Quantification du taux d'invalidité d'applications temps réel à contraintes strictes

Gaëlle Largeteau-Skapin, Dominique Geniet

Quantification du taux d'invalidité d'applications temps-réel à contraintes strictes

Analyse préliminaire au développement d'un atelier d'aide à la spécification d'application temps-réel.

Gaëlle Largeteau — **Dominique Geniet**

Laboratoire LISI (université de Poitiers - ENSMA)
Téléport 2 - 1 avenue Clément Ader BP 40109
F-86961 Futuroscope Chasseneuil Cedex, France.
glargeteau@yahoo.fr, dominique.geniet@laposte.net

RÉSUMÉ. *Ce travail porte sur la mise en place d'un atelier d'aide à la conception opérationnelle d'applications temps-réel. Actuellement, la validation temporelle s'appuie sur des outils de diagnostic déterminant la validité ou l'invalidité d'une application. Nous montrons ici comment construire des indicateurs de « qualité opérationnelle » aptes à diagnostiquer l'invalidité et surtout à la quantifier numériquement. Après avoir défini les caractéristiques attendues pour un indicateur de qualité opérationnelle, nous présentons une étude orientée « modèles », dans laquelle nous montrons comment, à partir d'une approche basée sur des automates finis, nous pouvons mettre en oeuvre un modèle géométrique qui permet :*
- de servir de support de calcul à nos indicateurs de qualité ;
- de servir de support de manipulation visuelle pour les utilisateurs de notre futur atelier d'aide à la conception opérationnelle.
Les tâches sont à durée fixe (pire cas) à départs différés et échéances reliées aux périodes fonctionnant sur une architecture multiprocesseur où les processeurs sont identiques et synchrones.

ABSTRACT. *This work is about an operational conception toolkit for real time applications. Temporal validation is usually based on tools with binary validation results. In this paper, we show how to build some quality indicators able to quantify invalidity numerically. First, we present an automata-based model and then its geometrical adaptation, which permits to compute a measure of invalidity.*

MOTS-CLÉS : *Ordonnançabilité, automates, langages rationnels, géométrie, mesure d'invalidité.*

KEYWORDS: *Schedulability, automata, regular languages, geometry, invalidity measure.*

RSTI - TSI. Volume 27 – n° 5/2008, pages 589 à 625

1. Introduction

Une application temps-réel est un système réactif, destiné à piloter un procédé (CNRS, 1988), elle est liée au procédé qu'elle contrôle et doit réagir à une vitesse instantanée du point de vue de ce procédé (Stankovic, 1996). Un tel système doit avoir un fonctionnement correct mais aussi ponctuel. C'est cette propriété que nous nous proposons d'étudier ici.

Le nombre de fonctions assurées par ce type de système est en constante augmentation et impose de plus en plus fréquemment plus d'un processeur. Notre étude se place donc dans un contexte multiprocesseur. Dans ce cadre, nous supposons que tous les processeurs sont identiques (même vitesse de calcul) et synchrones.

La conception et la réalisation d'une application temps-réel nécessitent plusieurs étapes : le cahier des charges précède la spécification (tant fonctionnelle qu'opérationelle), puis vient la validation. Nous nous intéressons plus particulièrement à la phase de validation temporelle et à la rétroconception qui intervient lorsqu'une application est détectée invalide. Nous souhaitons proposer aux concepteurs de telles applications un outil non seulement de validation mais aussi d'analyse permettant de détecter, en cas d'invalidité, les causes de l'invalidité et leurs influences respectives sur la non-ordonnançabilité. La validation temporelle s'appuie usuellement sur trois alternatives :

– l'utilisation de critères analytiques ;

– la simulation d'algorithmes d'ordonnancement en-ligne ;

– une analyse de type « model checking », dans laquelle des résultats de décision obtenus sur les modèles étudiés fournissent la décision d'ordonnançabilité.

L'utilisation de critères analytiques n'est utilisable que dans de rares cas réel, les seuls critères existants dans le contexte multiprocesseur concernent les systèmes à départs simultanés et échéance sur requête (Lee *et al.*, 1994; Baruah *et al.*, 1995; Srinivasan *et al.*, 2002; Carpenter *et al.*, 2004). Il n'existe aucune condition nécessaire et suffisante et aucun algorithme n'est optimal dans le contexte que nous étudions (Mok, 1983). La simulation est très utilisée, mais elle reste centrée sur un résultat de type « décision d'ordonnançabilité » pour une séquence d'ordonnancement particulière (RM ou EDF, par exemple (Buttazzo, 1997)). De plus, nous considérons des tâches à départs différés, contexte dans lequel la durée de la montée en charge du système (et par la même la durée minimale de simulation nécessaire pour valider l'application) est inconnue à ce jour (Choquet-geniet, 2005).

Dans notre étude, nous ne cherchons pas à diagnostiquer une séquence d'ordonnancement particulière, mais le système temps-réel dans son ensemble : est-il valide ? si non, est-il très « fortement » invalide ? etc. Répondre à ce genre de question suppose d'avoir une vue globale sur les possibilités d'ordonnancement du système, *i.e.* sur l'ensemble des séquences d'ordonnancement le concernant. C'est pourquoi nous devons nous orienter vers une approche « modèle », de façon à être capables de positionner le système étudié par rapport au « système valide le plus proche ».

Il nous faut donc :

– définir la notion de « système proche », et donc disposer d'une métrique sur l'ensemble des systèmes temps-réel ciblés par notre étude : ce sera l'objet de notre mesure d'ordonnançabilité ;

– utiliser cette mesure pour localiser les zones des séquences d'ordonnançabilité qui posent problème : c'est l'objet de la section 5 de ce papier ;

– déterminer le système valide « le plus proche » du système étudié, si il existe : ce point constitue le corps de nos recherches en cours, et sera présenté dans de futurs articles.

Deux grandes classes de modèles sont utilisées : les modèles temporisés pour lesquels une information temporelle explicite est incluse dans les états ou les transitions (Ramchandani, 1974; Chrétienne, 1983; Juanole, 1999; Alur *et al.*, 1994) et les modèles sans temps (à temps implicite) pour lesquels une transition représente l'écoulement d'une unité de temps (Choquet-Geniet *et al.*, 1996; Geniet, 2000; Kwak *et al.*, 2002). Les modèles temporisés sont très expressifs mais les propriétés d'accessibilité nécessaires pour valider un système temps-réel sont dans certains cas non décidables, dans d'autres cas difficiles. Nous avons donc choisi un modèle sans temps permettant de représenter un système fonctionnant sur une architecture multiprocesseur : les automates finis et les langages rationnels qui leurs sont associés (Geniet, 2000). Dans ce modèle, une séquence d'exécution d'une tâche est associée à un mot. Un produit de langages permet de représenter l'ensemble de tous les ordonnancements possibles du système en tenant compte des partages de ressources et de processeurs.

Les techniques de validation actuelles apportent un diagnostic binaire au problème de l'ordonnançabilité. Nous souhaitons raffiner cette réponse : lorsqu'une application est validée par une analyse pire cas, nous voulons pouvoir rajouter un commentaire sur la « qualité » de la validité, l'application est soit valide mais très contrainte et ne pourra pas accepter de tâches supplémentaires (tâches d'alarme apériodiques ou sporadiques) ou au contraire peu contrainte et pourra sans problème faire face à un imprévu. Dans le cas d'applications invalides, en donnant une mesure numérique de l'invalidité, nous comptons évaluer l'effort à fournir (le nombre de modifications sur les contraintes temporelles des tâches) pour rendre valide une configuration de tâches qui ne l'est pas.

A notre connaissance, les seuls travaux portant sur une mesure d'invalidité sont basés sur une étude probabiliste de systèmes non déterministes (nombre de tâches et caractéristiques temporelles non connus de façon certaine) (Nissanke *et al.*, 2002). Notre travail consiste donc à proposer une définition d'une telle mesure dans le cadre de système deterministe et donc à déterminer un modèle adapté à cette définition.

Le modèle que nous utilisons, MARTA (*Model based on Automata for Real Time Applications*), collecte l'ensemble des comportements d'une tâche dans un langage. La concurrence et la dépendance sont modélisées par le biais d'opérations sur les langages (produit homogène, opération de mélange et calcul de centre de langage). Ce modèle s'est avéré extrêmement riche pour représenter les ensembles de séquences

d'ordonnancement (il permet en particulier de s'abstraire de l'utilisation de WCET (Geniet *et al.*, 2001)). MARTA a permis d'obtenir des résultats sur la structure des ordonnancements ainsi qu'une évaluation dans le cas de configurations ordonnançables. Mais dans le cadre qui nous occupe ici, il ne permet pas de positionner le système étudié par rapport au « plus proche système valide », puisque lorsqu'il n'y a pas ordonnançabilité, l'automate obtenu accepte le langage vide. Il est donc très utile pour étudier des mesures de type « confort » lorsque le système est ordonnançable (on peut évaluer les temps de réponse, la gigue, etc. à l'aide de méthodes analytiques sur les automates). Une analyse de la structure des graphes de ces automates nous a assez naturellement amenés à les épurer (ne conserver que les 3 ou 4 états effectivement caractéristiques), puis finalement à les transformer en des objets géométriques. Ceci a donné naissance au modèle GRETA (*Geometrical model for REal Time Application*) (Largeteau *et al.*, 2004) :

– les chemins dans les automates se transforment naturellement en trajectoires dans l'espace géométrique ;

– le produit d'automates (dont la sémantique est le parallélisme) se transforme en un produit pseudo-cartésien ;

– les intersections de langages (qui implémentent les contraintes temporelles et « système ») deviennent des intersections par des demi-espaces.

Ce modèle, équivalent à MARTA en termes de puissance de représentation, fournit un objet géométrique non vide aussi bien dans le cas des « systèmes valides » que dans le cas des « systèmes invalides » . La distinction se fait par la topologie des objets : un objet connexe (en un seul bloc) modélise un système valide. Nous pouvons alors définir les mesures de validité en nous intéressant à la géométrie de ces objets : lorsqu'il y a non-connexité, alors il y a non-validité, et la distance euclidienne (par exemple) maximale entre deux composantes connexes de l'objet donne une valeur à laquelle on peut associer une sémantique d'*effort à produire*. Bien entendu, la géométrie que l'on met sur l'espace permet d'engendrer n'importe quel type d'étude : temps de réponse, gigue, toute combinaison possible.

En section 2, nous présentons nos objectifs de quantification ainsi qu'une spécification des propriétés que doit posséder une mesure de validité. En section 3 et 4 nous discutons des modèles et de leurs compétences en termes de mesure. En section 5, nous donnons une définition de notre mesure d'invalidité , nous vérifions qu'elle répond bien à la spécification faite en section 2 puis nous donnons les résultats de notre étude préliminaire sur le comportement de cette mesure.

