique de durée 2 utilisant la ressource R et enfin termine par un bloc de durée 1.

Ces tâches possèdent les paramètres temporels décrits ci-après τ_1 : <r1 = 3, C_1=3, D_1=P_1=8> , τ_2 : <r_2=0, C_2=2, D_2= P_2=8> et τ_3 : <r_3=0, C_3=4, D_3=14, P_3=16>. La charge processeur est égale à 14/16, donc il y a 2 temps creux toutes les 16 unités de temps d'exécution.

Le diagramme des charges est donné sur la figure 6. Il est construit entre les instants 0 et Max(r_i) + P = 19. Dans cette plage apparaissent 2 temps creux. Ce sont les temps creux cycliques. Il n'y a donc pas de temps creux acycliques. On adjoint donc au système une tâche τ_0 <r_0=0, C_0=2, D_0=P_0=16>, mais il est inutile d'ajouter une tâche creuse acyclique au système. Le réseau présenté sur la figure 7 modélise cette application.

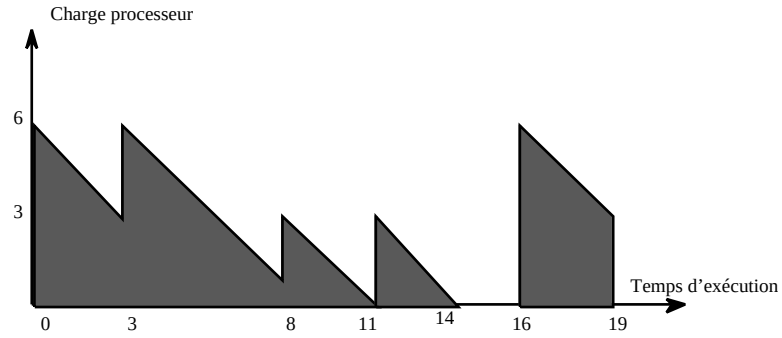


Figure 6. *Le diagramme des charges de l'application*

Les tâches τ_1, τ_2 et τ_0 sont à échéances sur requête, et la tâche τ_3 a une échéance inférieure à sa période. L'ensemble terminal i est ici défini par les relations :

M(Time(1)) = 1 M($Activ_1$) = a + b.
M(Time(2)) = 1 M($Activ_2$) = a + b.
M(Time(3)) > 14 M($Activ_3$) = b.
M(Time(0)) = 1 M($Activ_0$) = a + b.

Notons enfin que de façon générale, la tâche oisive consiste en un bloc de durée C_0 et que donc sa prise en compte dans le modèle nécessite au pire l'adjonction de 4 places : une place Time(0) et trois places au plus pour le corps de la tâche, et de 4 transitions : la transition Clk_0 et 3 transitions pour le corps de la tâche, selon le schéma de la figure 5. La tâche creuse, lorsqu'elle existe, consiste en une place contenant n_c jetons et en une transition reliée à cette place et à la place Processeur.

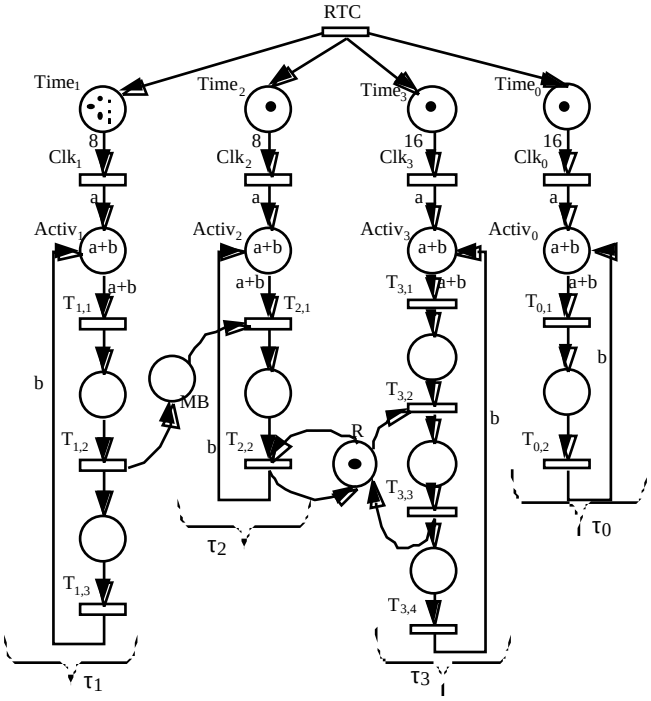


Figure 7. *Réseau modélisant un système de trois tâches se synchronisant et utilisant une ressource critique*

3.3. *Validation du modèle*

Le centre du langage terminal du réseau décrit dans le paragraphe précédent permet d'obtenir exactement l'ensemble des séquences valides. Puisque le réseau fonctionne sous la règle de tir maximal, les séquences obtenues par simulation du réseau de Petri sont écrites sur l'alphabet $p(T)$. Plus précisément, à chaque étape, des transitions du système d'horlogerie tirent, ainsi qu'une transition du système de tâche. L'adjonction de la tâche oisive et de la tâche creuse garantit qu'il y aura toujours une transition à tirer. Si l'on projette les séquences obtenues sur le système de tâches (on omet les transitions de l'horlogerie) et si l'on étiquette chaque transition par le nom de la tâche à laquelle elle appartient, alors, il a été établi dans [GRO 96] que l'ensemble des séquences obtenues est exactement l'ensemble des séquences valides :

une séquence d'ordonnancement d'un système de tâches S est valide si et seulement si elle peut être obtenue par simulation du réseau de Petri modélisant le système.