2. Cahier des charges de l'evaluation de l'invalidité

Etant un outil permettant de quantifier l'effort nécessaire pour rendre une application ordonnançable, une mesure se doit de tenir compte du contexte d'ordonnancement : une application non ordonnançable sur un seul processeur peut l'être sur deux. Nous recherchons des mesures homogènes en termes d'effort pour atteindre l'ordon-

nançabilité : le principe est que si un système A est moins « complexe » à rendre ordonnançable qu'un système B, alors la mesure du système A est inférieure à la mesure du système B. Cette propriété entraîne en particulier l'homogénéïté pour ce qui concerne le nombre de processeurs (mais il ne s'agit bien sûr que d'une conséquence) : nous avons nécessairement une mesure nulle pour chaque système ordonnançable, et la mesure associée à un système en considérant une architecture à p processeurs est supérieure à la mesure de ce même système en considérant un $p+1$ processeurs.

De manière plus précise, nous notons $\mathrm{M}_p(\Gamma)$ la mesure d'invalidité de la configuration Γ en considérant un contexte comportant p processeurs. Nous notons $\mathcal{A}$ l'ensemble des configurations de tâches, $\mathcal{C}$ un contexte d'ordonnancement comportant k processeurs et $\mathcal{O}_C(\Gamma)$ l'ensemble des ordonnancements valides du système Γ. Une mesure d'invalidité définit une relation d'ordre *au plus aussi valide que* dans le contexte $\mathcal{C}$ (notée $\prec_{\mathcal{C}}$) sur l'ensemble $\mathcal{A}$: soit A et B $\in \mathcal{A}$,

$$A \prec_{\mathcal{C}} B \Leftrightarrow M_k(A) \leq M_k(B).$$

La première propriété d'une mesure d'invalidité est qu'elle doit être nulle pour les applications valides et non nulle pour les applications invalides puisqu'elle a pour vocation d'évaluer l'invalidité et non la validité.

$$M_k(\Gamma) > 0 \Leftrightarrow \mathcal{O}_C(\Gamma) = \emptyset.$$
$$M_k(\Gamma) = 0 \Leftrightarrow \mathcal{O}_C(\Gamma) \neq \emptyset.$$

De plus, une application est *moins* ordonnançable sur un processeur que sur deux. En effet, s'il n'existe aucune séquence valide pour deux processeurs, il n'en existe pas pour un seul processeur. Les modifications à apporter dans le cas monoprocesseur sont donc d'une part celles que l'on aurait dû effectuer dans le cas biprocesseur et d'autre part celles dûes au supplément de charge engendré par la perte d'un processeur. Il y a donc plus de modifications à apporter aux caractéristiques temporelles des tâches dans le cas où on ne dispose que d'un processeur. Ceci se traduit par :

$$M_p(\Gamma) > 0 \Rightarrow \forall k < p,\ M_k(\Gamma) \geq M_p(\Gamma).$$

Réciproquement, une application ordonnançable sur deux processeurs l'est sur toute architecture comportant plus de deux processeurs. La mesure doit donc vérifier :

$$M_p(\Gamma) = 0 \Rightarrow \forall k > p,\ M_k(\Gamma) = 0.$$

Un premier résultat, découlant directement de ces propriétés, est qu'une mesure satisfaisant à cette spécification sera à même de détecter les partages de ressources problèmatiques. En effet, si on considère une application de n tâches pour laquelle la mesure M_n serait non nulle, nous pouvons en déduire que malgrès le fait que chaque tâche puisse disposer d'un processeur, l'application est invalide. Dans ce cas, seul un partage de ressource peut-être la cause de l'invalidité. Dans la suite, nous proposons deux modèles pouvant supporter la définition d'une telle mesure.

3. Vers un modèle adapté à l'évaluation : le modèle MARTA

Le premier modèle que nous avons testé pour la définition d'une mesure d'invalidité est basé sur les langages rationnels et les automates finis (Autebert, 1994) non temporisés qui leur sont associés (Geniet, 2000). Le problème de l'ordonnançabilité se traduit alors par des problèmes d'accessibilité dans un graphe qui sont décidables dans le cadre des automates non temporisés. Nous présentons ce modèle dans le contexte (centralisé, processeurs identiques et durées d'exécution fixes), prenant en compte les caractéristiques temporelles des tâches ainsi que les propriétés de concurrence et de dépendance des systèmes de tâches temps-réel.

3.1. *Le modèle de tâche*

Les tâches que nous étudions sont caractérisées (Liu *et al.*, 1973) par des attributs temporels : une date de réveil (notée r), une durée d'exécution au pire (notée C), un délai (noté D) et enfin une période (notée T). Nous considérons par la suite que les dates de départ ne sont pas necessairement toutes égales (départs différés) et que les échéances ne sont pas sur requête ($D < T$).

Dans le modèle MARTA, l'exécution d'une tâche est représentée par un mot formé sur un alphabet de deux lettres associées à l'activité ou l'oisiveté de la tâche. L'ensemble de ces mots forme un langage qui constitue le modèle temporel de la tâche. Nous détaillons dans la suite de cette section comment construire de tels langages et les opérations permettant d'obtenir le modèle de l'application dans son ensemble.

Lors de son exécution, la tâche τ peut être active ou suspendue, selon qu'elle possède ou non le processeur. Pour chaque unité de temps u, l'observation de l'état de τ permet de construire une séquence des périodes d'activation/désactivation de τ. L'ensemble des séquences respectant la spécification temporelle constitue le modèle temporel de τ.

Nous nous donnons donc l'alphabet {**a**,•}, où la lettre **a** représente l'état d'activation d'une tâche pendant une unité de temps, et la lettre • représente une unité de temps d'inactivité de la tâche. Un mot sur cet alphabet représente alors une séquence d'activations/désactivations de la tâche.

Considérons, dans un premier temps, le régime permanent. La tâche τ, pour une instance, doit exécuter C instructions pendant les D premiers instants de sa période pour ne pas rater son échéance. Les C unités de temps pendant lesquelles τ doit être active engendrent autant d'occurences de la lettre **a**, les autres instants engendrent chacun une lettre •. La tâche étant préemptible à tout instant, les instants où la tâche est active et ceux où elle ne l'est pas ne sont pas placés de manière déterminée. Comme nous désirons représenter tous les comportements possibles (nous avons choisi une méthode exhaustive), nous devons modéliser toutes les combinaisons possibles avec les C lettres **a** et les $(D-C)$ lettres •. Nous utilisons ici l'opération de mélange, notée III, définie de la façon suivante :

$\forall$ a $\in \Sigma$, a ш ϵ = ϵ ш a = {a}.
$\forall$ (a,b,w,w') $\in \Sigma^2 \times (\Sigma^*)^2$, $a.w$ ш $b.w' = a.(w$ ш $b.w') \cup b.(a.w$ ш $w')$.
$\forall$ $L_1 \subset \Sigma^*$, $L_2 \subset \Sigma^*$, L_1 ш $L_2 = \bigcup_{(\alpha,\beta)\in L_1\times L_2} (\alpha$ш$\beta)$.

Intuitivement, cette opération consiste à entrelacer les lettres de deux mots. Sur les D premiers instants de la période de τ, $(\mathbf{a}^C$ ш $\bullet^{D-C})$ collecte donc l'ensemble des séquences d'allocation processeur compatibles avec les contraintes temporelles de τ. La tâche, ayant terminé son exécution, doit rester inactive jusqu'au réveil de son instance suivante. Comme la période est de T et que l'on a déjà considéré D unités de temps, il reste $T - D$ unités de temps d'inactivité à représenter. Le mot modélisant une période complète de τ appartient donc au langage : $(\mathbf{a}^C$ ш $\bullet^{D-C})$ $\bullet^{T-D}$. Pour modéliser la totalité du comportement de la tâche τ, il faut tenir compte de toutes ses instances qui peuvent être en nombre infini ; nous utilisons donc l'opérateur « * » et nous obtenons le langage : $((\mathbf{a}^C$ш$\bullet^{D-C})\bullet^{T-D})^*$. Il nous faut maintenant représenter l'inactivité de la tâche τ avant le réveil de sa première instance qui intervient après **r** unité de temps (régime transitoire). Nous utilisons un mot de taille r comportant uniquement la lettre $\bullet$ pour représenter cette séquence d'inactivité et nous obtenons finalement le langage (cf. figure 1) : $\bullet^r \left((\mathbf{a}^C$ш$\bullet^{D-C})\bullet^{T-D}\right)^*$.

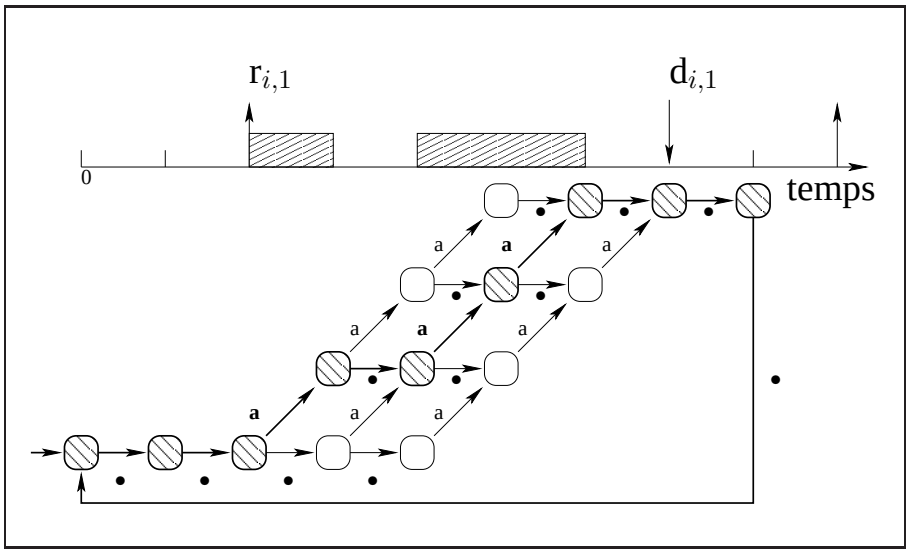

Figure 1. *Modèle de la tâche τ (r=2,C=3,D=5,T=7) avec visualisation d'une exécution particulière*

Valider temporellement une application temps-réel consiste, à l'instant t, à prouver qu'aucune tâche ne ratera d'échéance pendant tout le reste de la vie de l'application, c'est-à-dire prouver qu'il existe un chemin de longeur infinie dans l'automate modélisant l'application. Nous devons donc considérer uniquement les cycles de l'automate.

Nous utilisons ici la définition du centre d'un langage : le *centre* d'un langage L est l'ensemble des préfixes de L qui sont infiniment prolongeables dans L :

$$centre(L) = \{u \in L \mid \forall N \in \mathbb{N},\ \exists v \in \Sigma^*, |v| > N \ et\ u.v \in L\}.$$

REMARQUE. — $centre(L^*) = L^*.\mathit{Pref}(L)$ où $\mathit{Pref}(L)$ est l'ensemble des préfixes de L.

Pour une tâche τ, son exécution passée (avant l'instant t) est connue et représentée par un mot ω de la forme $\omega = \omega_1.v$, où ω_1 correspond à l'ensemble des instances finalisées et v au début de l'instance courante de τ. Le mot ω_1 appartient donc à $\bullet^r \left((\mathbf{a}^C \amalg \bullet^{D-C})\bullet^{T-D}\right)^*$. Le mot v est tel que $\exists\ \eta \in$ {a,•}*, $v.\eta \in (\mathbf{a}^C \amalg \bullet^{D-C})\bullet^{T-D}$, c'est-à-dire. l'instance courante peut terminer son exécution en accord avec les caractéristiques temporelles de τ. De plus, par construction, pour tout instant *t'* dans le futur (*t'* $\in$]*t*, ∞[), il existe un mot $\nu \in \left((\mathbf{a}^C \amalg \bullet^{D-C})\bullet^{T-D}\right)^*$ tel que $|\omega.\eta.\nu| > t'$. Ainsi, à chaque instant *t*, le passé ω de l'exécution de la tâche τ est un mot du centre du langage $\bullet^r \left((\mathbf{a}^C \amalg \bullet^{D-C})\bullet^{T-D}\right)^*$. Réciproquement, chaque mot du centre de ce langage est le passé d'une séquence d'exécution valide de la tâche.

Définition 1 *Nous appelons modèle temporel de la tâche τ (r,C,D,T), le langage :*

$$centre\left(\bullet^r \left((\boldsymbol{a}^C \amalg \bullet^{D-C})\bullet^{T-D}\right)^*\right).$$

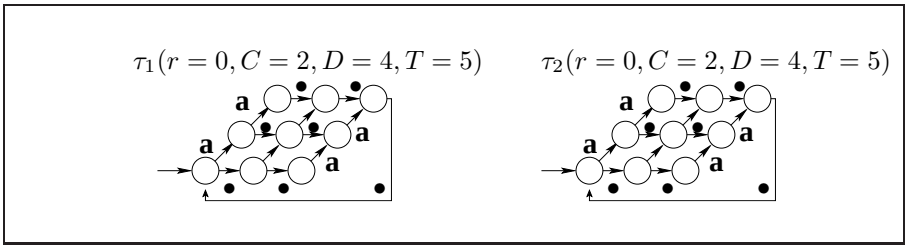

Figure 2. *Modèles temporels de deux tâches*

Dans la suite, nous nous basons sur l'exemple présenté en figure 2 pour illustrer nos définitions.

3.2. *Modélisation du fonctionnement simultané des tâches*

Les tâches composant une application temps-réel sont concurrentes. La modélisation de leurs comportements simultanés passe par l'utilisation du *produit homogène* des langages qui les modélisent.

L'opération de produit homogène, notée Ω, est définie par : soit A=< Σ,Q,{q_0},T,F> et A'=< Σ',Q',{q'$_0$},T',F'> deux automates où Σ et Σ' sont

des alphabets, Q et Q' les ensembles des états, $\{q_0\}\{q'_0\}$, les états initiaux, T et T' les ensembles d'états terminaux et F et F' les fonctions de transitions déterminant les arcs des automates.

Le produit homogène de ses deux automates est :A Ω A' = $< \Sigma \times \Sigma'$,Q",q"$_0$,T",F"$>$ tel que : Q" $\subset$ Q $\times$ Q' ; q"$_0$=(q$_0$,q'$_0$) ; T" = T $\times$ T ; F" est telle que : $\forall\ w{=}w_1...w_k \in$ L(A), $\forall\ w'{=}w'_1...w'_k \in$ L(A'), $w"=\binom{w_1}{w'_1}...\binom{w_k}{w'_k} \in$ L(A Ω A').

Le langage modélisant l'exécution simultanée de l'ensemble des tâches Γ de l'application est défini sur l'alphabet Σ^n ; une lettre est un n-uplet (x$_1$,...,x$_n$) où chaque composante x$_k$ représente l'activité de la tâche τ_k, c'est-à-dire, x$_k \in$ {a,$\bullet$} (cf. figure 3). Par exemple le triplet $(a, \bullet, a)$ représente une unité de temps durant laquelle les tâches τ_1 et τ_3 sont actives et la tâche τ_2 est inactive. La sémantique associée à l'opérateur Ω est donc la concurrence. Le langage modélisant le comportement simultané des tâches est alors défini par :

$$L(\Gamma) = \underset{i \in [1,n]}{\Omega} L(\tau_i).$$

La figure 3 présente le produit homogène des deux tâches de notre exemple d'application (cf. figure 2). Chaque transition est étiquetée par un couple dont la première composante correspond au comportement de la tâche τ_1 et la seconde à celui de τ_2. Les partages de ressources et de processeurs ne sont pas pris en compte, le produit homogène modélise uniquement le fonctionnement simultané des tâches. Un exemple d'exécution est présenté en gras sur la figure, le diagramme de Gantt associé est présenté en bas à gauche de la figure. Les dépendances entre tâches sont présentées dans la section suivante.

REMARQUE. — Le produit homogène permet de modéliser la concurence dans un système de tâche et permet également de connaître le parallélisme maximal d'une application : l'évaluation peut se faire en calculant le nombre moyen de tâches actives le long d'une séquence d'ordonnancement — *i.e.* un mot accepté par l'automate — puis en sélectionnant les séquences maximisant ce nombre moyen (de telles approches ont été menées par exemple dans (Thimonier, 1988)).

3.3. *La dépendance*

3.3.1. *Partage de ressources*

Pour modéliser le partage de ressources, nous utilisons la méthode définie par (Arnold *et al.*, 1982) qui consiste à tracer l'état de la ressource en utilisant un automate virtuel (cf. figure 4). Une ressource peut être dans deux états : soit elle est libre (état 1 de la figure 4), soit elle est utilisée par une tâche (état 2 de la figure).

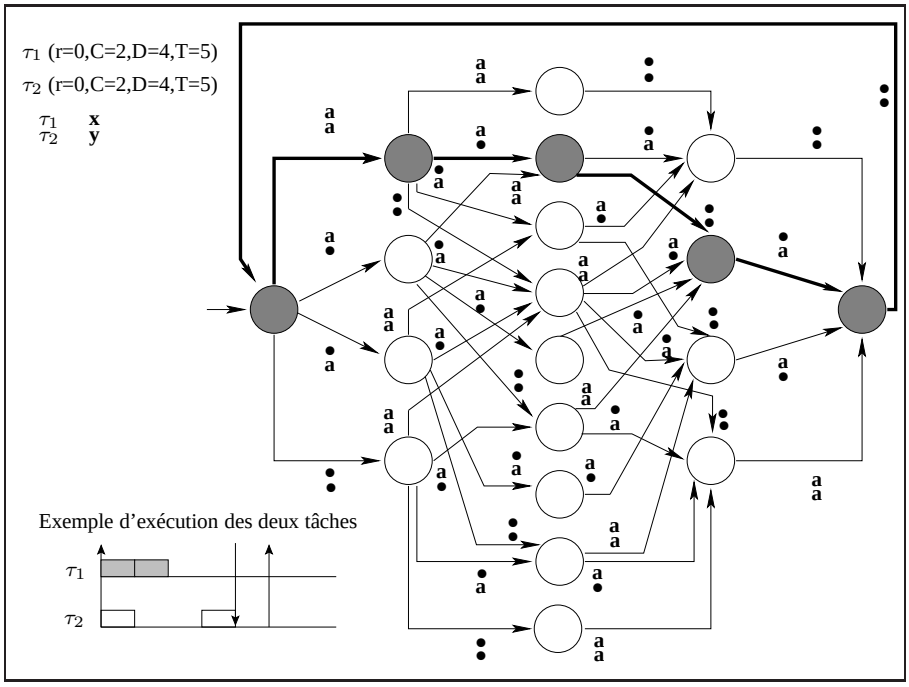

Figure 3. *Produit homogène des automates de deux tâches : pour la première unité de temps, chaque tâche peut ou non s'exécuter, il y a 4 possibilités et donc 4 transitions. Les transitions du produit homogène correspondent au produit des transitions de profondeur égale des deux automates*

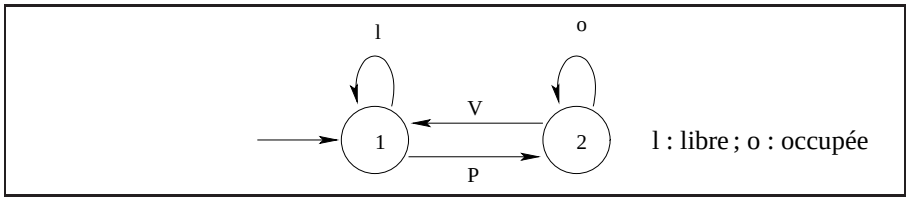

Figure 4. *Modèle de ressource à une seule instance*

Nous commençons par raffiner notre observation du code de la tâche en complétant notre alphabet pour tenir compte des instructions critiques (notées P_c) : l'alphabet {**a**,•} devient {**a**,•}$\cup P_c$ ={**a**,$P(R_1), V(R_1), ..., P(R_r), V(R_r)$,•} où P est associé à l'instruction élémentaire « Prendre », V à l'instruction élémentaire « Vendre » et où r est le nombre de ressources partagées. Nous associons à chaque ressource critique R_j, $j \in [1, r]$, une tâche virtuelle V_{R_j} (et un langage rationnel L(V_{R_j})) qui modélise un processus virtuel : l'état de la ressource. Nous associons au système $\Gamma = (\tau_i)_{i \in [1,n]}$

le produit homogène des L(τ_i) et des L(V_{R_j}) pour obtenir les états des tâches et des ressources. Soit $\mathrm{L}(\Gamma, R)$ ce langage :

$$L(\Gamma, R) = \Big(\underset{i=1}{\overset{n}{\Omega}} L(\tau_i) \Big) \Omega \Big(\underset{i=1}{\overset{r}{\Omega}} L(V_{R_i}) \Big).$$

Le langage $\mathrm{L}(\Gamma, R)$ collecte l'ensemble des comportements des tâches et des ressources sans tenir compte des contraintes (d'exclusion mutuelle ou de nombre de processeurs). Nous considérons, pour notre exemple (cf. figure 3), que les deux tâches utilisent la même ressource pendant toute leur exécution.

Nous utilisons alors les contraintes induites par le partage de ressources. Ces contraintes vont invalider certaines étiquettes de transition du produit homogène $L(\Gamma, R)$ suivant que le comportement des tâches correspond à une utilisation correcte des ressources ou non. Dans notre exemple, les étiquettes valides sont les suivantes :

$$S_R = \left\{ \begin{matrix} \tau_1 \\ \tau_2 \\ res \end{matrix} \begin{pmatrix} P \\ \bullet \\ P \end{pmatrix}, \begin{pmatrix} \bullet \\ P \\ P \end{pmatrix}, \begin{pmatrix} V \\ \bullet \\ V \end{pmatrix}, \begin{pmatrix} \bullet \\ V \\ V \end{pmatrix}, \begin{pmatrix} \bullet \\ \bullet \\ l \end{pmatrix}, \begin{pmatrix} \bullet \\ \bullet \\ o \end{pmatrix} \right\}.$$

D'une part, nous avons le produit homogène qui collecte l'ensemble des comportements des tâches sans tenir compte des ressources et d'autre part nous avons l'ensemble $S_R{}^*$ qui collecte l'ensemble des utilisations valides des ressources sans tenir compte des tâches. L'intersection des deux ensembles collecte l'ensemble des comportements des tâches valides du point de vue de la ressource.