4. Analyse d'une application

4.1. *Description de la méthodologie*

La méthodologie générale comporte 4 étapes :

1 - Constitution du système de tâches à partir du cahier des charges, avec détermination des contraintes temporelles. Nous ne traitons pas cette étape (voir par exemple [BAB 96b]).

2 - Modélisation du système de tâches par un réseau de Petri avec marquages terminaux, fonctionnant sous la règle de tir maximal.

3 - Construction du graphe des marquages associé au réseau.

4 - Construction de l'ensemble des séquences valides à partir du graphe des marquages, avec extraction des séquences optimales par application de critères de sélection.

4.2. *Construction du graphe des marquages*

Un graphe des marquages dont les arcs sont étiquetés par l'alphabet $\{\tau_0, \ldots, \tau_n\}$ est construit. Nous allons successivement étudier la profondeur de ce graphe, donner quelques heuristiques concernant sa taille et décrire une méthode permettant d'en limiter le temps de calcul.

4.2.1. *Profondeur du graphe des marquages*

Si le système est à départs simultanés, les tâches sont toutes dans le même état aux instants 0 et P (PPCM des périodes) [LEU 80]. On construit donc le graphe des marquages sur une profondeur P, l'état initial étant un état d'accueil.

Si le système n'est plus à départs simultanés, nous avons vu que l'on peut à l'aide du diagramme des charges déterminer la date t_c du dernier temps creux acyclique, et qu'à l'instant t_c+1, le système entre dans un comportement cyclique de période P. Il s'ensuit que l'on construit le graphe des marquages sur une profondeur $t_c + P + 1$ et que l'état obtenu à l'instant t_c+1 est un état d'accueil.

4.2.2. *Taille du graphe*

Considérons un système de tâches à départs simultanés, de méta-période P.

Au cours de la méta-période P, une tâche τ_i est exécutée $\frac{P}{P_i}$ fois, donc en tout, $\frac{P \times C_i}{P_i}$ transitions de la tâche τ_i vont tirer pendant la méta-période.

Le graphe sera alors contenu dans un hyperpavé qui est une généralisation à une dimension quelconque de la notion de parallélogramme complètement maillé, de

dimension n + 1 et de côtés respectifs de tailles $\frac{P.C_0}{P_0}$, …, $\frac{P.C_n}{P_n}$. Le nombre de sommets est donc borné par $\prod_{i=0}^{n}\left(\frac{P.C_i}{P_i}+1\right)$.

Mais cette borne est extrêmement pessimiste : en effet, les interactions entre tâches, ainsi que le respect des diverses échéances vont en fait limiter fortement le nombre de sommets, en interdisant un certain nombre de préemptions d'une part, et en conférant d'autre part au graphe une structure de grappe autour de la diagonale de l'hyperpavé. A titre d'illustration, considérons le système comportant deux tâches τ_1 et τ_2 de paramètres temporels $<r_1 = 0, C_1 = 9, D_1 = P_1 = 21>$ et $<r_2 = 0, C_2 = 4, D_2 = P_2 = 7>$. Les deux tâches étant indépendantes, les séquences valides sont obtenues par entrelacement d'une instance de τ_1 et de trois instances de τ_2. Les entrelacements retenus doivent garantir que chacune des instances de τ_2 vérifie bien son échéance Le graphe associé est donné figure 8.

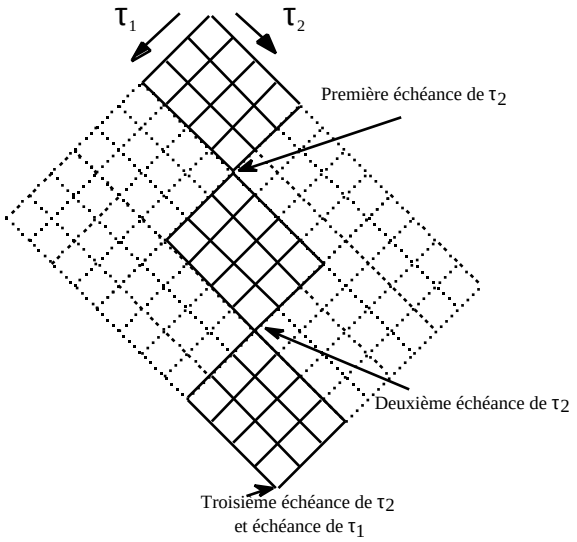


Figure 8. *Un graphe d'ordonnancement pour une application à départs simultanés*

Si nous considérons le réseau donné figure 7, la borne théorique donne 525 états, alors que le graphe ne comporte en fait que 53 états.