Soit $\pi_{[1,n]}$ la projection permettant de conserver uniquement les composantes correspondant aux tâches (c'est-à-dire. qui *efface* les composantes liées aux ressources virtuelles) : $\pi_{[1,2]} \begin{pmatrix} P \\ \bullet \\ P \end{pmatrix} \begin{matrix} \tau_1 \\ \tau_2 \\ ress \end{matrix} = \begin{pmatrix} P \\ \bullet \end{pmatrix} \begin{matrix} \tau_1 \\ \tau_2 \end{matrix}$. Le langage collectant l'ensemble des comportements valides de l'application est le suivant :

$$L_R(\Gamma) = \pi_{[1,n]} \Big(centre \Big(L(\Gamma, R) \cap S_R^* \Big) \Big).$$

Pour notre exemple, l'automate résultant du produit et de la synchronisation avec l'automate de la ressource est présenté figure 5a. Les états grisés n'appartiennent à aucun cycle, il ne sont pas dans l'automate associé au centre du langage. L'automate associé à ce langage est présenté figure 5b. Cet automate collecte l'ensemble des comportements valides de l'application du point de vue du partage de la ressource.

L'utilisation conjointe des automates traçant les états des ressources et du filtrage des étiquettes valides permet donc de garantir une utilisation correcte (respectant l'exclusion mutuelle) des ressources.

3.3.2. *Partage de processeurs*

Dans la spécification de l'architecture sur laquelle doit s'exécuter l'application, le nombre de processeurs est généralement inférieur au nombre de tâches. Dans le cas contraire, il n'y a pas de problème d'ordonnancement : chaque tâche est attribuée à un processeur.

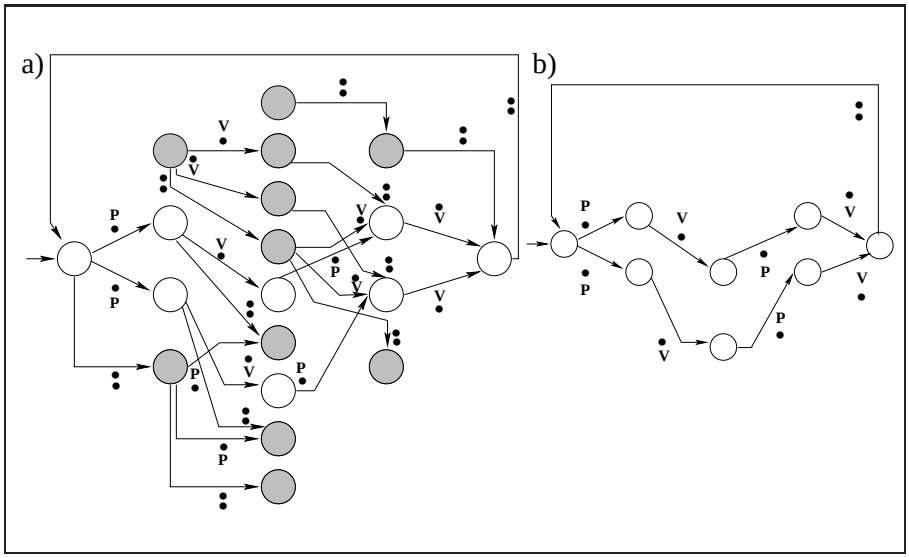

Figure 5. *Comportements valides du point de vue du partage de ressource : après suppression des transitions invalides, les états sans prédecesseur ou successeur (en gris) sont supprimés puisqu'ils sont soit non atteignables soit aboutissent à un dépacement d'échéance*

A tout instant, le nombre de tâches actives doit être inférieur ou égal au nombre de processeurs. Cela signifie qu'un vecteur X=(x_1,...,x_n) de Σ^n n'est valide du point de vue du partage de processeurs que si le nombre de composantes associées à une utilisation d'un processeur (lettres **a**, P et V) est inférieur ou égal au nombre de processeurs. Ainsi la lettre $\begin{pmatrix} \mathbf{a} \\ \bullet \\ \mathbf{a} \end{pmatrix}$ sera valide dans le cas biprocesseur mais pas dans le cas monoprocesseur.

Soit p le nombre de processeurs. Nous notons $|X|_{\mathbf{a}}$ le nombre d'occurences de la lettre a dans le n-uplet X et S_p l'ensemble des lettres valides : S_p={X $\in \Sigma^n$ / $|X|_{\mathbf{a}}+|X|_{\mathbf{P}}+|X|_{\mathbf{V}} \leq p$}.

Le langage modélisant l'ensemble de tâches Γ et prenant en compte le nombre de processeurs p est l'ensemble des mots de L(Γ) qui ne contiennent que des lettres de S_p :

$$L(\Gamma) \cap S_p^*.$$

L'opération d'intersection n'est pas interne dans la classe des centres de langages rationnels, le modèle temporel $L(\Gamma, p)$ de l'application fonctionnant sur une architecture ayant p processeurs est donc le langage suivant :

$$L(\Gamma) \cap S_p^* = \left[centre(L(\Gamma) \cap S_p^*)\right] \cup \left[L(\Gamma) \cap S_p^* \setminus centre(L(\Gamma) \cap S_p^*)\right].$$

Les mots de la composante $\left[centre(L(\Gamma) \cap S_p^*)\right]$ correspondent aux comportements valides, dans le sens où ils sont prolongeables aussi loin que l'on veut dans le respect des échéances (par définition de l'opération *centre*). A l'opposé, les mots du langage $\left[L(\Gamma) \cap S_p^* \setminus centre(L(\Gamma) \cap S_p^*)\right]$ correspondent aux comportements dont on est certain qu'ils finiront pas manquer une échéance (puisqu'ils ne sont pas infiniment prolongeables dans le respect des échéances). Le sous-ensemble de $L(\Gamma) \cap S_p^*$ qui comporte exactement les comportements valides au sens où nous l'entendons ici est donc $centre(L(\Gamma) \cap S_p^*)$. C'est pourquoi nous posons :

$$L(\Gamma, p) = centre\left(L(\Gamma) \cap S_p^*\right).$$

3.4. *Validation*

Les partages de ressources et de processeurs peuvent amener les tâches à rater leurs échéances. Nous montrons ici que le modèle basé sur les langages rationnels permet de décider de l'ordonnançabilité d'une configuration Γ partageant des ressources.

Soit S=$S_R \cup S_p$ l'ensemble des vecteurs valides du point de vue du partage de ressources et de processeurs. Nous notons $\prod_{1,n}^{1,r}(\Gamma)$, le *produit synchronisé*, de la manière suivante :

$$\prod_{1,n}^{1,r}(\Gamma) = \pi_{[1,n]}\left(\left(\overbrace{\left(\underset{i=1}{\overset{n}{\Omega}} L(\tau_i)\right)}^{\text{tâches}} \Omega \overbrace{\left(\underset{i=1}{\overset{r}{\Omega}} L(V_{R_i})\right)}^{\text{ressources}}\right) \cap S^*\right).$$

Le prédicat permettant de décider de la validité de Γ est $centre\left(\prod_{1,n}^{1,r}(\Gamma)\right) \neq \emptyset$. De plus, nous avons proposé une méthode de modélisation des systèmes de tâches à durées variables (Geniet *et al.*, 2001) et distribués (Largeteau *et al.*, 2002). Pour le calcul effectif d'un automate d'acceptation de $\prod_{i=1}^{i=n}(\Gamma)$, nous avons mis en place une méthode basée sur l'entrelacement de calculs de produits synchronisés d'automates et d'automates d'acceptation de centres de langages rationnels. (Geniet *et al.*, 2001) établissent l'efficacité de cette stratégie comparativement aux alternatives envisageables sur la base d'une campagne d'expérimentation.

Dans le contexte d'ordonnancement que nous avons choisi d'étudier, le problème est NP-complet. La complexité des algorithmes connus est donc exponentielle et on suppose qu'il n'y en a pas de meilleurs. Un parcours d'automate tel que nous en utilisons lors des calculs de produits homogènes est de complexité O (*nombre d'états* + *nombre de transitions*). Le nombre d'états d'un automate de tâche est $r + C \times (D - C) + (T - D)$, le nombre de transitions est du même ordre. La complexité du produit homogène est O ((*nombre d'états* + *nombre de transitions*)^n). Nous effectuons des parcours complets de la structure pour chaque opération (nous calculons $n + R$ produits) et les temps de calculs observés vont jusqu'à quelques jours pour de petites applications (environ 15 tâches pour les systèmes embarqués usuels).

3.5. *Considérations géométriques sur les automates manipulés par MARTA*

Le modèle MARTA est utilisable dans une optique d'analyse des séquences valides, lorsqu'il y en a. L'automate fini intégrant l'ensemble de ces séquences, un certain nombre de propriétés peuvent en être extraites, concernant aussi bien des approches d'ordonnancement en ligne (puisque les séquences ED, RM, etc. sont des cas particuliers des séquences engendrables à partir de l'automate) que des approches hors-ligne (*via* un séquenceur).

Bien qu'ayant une très grande puissance d'expression, ce modèle ne permet pas de positionner le système étudié par rapport au « plus proche système valide », puisque lorsqu'il n'y a pas ordonnançabilité, l'automate obtenu accepte le langage vide. Nous avons donc cherché à l'adapter, en conservant autant que possible les parties invalides. Notre idée était qu'une partie valide mais inatteignable de l'automate est intéressante du point du vue du nombre d'états invalides qui permettaient de l'atteindre mais qui ont été supprimés.

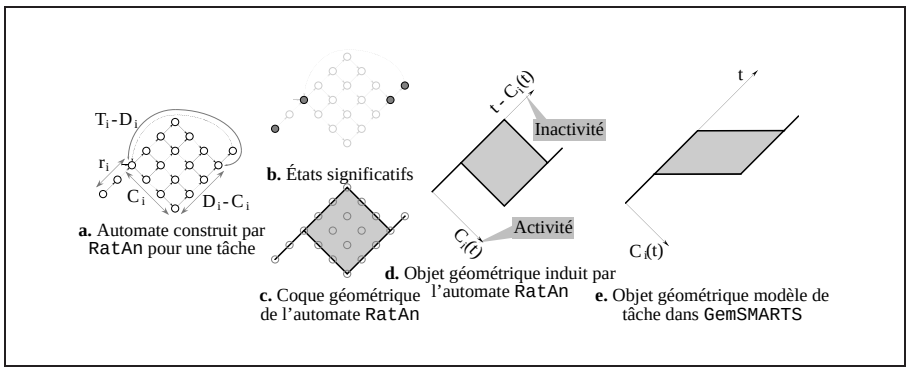

Figure 6. *Le modèle* GRETA *: un raffinement géométrique des automates utilisés par* RatAn

Par ailleurs, l'observation de la structure des automates finis associés aux tâches (voir figure 6a) met en évidence l'inefficacité de l'énumération de l'ensemble des états : bien que cet automate soit minimal, quatre états suffiraient à caractériser l'ensemble de son graphe, au lieu des $(D_i - C_i)\, C_i + T_i - D_i + R_i$ qu'il contient effectivement (voir figure 6b). Nous avons donc cherché à réduire la complexité spatiale des automates, en n'en construisant que l'enveloppe (figure 6c), ce qui nous a assez naturellement conduit vers un raffinement géométrique du modèle, dans lequel les notions de mot, d'état (et par là même d'automate) disparaissent totalement (figure 6d) : les automates des tâches sont injectés dans $\mathbb{R}^2$, la sémantique associée aux axes du repère étant respectivement l'activité et l'inactivité de la tâche. Pour avoir accès au temps écoulé dans cet espace, nous devons le reconstruire en fonction des coordonnées. Afin d'éviter ce calcul inutile, nous avons effectué un changement de repère (figure 6e) qui nous donne finalement la représentation de l'automate dans l'espace (temps écoulé, avancement de la tâche).

Dans cet espace, un chemin dans l'automate (un mot) est représenté par une courbe continue (figure 7). L'objet de la section suivante est, connaissant la représentation géométrique des automates des tâches, de traduire les opérations appliquées aux automates en opérations géométriques équivalentes. Ainsi, nous associons au produit homogène d'automates un produit cartésien « adapté » permettant d'obtenir la représentation géométrique de l'ensemble des comportements concurrent du système. De même, une intersection géométrique traduit l'intersection de langage permettant de représenter les partages de ressources.

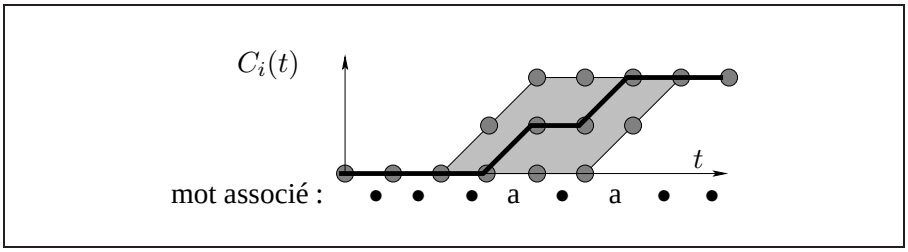

Figure 7. *Représentation géométrique d'un mot*

4. Vers un modèle adapté à l'évaluation : le modèle GRETA

4.1. *Modèle de tâche*

Dans MARTA, nous avons la possibilité d'observer une tâche lors de ses séjours dans les états. Le modèle temporel est donc discret : on associe à un mot ω la durée $|\omega|$, qui est discrète.

Dans le modèle GRETA, nous nous positionnons sur un espace des temps continu : un état du modèle est un couple (t, c), où t est le temps écoulé depuis l'origine (démarrage de l'application) et c le temps CPU capitalisé par la tâche observée. L'observation de l'état de la tâche peut donc être effectuée pour tout $t \in \mathbb{R}^+$. Considérons maintenant notre tâche sur l'intervalle de temps $[0, t[$. L'ensemble $\{(u, C(u)), u \in [0, t[\}$ des états par lesquels elle passe tout au long de cet intervalle constitue une trajectoire continue (Lay, 2005). En effet, cette trajectoire n'est rien d'autre que le graphe de la fonction $t \rightarrow C(t)$ (où $C(t)$ est le temps CPU capitalisé par la tâche depuis l'origine des temps). Or d'une part, la valeur $C(t)$ existe pour tout $t \geq 0$, et d'autre part, la propriété $\forall t_0 > 0,\ \forall \epsilon > 0, \exists \eta > 0$ telle que $|t - t_0| < \eta \Rightarrow |C(t) - C(t_0)| < \epsilon$ est acquise dès que l'on prend, par exemple $\eta = \epsilon$, car la tâche ne peut se voir attribuer plus de temps CPU sur un processeur que le temps absolu qui s'est écoulé entre les instants t et t_0. Ces deux propriétés établissent le fait que la trajectoire de la tâche est définie et continue sur $\mathbb{R}^+$.

Considérons maintenant un mot ω décrivant un comportement de la tâche dans le modèle MARTA, et caractérisons une trajectoire GRETA « équivalente » (dans le sens où elle modélise le même comportement).

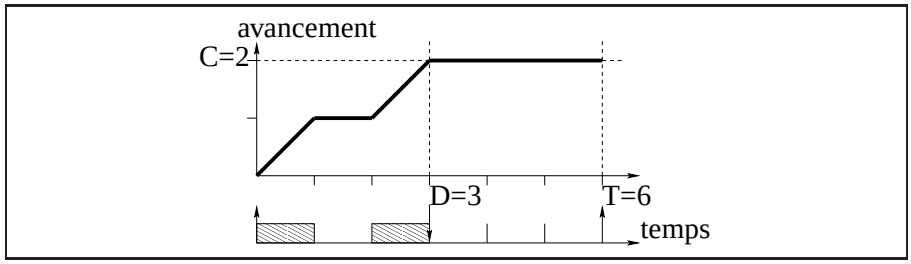

Figure 8. *Une trajectoire de la tâche τ (r=0,C=2,D=3,T=6)*

Dans ω, ω_1 représente l'état de la tâche pendant la première unité de temps, c'est-à-dire sur l'intervalle de temps $[0, 1[$. ω_2 représente l'état de la tâche sur l'intervalle $[1, 2[$, et ainsi de suite. Donc, ω_i représente l'état de la tâche sur l'intervalle de temps $[i-1, i[$.

Considérons un instant entier $t \in [0, |\omega|[\cap \mathbb{N}$. La charge CPU cumulée par la tâche à l'instant t vaut $C(t) = |\omega_{1..t-1}|_a$. Cette valeur est nécessairement entière. Une trajectoire GRETA « équivalente » à ω passe donc nécessairement par l'état $(t, C(t))$. Considérons maintenant l'état de la tâche à l'instant $t+1$. Deux cas sont possibles : soit $\omega_t = \bullet$, soit $\omega_t = a$. Dans le premier cas, la tâche ne dispose jamais d'un processeur sur l'intervalle $[t, t+1[$. On a donc $C(t+1) = C(t)$, et toute trajectoire T « équivalente » à ω est constante sur l'intervalle $[t, t+1[: \forall u \in [t, t+1[, (u, C(t)) \in T$. Dans le second cas, la tâche dispose de manière permanente d'un processeur sur l'intervalle $[t, t+1[$. Nous avons donc $\forall u \in [t, t+1[, (u, C(t)+u-t) \in T$. Notons que la continuité de la trajectoire en $t+1$ établit bien le fait que $(t+1, C(t)+1) \in T$, ce qui correspond bien au point $(t+1, C(t+1)$ obtenu en analysant ω à l'instant $t+1$.

L'ensemble de ces remarques permet d'identifier de manière unique la trajectoire T équivalente à ω.

Définition 2 *On appelle trajectoire valide de τ (r, C, D, T) toute trajectoire de τ ayant les propriétés suivantes (cf. figure 8) :*

$$
\begin{array}{lllll}
Tr_\tau V & : & \mathbb{R}^+ & \rightarrow & \mathbb{R}^+ \\
 & \forall x \in [0, r_1] & Tr_\tau V(x) & = & 0 \\
 & \forall x \in [r_i, r_i + D] & Tr_\tau V(x) & \in & [(i-1) \times C, i \times C] \\
 & \forall x \in [r_i + D, r_{i+1}] & Tr_\tau V(x) & = & i \times C
\end{array}
$$

Chaque exécution de la tâche en dehors de l'intervalle $[r_i, r_i + D]$ constitue une faute temporelle, c'est-à-dire. un non respect des contraintes temporelles de la tâche, c'est pourquoi les applications $Tr_\tau V$ sont constantes sur les intervalles $[0, r]$ et $[r_i + D, r_{i+1}]$. La valeur $Tr_\tau V(r_i + D) = i \times C$ garantit l'exécution de la totalité du code de l'instance i au plus tard à son échéance $r_i + D$ et ce pour chaque instance i de la tâche. Il n'y a donc pas de faute temporelle dans la séquence d'exécution

de τ associée à une trajectoire de cette forme. On appelle $T_V(\tau)$ l'ensemble des trajectoires valides pour τ.

Définition 3 *Le domaine de validité $\Omega(\tau)$ des états d'une tâche τ est :*

$$\Omega(\tau) = \bigcup_{\psi \in T_V(\tau),\ t \in \mathbb{R}^+} \{(t, \psi(t))\}.$$

L'ensemble $\Omega(\tau)$ associé à τ est l'ensemble des points faisant partie d'une trajectoire valide de τ, c'est-à-dire. chaque point est une représentation géométrique d'un état temporellement valide et atteignable de τ (cf. figure 9).

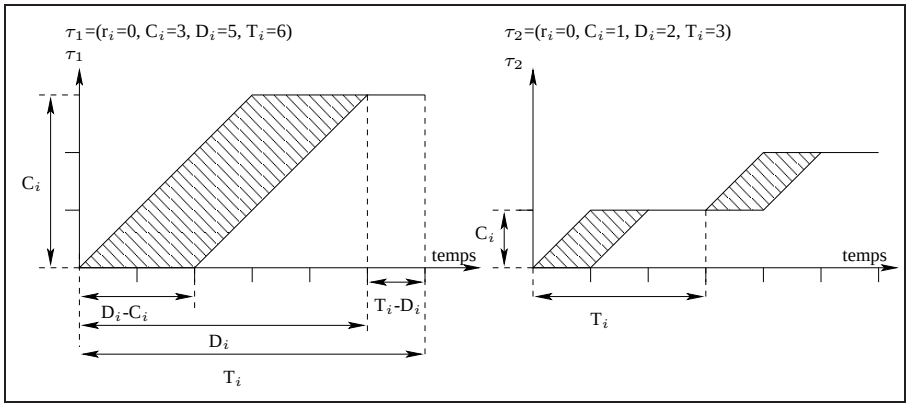

Figure 9. *Enveloppe de l'ensemble des états valides de deux tâches*

On définit la fonction T qui à un ensemble de points $\Omega(\tau)$ associe l'ensemble des trajectoires valides de τ : $T(\Omega(\tau)) = T_V(\tau)$.

Cette définition peut être généralisée à un ensemble de n tâches : pour $\Omega \subset \mathbb{R}^{n+1}$, la fonction T associe l'ensemble des trajectoires valides de Ω.

$$T(\Omega) = \{\psi\ : \mathbb{R} \to \mathbb{R}^n,\ continue,\ valide / \forall t \in \mathbb{R},\ (t, \psi(t)) \in \Omega\}.$$

4.2. *Un opérateur géométrique pour la concurrence*

Le fonctionnement simultané des tâches d'un système Γ définit le fonctionnement de ce système. L'état du système Γ à un instant t est donc défini par l'état d'avancement de chacune des tâches τ_i à cet instant t. De plus, la validité d'un état à l'instant t du système dépend de la validité de l'état de chaque tâche de ce système à l'instant t.

Définition 4 *Un état valide d'un système Γ à un instant t est défini par :*

$$P = (t, x_1, ..., x_n) / \forall i \in [1, n],\ (t, x_i) \in \Omega(\tau_i).$$

En effet, un état du système n'est valide que si toutes les tâches qui le composent sont dans un état valide (voir définition 3). L'espace des points représentant les états du système a donc $n+1$ dimensions.