Dans le cas de systèmes à départs différés, les résultats sont similaires, mais un premier hyperpavé de montée en charge du système (de profondeur t_c) préfixe le pavé de profondeur P comme l'illustre la figure 9. Le système considéré comporte deux tâches τ_1 et τ_2 de paramètres temporels $<r_1=5, C_1= 3, D_1 = P_1 = 7>$ et $<r_2= 0, C_2= 8, D_2= P_2= 14>$. Les tâches sont indépendantes, donc à nouveau, les séquences valides correspondent aux entrelacements respectant toutes les échéances. Par

ailleurs, le système est à départ différé, de charge 100%, et le diagramme des charges montre qu'il existe un temps creux acyclique, le système comporte donc une troisième tâche τ_c.

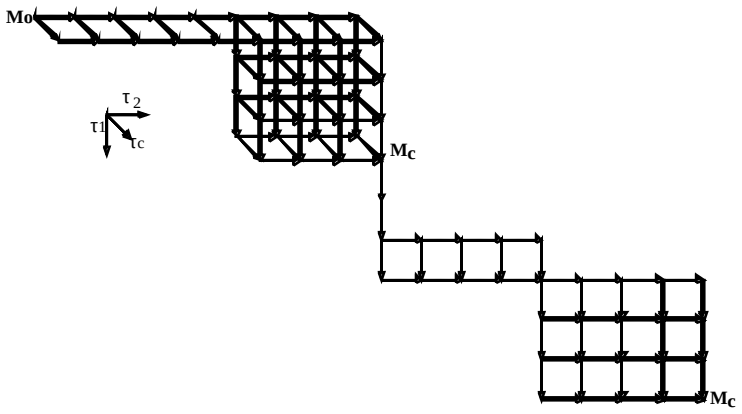


Figure 9. *Un graphe d'ordonnancement pour une application à départs différés incluant un temps creux acyclique*

4.2.3. *Algorithme de construction du graphe*

La construction du graphe des marquages se fait en profondeur d'abord et utilise des techniques de retour en arrière. Lorsque l'on atteint un nœud non terminal (une faute temporelle est commise), correspondant à un interblocage ou sans successeur, on le supprime, et on remonte à son père.

Afin de limiter l'utilisation du retour en arrière, coûteuse en temps, nous utilisons une table de prévision du futur qui permet d'anticiper sur la détection de faute temporelle : s'il reste u unités de charge de travail à une tâche dont la prochaine échéance se produira dans e unités de temps, avec e < u, on peut d'ores et déjà prévoir qu'il y aura faute temporelle, sans attendre d'atteindre effectivement un marquage non terminal.

Dans ce but, nous maintenons une table qui possède une entrée par tâche. Chaque entrée contient : le numéro de la tâche, la charge de travail qu'il lui reste à effectuer pour son instance courante, et sa **latence**, qui est le temps lui restant d'ici sa prochaine échéance (cf fig.10).

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

Figure 10. *Table de prévision à l'instant t triée par ordre croissant de latence*

Cette table est maintenue triée (en ordre croissant) par rapport aux latences, ce qui revient à la maintenir triée par rapport aux échéances absolues suivant les

priorités ED. Elle est mise à jour à chaque unité de temps. Soit la tâche $\tau_{t,i}$ dont le prochaine échéance se produira à l'instant t + latence($\tau_{t,i}$). Toutes les tâches $\tau_{t,k}$ avec k < i ont des échéances plus proches, par suite, les tâches $\tau_{t,0}$, …, $\tau_{t,i}$ devront toutes effectuer leur charge de travail restant avant l'instant t + latence($\tau_{t,i}$).

Donc, si i tel que $\sum_{k=0}^{i} \text{Travail}\,\tau_{t,k} > \text{latence}\,\tau_{t,i}$, alors quel que soit l'ordre dans lequel ces tâches sont ordonnancées, l'une au moins d'entre elles terminera après l'échéance de τ_i et donc dépassera sa propre échéance.

Il n'est pas possible d'estimer dans le cas général le gain en terme de nombre de nœuds du graphe, car celui-ci dépend étroitement de la configuration considérée. Considérons le système formé de deux tâches indépendantes τ_1 et τ_2, de paramètres temporels : $<r_1 = 0, C_1 = 10 = D_1, P_1 = 20>$ et $<r_2 = 0, C_2 = 10, D_2 = P_2 = 20>$. Pour respecter son échéance, la tâche τ_1 doit s'exécuter dès qu'elle est activée. Donc si on exécute une unité de temps d'exécution de τ_2 alors que l'instance en cours de τ_1 n'est pas terminée, il y aura faute temporelle, et il n'est pas nécessaire d'attendre l'instant 10 pour le constater. Dans ce cas, le gain apporté est de 50 %.

Enfin, la vérification aussi bien que la mise à jour de la table de prévision se font en temps linéaire par rapport au nombre de tâches [GRO 97].

4.3. *Calcul de l'ensemble des séquences valides et sélection*

L'ensemble exhaustif des séquences valides s'obtient à l'aide d'un parcours du graphe des marquages. Le principal problème provient de l'explosion combinatoire possible : le nombre de séquences valides peut être énorme. La suite de ce paragraphe traite de méthodes permettant de limiter cette explosion.