Définition 5 *L'ensemble des états valides d'un système de tâches* Γ=$(\tau_i)_{i\in[1,n]}$ *est l'ensemble* $\Omega(\Gamma)$ *défini par :*

$$\Omega(\Gamma) = \{P = (t, x_1, ..., x_n) / \ (t, x_i) \in \Omega(\tau_i)\}.$$

L'ensemble des trajectoires de $\Omega(\Gamma)$ est $T(\Omega(\Gamma))$. Une trajectoire $Tr_\Gamma V \in T(\Omega(\Gamma))$ valide pour l'ensemble des tâches du système Γ peut être définie à partir des trajectoires valides des tâches de Γ :

$$\begin{array}{lccl} Tr_\Gamma V : & \mathbb{R}^+ & \rightarrow & \mathbb{R}^{n+1} \\ & t & \rightarrow & Tr_\Gamma V(t) = (\psi_1(t), ..., \psi_n(t)), \ \forall i \in [1, n], \ \psi_i \in T(\Omega(\tau_i)). \end{array}$$

Le « produit intersecté de tâches »

Dans la suite, nous montrons que $\Omega(\Gamma)$, modélisant l'ensemble des comportements simultanés des tâches (sans tenir compte des contraintes liées aux ressources ou au nombre de processeurs), peut être obtenu par application d'opérations géométriques sur les modèles de tâches qui constituent l'équivalent géométrique du produit homogène des automates.

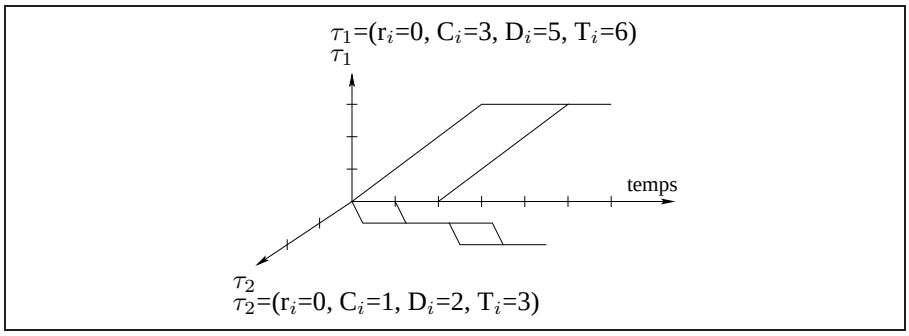

Figure 10. *Deux tâches placées dans l'espace 3D correspondant*

Pour l'exemple montré en figure 10, l'espace dans lequel est modélisée la concurrence pour deux tâches est à 3 dimensions. Dans le plan (temps,τ_1), nous voyons la représentation géométrique d'une instance de τ_1 et dans le plan (temps,τ_2), la représentation géométrique de deux instances de τ_2.

Chaque objet dans un plan (temps, τ_i) ne modélise que les états valides de la tâche i. Nous devons donc passer d'un état (t, x_i) de ce plan à un état $(t, x_1, ..., x_i, ..., x_n)$

de l'espace de l'application. Pour cela nous utilisons un produit cartésien qui conserve l'ordre des coordonnées (la coordonnée associée à la tâche τ_i doit se trouver à la $(i+1)^{\text{e}}$ position). Nous appelons cette opération le *produit cartésien injecté*.

Définition 6 *Le produit cartésien injecté $\mathcal{J}_{i,n}$ de l'espace $(t, x_i) = \mathbb{R}^2$ associé à la tâche τ_i avec $\mathbb{R}^n$ est l'application :*

$$\begin{aligned} \mathcal{J}_{i,n} : \quad \mathbb{R}^2 &\rightarrow \quad \mathcal{P}(\mathbb{R}^{n+1}) \\ (t, x_i) &\rightarrow \quad \{(\boldsymbol{t}, y_1, ..., y_{i-1}, \boldsymbol{x}_i, y_i, ..., y_{n-1}), \forall j \in [1, n-1]\ y_j \in \mathbb{R}\} \end{aligned}$$

REMARQUE. — la projection Π_i : (y_1,...,y_{i+1},...,y_{n+1}) $\rightarrow$ (y_1,y_{i+1}) est l'inverse à gauche de $\mathcal{J}_{i,n}$: $\Pi_i(\mathcal{J}_{i,n}(a,b))=\{(a,b)\}$.

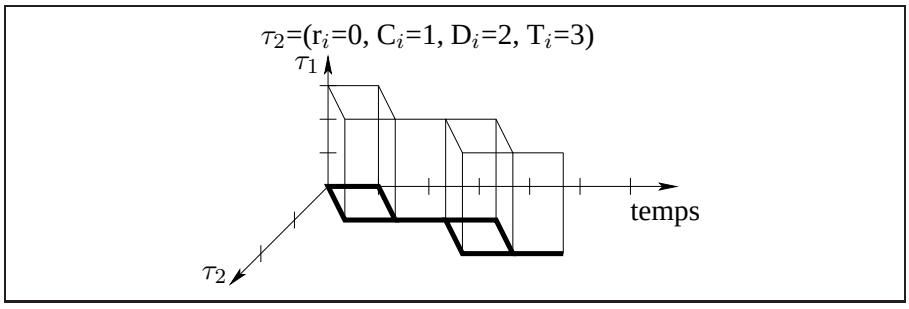

Figure 11. *Produit cartésien injecté : $\mathcal{J}_{2,2}(\Omega(\tau_2))$ qui correspond à une extrusion selon l'axe de τ_1*

La figure 11 fournit une représentation graphique du produit cartésien injecté de la deuxième tâche d'une application comportant deux tâches (n = 2). Cet objet contient tous les états de l'application tels que la tâche τ_2 soit dans un état valide et que la tâche τ_1 soit dans un état quelconque. Pour chaque tâche on peut construire un tel objet O_i. Un état valide de l'application est un état pour lequel chaque tâche est dans un état valide. L'objet associé à l'ensemble des états valides de l'application est donc l'intersection de tous les O_i associés aux tâches.

$\Omega(\tau_1,\tau_2)=\Omega(\tau_1) \otimes \Omega(\tau_2) = \mathcal{J}_{1,n}(\Omega(\tau_1)) \cap \mathcal{J}_{2,n}(\Omega(\tau_2))= O_1 \cap O_2$. Cette propriété se généralise par associativité de l'opérateur Ω :

Théorème 1
$$\Omega(\Gamma) = \bigotimes_{i=1}^{i=n} \Omega(\tau_i).$$

Ce théorème donne directement un algorithme de construction de $\Omega(\Gamma)$. La figure 12 montre différentes étapes et vues de la construction de l'ensemble des états valides de la configuration Γ. L'algorithme que nous avons implanté consiste

à calculer dans la même opération les extrusions (temps constant) et les intersections (linéaire en le nombre de points) en parcourant en même temps les n objets associés aux n tâches (n tests par point). La complexité de cet algorithme est en $O(n \times P \times (max(C_i))^n)$. Le gain est donc très appréciable et permet d'obtenir des temps de réponse de l'ordre d'une dizaine de secondes là où MARTA nécessite quelques heures.

REMARQUE. — le théorème implique également que les points de $\Omega(\tau_1, \tau_2) \cap \mathbb{N}^3$ sont en bijection avec les états de l'automate résultant du produit homogène A(τ_1) Ω A(τ_2). En effet, considérons un mot du produit homogène ω. Ce mot est de la forme $\omega = \binom{\omega_1}{\omega_2}$ où ω_1 (resp. ω_2) est un mot associé à un comportement de la tâche τ_1 (resp. τ_2). Le mot ω_1 (resp. ω_2) est équivalent à une trajectoire valide $Tr_{\tau_1}V$ (resp. $Tr_{\tau_2}V$). En utilisant le produit injecté à partir de ces deux trajectoires nous construisons une trajectoire du système : $\{(t, x_1, x_2) \mid (t, x_1) \in Tr_{\tau_1}V \ et \ (t, x_2) \in Tr_{\tau_2}V\}$. Cette trajectoire est équivalente au mot ω puisqu'elle est équivalente à ω_1 **et** ω_2. C'est pourquoi nous avons choisi une notation similaire pour ces deux résultats.

4.3. *Intégration de l'interdépendance : partage de ressources*

Nous devons maintenant prendre en compte les conflits engendrés par le partage de resources critiques (en exclusion mutuelle gérées par des sémaphores) : une seule tâche peut utiliser la ressource R à un instant t. Certains états de $\Omega(\Gamma)$ sont alors invalides du point de vue du partage de ressource. D'un point de vue géométrique, la $k^{\text{è}}$ instance d'une tâche τ_i correspond à l'intervalle $[C_i \times k, C_i \times k + C_i]$ sur l'axe « temps CPU » de cette tâche. Nous définissons $u_i^k(1)$ et $u_i^k(2)$ tels que cette instance utilise la ressource R dans l'intervalle : $]u_i^k(1), u_i^k(2)[\subset [C_i \times k, C_i \times k + C_i]$. L'instant $u_i^k(1)$ correspond au début de l'instruction « Prendre » et $u_i^k(2)$ correspond à la fin de l'instruction « Vendre » (cf. figure 13), $u_i^k(2) - u_i^k(1)$ donne donc la durée de la section critique.

Nous notons u_i l'union $\bigcup_{k \in \mathbb{N}}]u_i^k(1), u_i^k(2)[$ qui collecte toutes les sections temporelles (sections critiques) durant lesquelles τ_i utilise la ressource R (cf. figure 13).

Etant donné qu'une tâche ne peut prendre une ressource si celle-ci est déjà utilisée, deux tâches ne peuvent être en section critique en même temps. Du point de vue du partage de ressource, à un instant t, un état valide P$= (t, x_1, ..., x_n)$ du système de tâches Γ satisfait donc la propriété :$|\{x_i / x_i \in u_i\}| \leq 1$.

Un tel point est associé à un état pour lequel au plus une tâche utilise la ressource R ; ainsi, les points P correspondent à des états valides de Γ en ce qui concerne le partage de ressource.

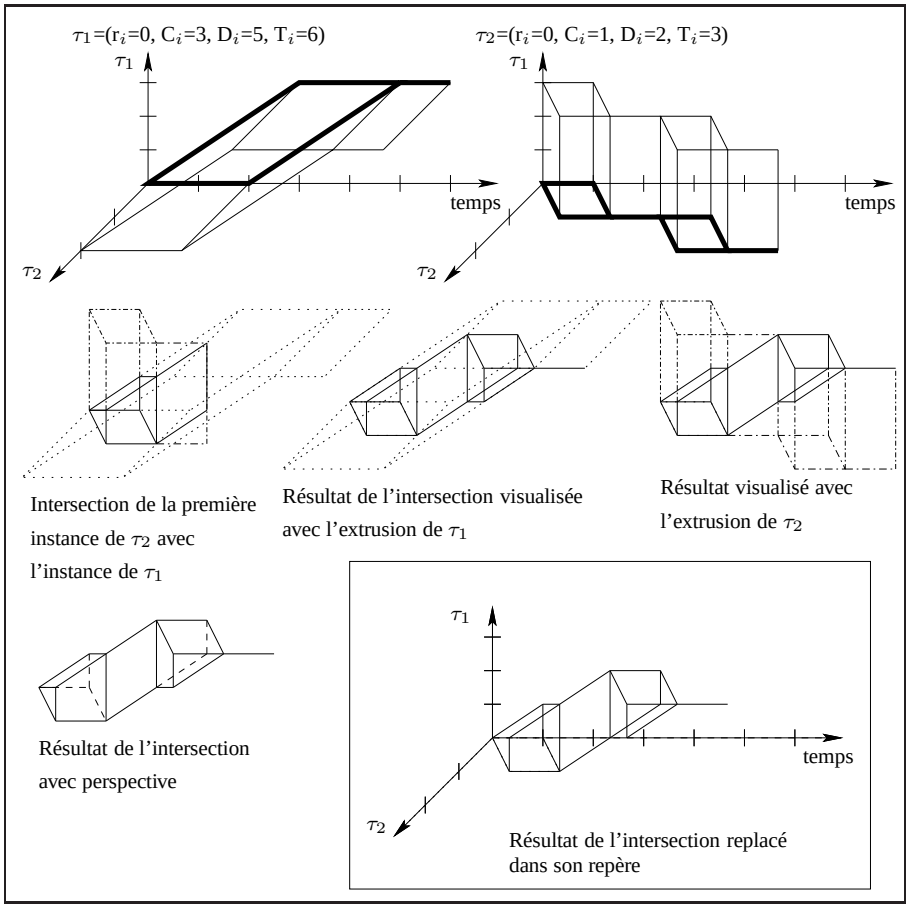

Figure 12. *Extrusions et intersection : modélisation de la concurrence. Le résultat de cette opération est l'ensemble des points représentants les états valides du système. Ces points sont en bijection avec les états de l'automate représentant le fonctionnement simultané des tâches*

L'ensemble des états valides de Γ du point de vue du partage de ressource est :

$$\Omega_R(\Gamma) = \{(t, x_1, ..., x_n) \in \Omega(\Gamma) / |\{x_i / x_i \in u_i\}| \leq 1\}.$$

Nous nous intéressons ici à une seule ressource notée R. Soit $\{\tau_j\}_{j \in \mathcal{I}_R}$ l'ensemble des tâches devant utiliser la ressource R (nous notons $\mathcal{I}_R$ l'ensemble des indices de ces tâches). Les états du sous-sytème Γ_R=$\{\tau_i\}_{i \in \mathcal{I}_R}$ associés à des utilisations simultanées de R sont invalides. L'objet associé à la ressource R (correspondant aux utilisations incorrectes de R) est inclus dans l'hyperplan formé par les tâches de Γ_R, puisqu'une mauvaise utilisation de R implique l'exécution simultanée d'au moins deux tâches de cet ensemble (cf. figure 13).

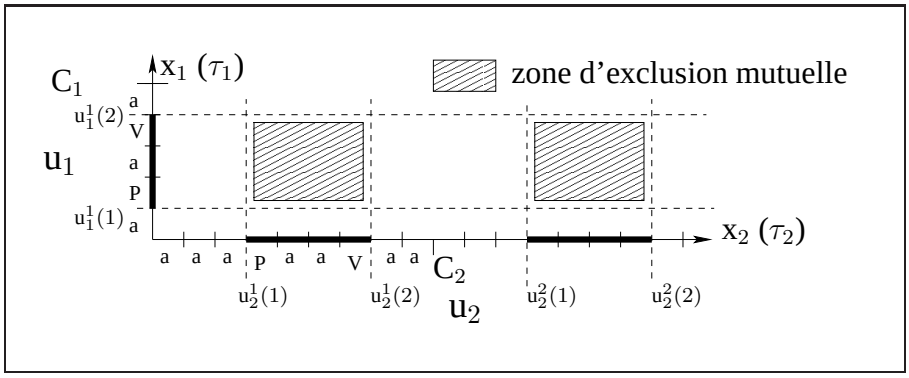

Figure 13. *Exemple : τ_1 et τ_2 utilisent la ressource R*

Définition 7 *La zone d'exclusion de R est (cf. figure 13) :*

$$\Omega(R) = \{(x_i)_{i\in\mathcal{I}_R} / \ |\{x_i / x_i \in u_i\}| > 1\}.$$

Cette zone d'exclusion ne tient pas compte des tâches qui n'utilisent jamais la ressource R. Nous définissons donc, à partir de l'ensemble $\Omega(R)$, l'ensemble $\Omega(\Gamma, R)$ des états invalides de l'ensemble des tâches de l'application (cf. figure 14b) :

$$\Omega'(R) = \{(x_i)_{i\in[1,n]} / (x_j)_{j\in\mathcal{I}_R} \in \Omega(R)\}.$$

Nous définissons enfin η_R l'ensemble des états invalides (du point de vue du partage de ressource) pour chaque instant t (cf. figure 14c) :

$$\eta_R = \{(t, x_1, ..., x_n) / t \in \mathbb{R}, \ |\{x_i / x_i \in u_i\}| > 1\}.$$

L'ensemble η_R est composé d'états pour lesquels au moins deux tâches utilisent la ressource en même temps et donc ne respectent pas l'exclusion mutuelle. La projection d'un état de η_R dans l'espace formé par les tâches de Γ_R appartient donc à la zone d'exclusion $\Omega(R)$ de R. On a donc :

$$\eta_R = \{(t, x_1, ..., x_n) / t \in \mathbb{R}, (x_i)_{i\in[1,n]} \in \Omega'(R)\} = \textit{extrusion}(temps, \Omega'(R)).$$

Cette dernière définition fournit directement un algorithme de construction de η_R et de l'ensemble $\Omega_R(\Gamma)$ (cf. figure 15) des états valides du système Γ en tenant compte des contraintes liées au partage de la ressource R.

L'application qui à un système Γ fait correspondre l'ensemble des trajectoires de Γ respectant l'exclusion mutuelle est appelée T :

$$T(\Omega_R(\Gamma)) = \{\psi \ : \mathbb{R} \to \mathbb{R}^n, continue \ / \ \forall t \in \mathbb{R}^+, \ (t, \psi(t)) \in \Omega_R(\Gamma)\}.$$

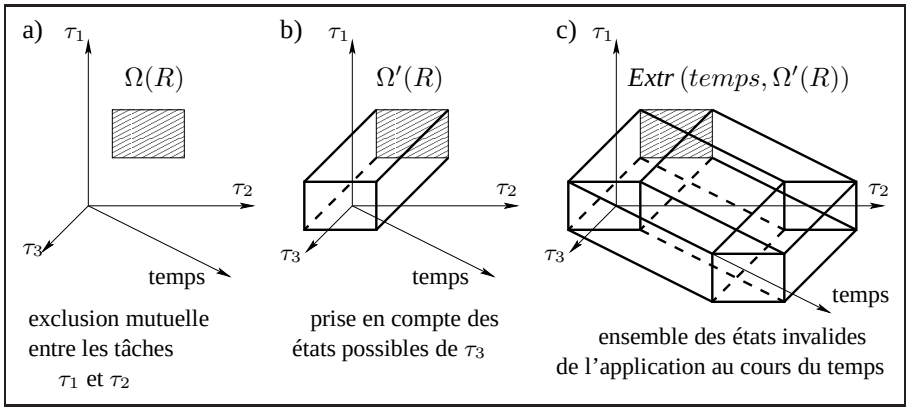

Figure 14. *Construction de η_R pour trois tâches dont deux partagent une ressource*

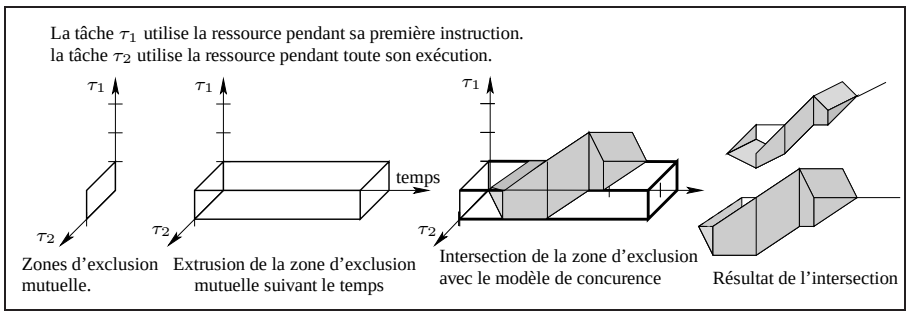

Figure 15. *Partage de ressources*

4.4. *Partage de processeurs : une nouvelle notion de voisinage*

Pendant l'étude de la construction de $\Omega_R(\Gamma)$, nous n'avons pas pris en compte le nombre de processeurs. Pour intégrer le partage de processeurs, nous devons connaître, pour chaque trajectoire de $T(\Omega_R(\Gamma))$, le nombre de processeurs nécessaires à son exécution. Nous avons donc à étudier le comportement des tâches entre deux commutations successives. En effet, une tâche ne peut commencer ou terminer son exécution qu'à un instant de commutation. Ainsi, le nombre de tâches actives entre deux instants de commutation est constant. Nous notons q l'intervalle de temps entre deux instants de commutation (le quantum d'ordonnancement). Le quantum q n'est pas fixé : il peut être défini comme un nombre entier de cycles processeurs, ainsi, sa durée, exprimée en secondes, dépend de la vitesse du processeur.

Nous définissons, dans cette section, la notion de k-voisinage nous permettant de savoir si il est possible de passer d'un état à un autre en utilisant une architecture à k processeurs. Ceci nous amène naturellement à des notions de k-continuité et de

k-distance entre deux états permettant de mesurer leur éloignement. Lorsque deux états sont éloignés et qu'il n'existe pas de chemin k-continu entre eux, la k-distance mesure l'importance de la k-discontinuité. C'est cette k-distance qui est à la base de notre mesure d'invalidité.

S'il faut au plus k processeurs pour passer d'un état de l'application à l'instant t à un état de l'application à l'instant $t+q$ nous disons que les deux points sont k-voisins (cf. figure 16). Une trajectoire passant par ces deux points sera dite k-continue entre les deux instants considérés.

Définition 8 *Deux points* $P=(t_1, p_1, ..., p_n)$ *et* $Q=(t_2, q_1, ..., q_n)$ *sont* k*-voisins* $\Leftrightarrow$ $t_1 = i \times q, t_2 = t_1 + q$ *et* $\sum_{j\in[1,n]} p_j - q_j \leq k \times q$.