4.3.1. *Contraintes de successeurs*

Une première cause d'explosion combinatoire provient de l'entrelacement entre des segments de tâches indépendants. Le nombre de solutions possibles est égal à (somme_des_tailles)!. En terme de puissance d'ordonnançabilité, la prise en compte de toutes ces solutions n'apporte rien [BLA 76]. On peut donc éviter les préemptions multiples (ce qui limitera le surcoût associé) en limitant les préemptions aux seuls instants où le contexte change, i.e. lors de la demande ou de la libération d'une ressource, lors de l'envoi ou de l'attente d'un message, lors du réveil ou de la terminaison d'une tâche. Dans cette optique, nous introduisons des contraintes de successeurs :

Soit T l'ensemble des transitions issues du réseau modélisant l'ensemble des tâches, et soit u appartenant à T. Succ(u) est défini par :

— Si u correspond à une émission ou à une libération de ressource, les successeurs de u sont : la transition qui la suit dans le graphe représentant la tâche, les transitions en sortie des places Activ_i, qui correspondent aux premières actions des tâches (cas du réveil d'une tâche) et les transitions libérées par u, à savoir, soit la transition qui contient l'attente du message, soit toutes les transitions de demande de la ressource qui vient d'être libérée.

— Si u est la dernière transition de la tâche, ou si u est la dernière action d'un bloc précédant une demande de ressource (dans le réseau local de la tâche) ou si u admet une place de type boîte aux lettres en entrée (réception de message), alors succ(u) = T (on ne s'occupe pas de ce qui se passe à l'intérieur de la tâche elle-même, cela sera traité par la règle de tir).
— Enfin, sinon, la transition u est une transition d'un bloc, et la tâche ne peut être interrompue que par le réveil d'une autre tâche : les successeurs de u sont donc, outre les successeurs induits par le fonctionnement du réseau local, toutes les premières transitions des autres tâches.

Remarquons que la notion de successeurs peut être prise en compte dès la construction du graphe des marquages, et même améliorée afin de limiter les points de préemption aux seuls instants de changements de contexte. Ceci a pour conséquence la diminution de sa taille, et de son temps de calcul.

Considérons à nouveau l'exemple de la figure 7. Les contraintes de successeurs sont définies par :
$Succ(T_{1,1}) = \{T_{1,2} ; T_{2,1}, T_{3,1}, T_{0,1}\}$
$Succ(T_{1,2}) = \{T_{1,3}, T_{2,1}, T_{3,1}, T_{0,1}\}$
$Succ(T_{3,1}) = \{T_{3,2}, T_{1,1}, T_{2,1}, T_{0,1},\}$
$Succ(T_{3,3}) = \{T_{3,4}, T_{2,2}, T_{2,1},, T_{1,1}, T_{0,1}\}$
Pour toutes les autres transitions x, on a Succ(x) = T.

Le système admet 432 séquences valides. Si l'on utilise les contraintes de successeurs, données ci avant, on n'obtient plus que 72 séquences.

4.3.2. *Extraction de séquences optimales*

L'objectif de notre approche est de fournir une séquence valide qui sera implémentée dans une table au niveau du séquenceur. Cette séquence est constituée éventuellement d'un premier segment (si le système est à départs différés et si $t_C \geq 0$) suivi d'un segment de taille P, qui sera répété à l'infini (partie cyclique de la séquence). Lorsque l'ensemble de toutes les séquences possibles est grand, nous proposons d'utiliser un certain nombre de critères permettant de choisir parmi toutes les séquences, et de limiter le nombre de solutions. Les critères de performances retenus sont les suivants (dans ce qui suit, E désigne un sous-ensemble de tâches) :

- Détermination de séquences au plus tôt : les tâches de E sont exécutées le plus tôt possible. Notons que ceci ne revient pas à attribuer une très forte priorité à ces tâches. En effet, l'attribution de priorité pourrait conduire à un système non valide. Nous nous contentons ici de choisir au mieux parmi les solutions valides.
- Minimisation du temps de réponse maximal (resp. moyen) : le temps maximal (resp. moyen) entre l'activation et la terminaison des tâches de E est minimisé.
- Maximisation de la latence maximale (resp. moyenne) : le temps minimal (resp. moyen) entre la terminaison et l'échéance des tâches de E est maximisé.
- Minimisation du taux de réaction maximal (resp. moyen) : le temps maximal (resp. moyen) de réponse divisé par l'échéance des tâches de E est minimisé.

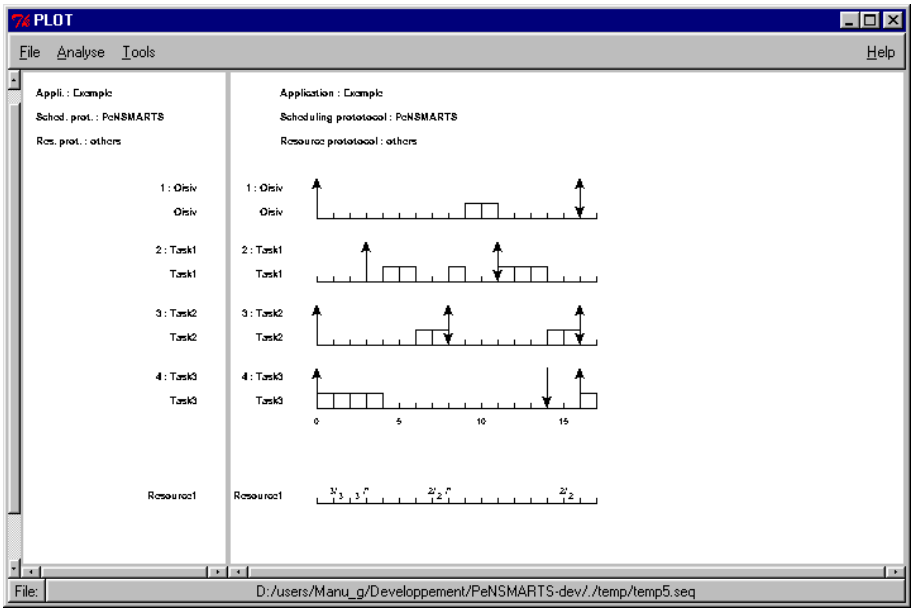


Figure 11. *Séquence minimisant le temps de réponse moyen*

Afin d'opérer l'extraction sélective des séquences, on commence dans un premier temps par valuer le graphe des marquages, la fonction de valuation dépendant du critère choisi : si l'on choisit d'exécuter au plus tôt un ensemble de tâches, un arc allant d'un nœud situé à profondeur n - 1 à un nœud situé à profondeur n sera valué par n s'il est étiqueté par une transition d'une tâche importante, et par 0 sinon; si l'on veut minimiser le temps de réponse, un arc allant d'un nœud situé à profondeur n-1 à un nœud situé à profondeur n sera valué par $\left(n - \left\lfloor \frac{n}{P_i} \right\rfloor * P_i\right)$ s'il est étiqueté par la dernière transition d'une tâche de E, et par 0 sinon. (voir [GRO 97] pour la description des autres fonctions de valuation). La détermination des séquences optimales se fait ensuite par une recherche de plus court chemin dans le graphe ainsi valué.

Considérons toujours le système de la figure 7. Si l'on choisit d'ordonnancer au plus tôt l'ensemble de tâches $\{\tau_1, \tau_2, \tau_3\}$, le nombre de séquences est de 2124. Si l'on veut minimiser le temps de réponse moyen de ce même ensemble de tâches, on obtient une unique séquence donnée figure 11

5. Étude de cas

Nous présentons ici une application constituée de tâches à départs différés, incluant des contraintes de précédence, utilisant une ressource exclusive et une ressource de type lecteur/écrivain, avec des échéances non toutes sur requêtes.

Considérons un système de tâches dédié au contrôle d'une pompe dans une mine. Dans un bassin de récupération des eaux, le niveau d'eau doit impérativement rester entre un niveau minimal et un niveau maximal. La mine est irriguée de manière naturelle par des infiltrations, et le système doit s'assurer que le niveau d'eau ne dépasse pas le seuil toléré (voir [MAT 96] pour une présentation plus détaillée). Quand le niveau d'eau devient trop haut, une pompe est mise en marche jusqu'à ce que le niveau ait atteint un niveau tolérable. Parallèlement, la concentration de méthane est aussi sous contrôle, une alarme est déclenchée si le niveau devient trop important, et en cas de taux jugé dangereux, la pompe à eau est arrêtée. Le procédé entier est visualisé sur un terminal de contrôle, et des sauvegardes sont effectuées régulièrement.

Le système est implémenté à l'aide de 7 tâches et de 2 ressources partagées : le terminal (Term) et une mémoire (Mem) partagée par les tâches d'acquisition (accès en écriture), les tâches d'affichage et de sauvegarde (accès en lecture). Le tableau de la figure 12 décrit le système de tâches.

Tâche	r	C	R	T	Term.	Mém.	Précède	U	B
Acquisition du niveau d'eau	0	13	100	100	non	oui	Contrôle	0,1	70
Acquisition du niveau de méthane	0	14	100	100	non	oui	Contrôle	0,1	70
Contrôle	10	15	70	100	non	non	Commande de la pompe Affichage de l'alarme	0,15	0
Affichage du système	20	70	500	500	oui	oui		0,14	25
Affichage de l'alarme	0	25	100	100	oui	non		0,1	70
Commande de la pompe	20	12	100	100	non	non		0,12	0
Trace	20	32	500	500	non	oui		0,06	14

Figure 12. *Configuration du système de tâches*

La colonne B donne les valeurs des durées de blocages liées à l'utilisation des ressources. Ces valeurs sont prise en compte lors de l'évaluation des conditions suffisantes analytiques d'ordonnançabilité. On constate ici que ces conditions ne sont pas vérifiées. Une approche hors ligne s'avère donc nécessaire.

La charge processeur du système est égale à 497/500, il y a donc 3 temps creux cycliques toutes les 500 unités de temps. La construction du diagramme des charges, incluant une tâche oisive de paramètres $<r_0 = 0, C_0 = 3, D_0 = P_0 = 500>$, montre qu'il n' y a pas de temps creux acyclique (cf fig 13). La durée de simulation sera donc de 500 unités de temps. Le réseau de Petri modélisant cette application est donné en annexe.

L'analyse de ce système à l'aide de notre méthodologie, en utilisant des contraintes de successeurs conduit à un graphe des marquages comportant 59958 nœuds, calculé en une dizaine de minutes sur un PENTIUM 450. On aboutit à un nombre de séquences de l'ordre de $335\ 10^{21}$. Mais si l'on choisit de minimiser le temps moyen de réponse, on n'obtient plus que 4 séquences pour un graphe de 521 nœuds. Nous ne donnons pas le chronogramme, faute d'une place suffisante pour représenter 500 unités de temps.

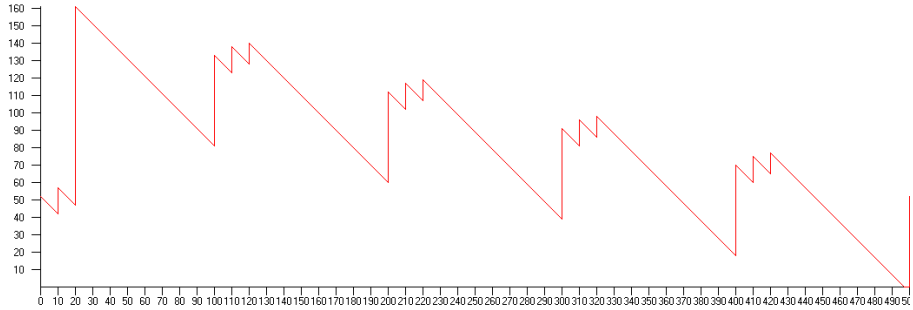


Figure 13. *Le diagramme des charges de l'application*

6. Conclusion

Nous avons présenté une méthodologie d'étude systématique de l'ordonnançabilité d'une application temps réel composée de tâches périodiques. Nous avons modélisé l'application par un réseau de Petri de type place/transition, avec marquages terminaux, fonctionnant sous la règle de tir maximal. Ce réseau permet, par simulation (donc construction de l'ensemble des marquages) d'obtenir toutes les séquences d'ordonnancement valides du système. Une telle méthode permet d'étudier n'importe quelle configuration, en particulier celles pour lesquelles les techniques classiques de validation sont inopérantes. Par ailleurs, cette approche permet de prendre en compte des critères de performances non exprimables en terme de priorité ou de contraintes temporelles, comme par exemple des notions de fonctionnement au plus tôt pour certaines tâches, ou l'optimisation des temps de réponse. Nous avons montré que la prise en compte de critères couplée avec l'utilisation d'heuristiques permet de limiter l'explosion combinatoire. La séquence ainsi obtenue et choisie pourra être implémentée directement au niveau du matériel, sans qu'aucun calcul ne soit plus nécessaire au cours de l'exécution de l'application. Notre approche se distingue des autres approches hors ligne par le fait que tout d'abord elle traite complètement le cas des applications à départs différés, ensuite, elle permet de prendre en compte des ressources de type lecteur/écrivain et enfin, les tâches se synchronisant peuvent avoir des périodes différentes, dès lors que les taux d'émission et de réception sont les mêmes.

La méthodologie présentée est mise en œuvre dans un outil nommé PeNSMARTS (Petri Net Scheduling, Modelling and Analysis of Real Time Systems) écrit en C, C++ et Tcl/Tk, et tournant sous Unix et sous Windows [GRO 98].

Les extensions de ce travail sont multiples : toujours dans le cadre mono processeur, nous nous intéressons d'une part à la prise en compte de tâches sporadiques de type alarme, et d'autre part à la prise en compte de durées réalistes, et non pas seulement des durées au pire lorsque les tâches contiennent des instructions conditionnelles. Une autre direction consiste à étendre la méthode aux systèmes multi processeur, puis, à passer aux systèmes distribués, avec prise en compte des durées de communication.

7. Bibliographie

[BAB 96a] J.P. BABAU, S. GERARD, F. COTTET, « Méthodologie de mesure des durées d'exécution des tâches d'une application temps réel », *RTS 96*, p 209 - 222, Paris 1996.

[BAB 96b] J.P. BABAU, « Étude du comportement temporel des applications temps réel à contraintes strcites basées sur une analysse d'ordonnançabilité », Thèse de Doctorat, Université de Poitiers, Juillet 1996.

[BAK 74] T.P. BAKER, Z.S. SU, « Sequencing with due-dates ans early start times to minimize maximum tardinees », Naval Research Logistic Quaterly, 21, p. 171-176, 1974.

[BAK 91] T.P. BAKER, "Stack-based Scheduling of Realtime Processes", *The Journal of Real-time systems*, n°3, p 67-99, Kluwer Academic Publishers, 1991.

[BAR 90] S.K. BARUAJ, L.E. ROSIER, R.R. HOWELL, « Algorithms and complexity concerning the preemptive scheduling of Periodic real-time tasks on one processor », *Real-time Systems*, 2, 1990.

[BEA 96] J.P. BEAUVAIS, « Étude d'algorithmes de placement de tâches temps- réel périodiques complexes dans un système réparti », Thèse de doctorat, École centrale de Nantes, Juin 1996.

[BEA 98] J.P. BEAUVAIS, A.M. DEPLANCHE, « Affectation de tâches dans un système temps réel réparti », *Technique et science informatiques*, vol 17 (1), 1998.

[BER 91] B. BERTHOMIEU, M. DIAZ, « Modelling and verification of time dependant systems using Time Petri nets », *IEEE Transactions on software engineering*, vol .17, n° 3, p 259-273, 1991

[BLA 76] J. BLAZEWICZ, « Scheduling dependent tasks with different arrival times to meet deadlines », in E. Gelenbe, H. Bellner (eds), *Modelling and performance evaluation of computer systems*, Noth-Holland, Amsterdam, 1976.

[BRA 75] P. BRATLEY, M. FLORIAN, P. ROBILLARD, « Scheduling with earliest start and due date constraints on multiple machines », Naval Research Logistic Quaterly, 22(1), p.165-173, 1975.

[BUT 97] G. BUTTAZZO, *Hard real-time computing systems*, Kluwer Academic Publishers, 1997.

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

[CHE 90] M. CHEN, K. LIN, « Dynamic priority ceilings : a concurrency protocol for real-time systems », *the Journal of real-time systems* 2 (n°4), p 325-346, 1990.

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

[COT 94] F. COTTET, J.P. BABAU, « Off-line temporal analysis of hard real-time applications », *second IEEE Workshop on Real-Time applications*, Washington DC, p 28-32, 1994.

[DEL 99] J. DELACROIX , « Ordonnancement de tâches apériodiques », *École d'été temps réel 99*, p. 69-82, Futuroscope, Septembre 1999.

[ELL 97] J.P. ELLOY, « Systèmes réactifs, synchronisme, tâches et événements », *École d'été temps réel*, p.28-37, Poitiers, Septembre 1997.

[DER 74] M.L. DERTOUZOS, « Control Robotics : the procedural control of physical processes », *Informatop, Processing* 74, North-Holland publishing company, 1974.

[GRO 96] E. GROLLEAU, « Ordonnancement de systèmes de tâches temps réel à l'aide de réseaux de Petri », Rapport de DEA, LISI, Poitiers, 1996.

[GRO 97] E. GROLLEAU, A. CHOQUET-GENIET, F. COTTET, « Ordonnancement optimal de systèmes de tâches temps réel à l'aide de réseaux de Petri », *AGIS* 1997, p.239-246, Angers.

[GRO 98] E. GROLLEAU, A. CHOQUET-GENIET, F. COTTET, « Analyse temporelle de systèmes temps réel à l'aide de réseaux de Petri : PENsMARTS », *AFADL 98*, p 137-148, Futuroscope, 1998.

[GRO 99] E. GROLLEAU, « Ordonnancement temps réel hors ligne optimal à l'aide de réseaux de Petri en environnement monoprocesseur et multiprocesseur », Thèse de l'unversité de Poitiers, Novembre 1999.

[GRO 00] E. GROLLEAU, A. CHOQUET-GENIET, « Cyclicité des ordonnancements des systèmes de tâches périodiques différées », RTS 2000, Paris.

[JEN 97] K. JENSEN, *Coloured Petri nets. Basic concepts, analysis methods and Practical Use*, Monographs in theoretical Computer Science, Springer Verlag 1997.

[KAI 82] C. KAISER, » Exclusion mutuelle et ordonnancement par priorité », *Technique et science informatiques*, vol 1, no 1, p 59-69, 1982.

[LIU 73] 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.

[LEU 80] 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.

[MAT 96] J. MATHAI ET AL., *Real time systems, secification, Verification and analysis* Prentice Hall, International Series in Computer Science, 1996.

[MEN 83] M. MENASCHE, B. BERTHOMIEU, « Time Petri nets for analysing and verifying time dependant communication protocols », *Protocols Specifications, testing and verification* III, H.Rudin and C. H. West eds, Elsewier Science Publishers, North-Holland, 1983.

[MER 76] P. MERLIN, D.J. FARBER, « Recoverability of communication protocols », *IEEE transactions on communications*, vol. COM_24, n°9, Set. 1976.

[MOK 83] A.K. MOK, « Fundamental design problems of distributed systems for the Hard-real time environnement », PH.D Dissertation, MIT, 1983.

[MOK 78] A.K. MOK, M.L. DERTOUZOS, « Multi processor scheduling in a hard real-time environment », *7th Texas conference on computer Systems*, 1978.

[PAR 93] C.Y. PARK « Predicting Program Execution Times by Analysing Static and Dynamic program Paths » *Real time Systems*, 5 (1), p 31-62, 1993.

[PET 81] J.L PETERSON, *Petri nets theory and the modeling of systems*, Prentice-Hall, 1981.

[PUS 89] P. PUSHNER, CH KOZA, « Calculating the Maximum Execution Time of Real time Programs », *Real Time systems* 1, p 159-176, 1989.

[RAM 74] C. RAMCHANDANI, « Analysis of Asynchronous concurrent systems by Petri nets », Project MAC, TR-120, MIT, Cambridge, MA, Fev. 1974.

[SHA 89] A. SHAW, « Reasonning about Time in Higher-Level Language Software », *IEEE Transactions on Software Engineering*, 15 (7), p 875-889, 1989.

[SHA 90] L. SHA, R. RAJKUMAR, J. LEHOCKZY, « Priority inheritance protocols : an approach to real time synchronisation », *IEEE transaction computers*, vol 39, N°9, p 1175-1185, 1990.

[STA 88] J.A. STANKOVIC, « Misconception about Real-Time Computing : a serious problem for the next generation systems », *IEEE Computer Magazine*, 21 (10), p 0-19, 1988.

[STA 90] P.H. STARKE, « Some properties of timed nets under the earlier firing rule », *LNCS* vol 424, Springer Verlag, p. 418-432, 1990.

[STA 98] J.A. STANKOVIC, M. SPURI , K. RAMAMRITHAM, G. C. BUTTAZZO, *Deadline scheduling for real-time systems , Kluwer Academic Publishers, 1998.*

[TSA 93] J. TSAI, S. YANG, Y. CHANG, « Schedulability analysis of real-time systems using Timing constraint Petri nets », *Comp Euro 93,* p.375-382, 1993.

[VAL 81] R. VALK, G. VIDAL-NAQUET, « Petri nets and regular languages », *Journal of Computer and System Sciences*, vol. 23, n° 3, p 299-3 25, Academic press, 1981.

[XUP 90] J.XU, D.L PARNAS « Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations », *IEEE Transactions on Software Engineering*, vol 16, N°3, p 360-369, Mars 1990.

[ZAM 97] J. ZAMORANO, A. ALONSO, J.A. DE LA PUENTE, « Automatic generation of cyclic schedules », WRTP'97- *22nd IFAC/IFIP Whorkshop on Real-Time programming* p 145-151, Elsevier Sciences, 1997.

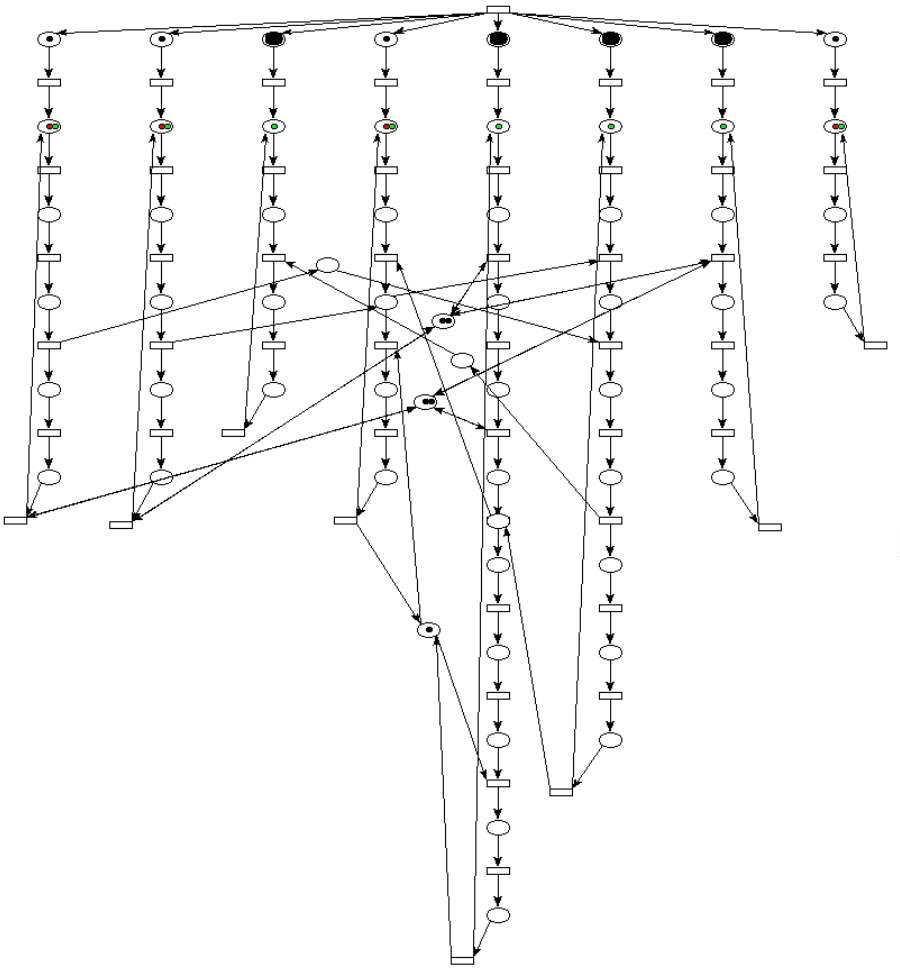
Article reçu le 22 Juin 1999
Version révisée le 2 Mars 2000
Rédacteur responsable : Bernard BERTHOMIEU

Annie Choquet-Geniet *est maître de conférences à l'université de Poitiers, et membre du laboratoire d'informatique scientifique et industrielle (LISI) de l'ENSMA, Futuroscope. Ses travaux portent sur l'ordonnancement des applications temps réel.*

Emmanuel Grolleau *est maître de conférences à l'ENSMA, Futuroscope. Il a soutenu en Novembre 1999 une thèse portant sur l'ordonnancement hors ligne des applications temps réel, effectuée au sein LISI à l'ENSMA, Futuroscope sous la direction de Francis Cottet et Annie Choquet-Geniet.*

Francis Cottet *est professeur à l'ENSMA, Futuroscope. Il est actuellement directeur adjoint LISI de l'ENSMA, et il dirige l'équipe temps réel.*

Annexe



Le réseau de Petri modélisant le système décrit figure 12