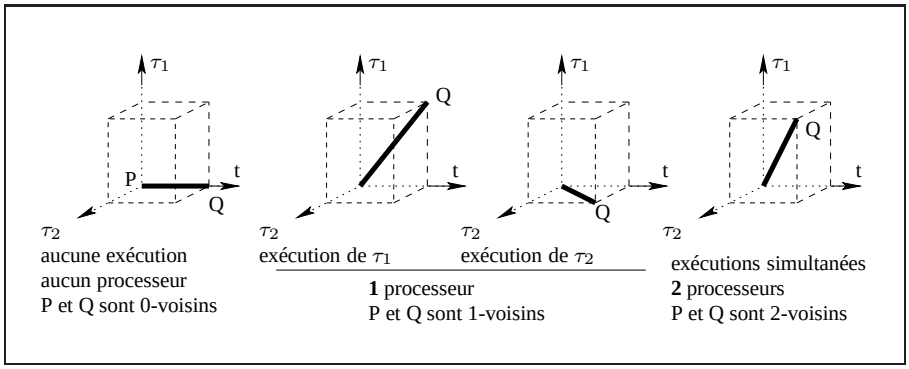

Figure 16. *Voisinages en 3D (deux tâches)*

Une trajectoire $\psi \in T(\Omega_R(\Gamma))$ est k-continue sur l'intervalle [i_1, i_2] si, quels que soient deux instants de commutation consécutifs de [i_1, i_2], elle est k-continue entre ces deux instants. Cette définition s'étend aux trajectoires dans leur ensemble : une trajectoire est dite k-continue si, quels que soient deux instants de commutation consécutifs, la trajectoire est k-continue entre ces deux instants. Une trajectoire est donc k-continue si deux points, associés à deux instants de commutation consécutifs quelconque, de cette trajectoire sont k-voisins.

Définition 9 *Un ensemble* $\Omega_R(\Gamma)$ *est dit* k*-continu s'il existe au moins une trajectoire* k*-continue* $\psi \in T(\Omega_R(\Gamma))$.

A partir de cette définition, nous étendons l'application T en définissant T_k l'application donnant l'ensemble $T_k(\Omega_R(\Gamma))$ des trajectoires k-continues de $\Omega_R(\Gamma)$.

4.5. *Décision d'ordonnançabilité*

Théorème 2 *$\Omega_R(\Gamma)$ est p-continu $\Leftrightarrow$ il existe un ordonnancement temporellement valide pour Γ sur p processeurs.*

Ce théorème découle directement des définitions relatives à la k-continuité. Un ordonnancement valide, pour un système s'exécutant sur une architecture comportant k processeurs et utilisant un ensemble de ressources R, est une trajectoire k-continue de $T(\Omega_R(\Gamma))$. La décision d'ordonnançabilité pour un système $(\tau_i)_{i\in[1,n]}$ utilisant une ressource R et fonctionnant sur k processeurs est calculée en évaluant le prédicat suivant : $T_k(\Omega_R(\Gamma)) \neq \emptyset$.

Notons que même si $T_k(\Omega_R(\Gamma))$ est vide, l'ensemble $\Omega_R(\Gamma)$, n'est jamais vide : en effet, il contient toujours au moins l'état (0,...,0), l'état (PPCM(T_i),C_1,...,C_n) (correspondant à l'état initial du cycle suivant), etc. Il contient, de plus, tous les états compatibles avec la gestion des ressources et les contraintes temporelles.

5. La quantification dans GRETA : définition d'une mesure d'invalidité

Dans le cas d'une configuration non ordonnançable, l'objet géométrique n'est pas vide, mais composé de trajectoires discontinues. Nous pouvons alors analyser l'objet résultant de la modélisation et raffiner ainsi notre réponse binaire de validité, c'est l'objet de la section suivante.

5.1. *Définitions*

Nous avons vu que même lorsqu'une application est invalide, son modèle géométrique n'est pas vide. L'ensemble $\Omega_R(\Gamma)$ contient donc des états valides qui font ou non partie d'une trajectoire valide de Γ. Dans le cas où il n'existe pas de trajectoire valide, les états de $\Omega_R(\Gamma)$ font partie de trajectoires discontinues. Pour mesurer l'invalidité de l'application, nous mesurons les tailles des intervalles de discontinuité des trajectoires. Nous nous plaçons donc dans l'espace $\mathbb{N}^{n+1}$ correspondant à l'ensemble des états de commutation.

5.1.1. *Composantes p-continues*

Considérons une configuration Γ de n tâches, son modéle géométrique $\Omega_R(\Gamma)$ et l'ensemble de ses trajectoires (valides et invalides) $T(\Omega_R(\Gamma))$. Nous utilisons, dans la suite, les notions suivantes apparentées à celles de connexité et de composante connexe.

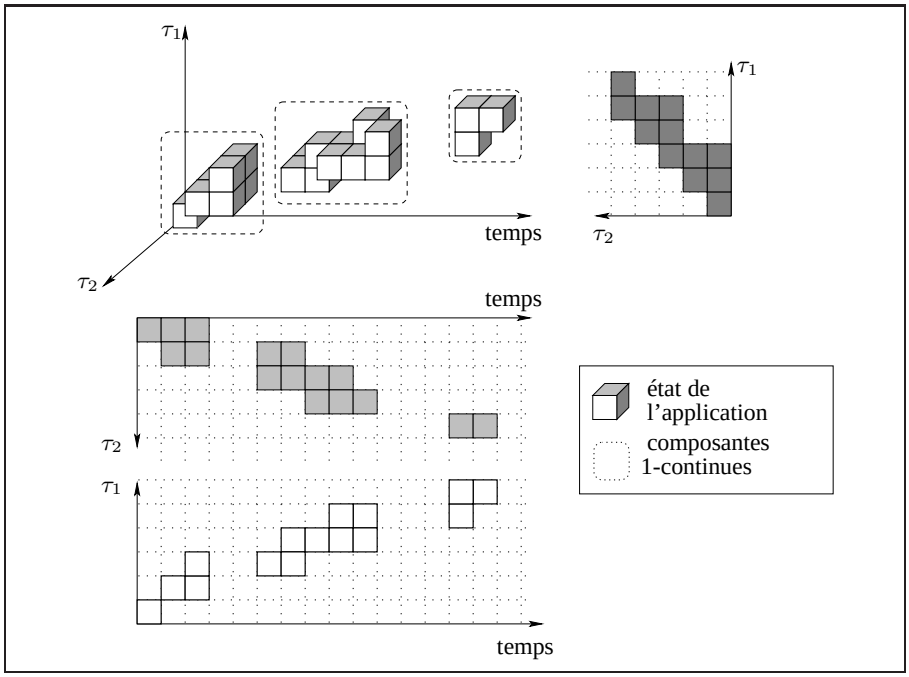

Figure 17. *Composantes 1-continues*

Définition 10 *Un sous-ensemble A d'états de* $\Omega_R(\Gamma)$ *est* p*-continu si :* $\forall (P_1, P_2) \in A^2,\ \forall (t_1, t_2) \in \mathbb{N}$

$$\left.\begin{array}{l} P_1 = (t_1, (x_i)_{i\in[1,n]}) \\ P_2 = (t_2, (x_i)_{i\in[1,n]}) \\ t_1 < t_2, \end{array}\right\} \Rightarrow \begin{array}{l} \exists \psi \in T(\Omega_R(\Gamma))\ p\text{-continue sur } [t_1, t_2] \mid \\ \psi(t_1) = P_1, \psi(t_2) = P_2. \end{array}$$

Définition 11 *Une composante* p*-continue de* $\Omega_R(\Gamma)$ *est un sous-ensemble* p*-continu maximal de* $\Omega_R(\Gamma)$.

Une composante p-continue A est donc telle qu'il n'existe pas d'état n'appartenant pas à A qui soit sur une trajectoire p-continue passant par un état de A (cf. figure 17). Nous avons alors les deux propriétés suivantes :

– une composante p-continue est fermée ;

– deux composantes p-continues sont disjointes.

Nous notons $\mathcal{X}^p$ l'ensemble des composantes p-continues de $\Omega_R(\Gamma)$. Dans le cadre particulier de notre modèle, nous avons la propriété suivante :

soit $k < l$, $\mathcal{X}^l = \{X_1, ..., X_r\},\ \mathcal{X}^k = \{Y_1, ..., Y_s\}$ alors

$$\forall j \in [1, s], \exists i \in [1, r] / Y_j \subset X_i.$$

Ceci découle directement de la définition de la p-continuité qui implique qu'une trajectoire k-continue sur un intervalle I est l-continue sur I pour tout $l < k$.

5.1.2. *Distance*

Nous associons la taille d'un intervalle de discontinuité d'une trajectoire à un *indicateur géométrique* entre deux composantes p-continues de $\Omega_R(\Gamma)$.

Soit P_1 et P_2 deux états quelconques de l'espace $(t, (\tau_i)_{i\in[1,n]})$. Nous définissons δ_p comme le nombre d'unités de temps d'exécution minimum nécessaire, avec p processeurs, pour passer de l'état de l'application correspondant au point P_1 à celui associé au point P_2. La fonction δ_p est définie de la façon suivante.

Définition 12 *Soit $P_1 = (t_1, (x_i)_{i\in[1,n]})$ et $P_2 = (t_2, (y_i)_{i\in[1,n]})$ tels que $t_1 < t_2$. La distance entre P_1 et P_2 est :*

$$\delta_p \ : \ \mathbb{N}^{n+1} \times \mathbb{N}^{n+1} \rightarrow \mathbb{N}$$
$$\delta_p\,(P_1, P_2) = max\Big(\Big\lceil \frac{\big(\sum_{i=1}^{n} |y_i - x_i|\big)}{p} \Big\rceil,\ max_{i\in[1,n]}|y_i - x_i|, |t_2 - t_1|\Big).$$

Le terme $\Big\lceil \frac{\big(\sum_{i=1}^{n} |y_i - x_i|)\big)}{p} \Big\rceil$ correspond à une répartition au mieux de la charge sur les processeurs disponibles. Toutefois, une tâche τ_i doit être exécutée pendant $|y_i - x_i|$ unités de temps pour passer de l'état P_1 à l'état P_2. Les tâches n'étant pas réentrantes, la durée $\delta_p\,(P_1, P_2)$ ne peut être inférieure à la plus grande durée d'exécution nécessaire à une tâche pour passer d'un état à l'autre. Ceci nous donne le terme $max_{i\in[1,n]}|y_i - x_i|$. Le dernier terme, $|t_2 - t_1|$, signifie que pour passer de l'état P_1 à l'état P_2 on ne peut pas mettre moins de temps que le temps qui sépare les deux états.

Théorème 3 *δ_p est une distance métrique, c'est-à-dire, δ_p vérifie les axiomes de séparation ($\delta_p(P_1,P_1)=0$), de symétrie ($\delta_p(P_1,P_2)=\delta_p(P_2,P_1)$) et triangulaire ($\delta_p(P_1,P_3)\leq \delta_p(P_1,P_2)+\delta_p(P_2,P_3)$).*

Ce théorème découle directement des propriétés de la valeur absolue ($|a - a| = 0, |a - b| = |b - a| et |a + b| \leq |a| + |b|$).

Nous définissons la distance δ_p entre deux composantes p-continues de la façon suivante.

Définition 13 *La distance δ_p entre deux composantes p-continues X_i et X_j de $\Omega_R(\Gamma)$ est :*

$$\delta_p(X_i, X_j) = min_{P_1\in X_1, P_2\in X_2}\big(\delta_p(P_1, P_2)\big).$$

REMARQUE. — La distance δ_p entre deux composantes p-continues n'est jamais nulle car les composantes p-continues sont disjointes : une composante p-continue est un ensemble p-continu maximal.

Ces notions de distance nous offrent la possibilité de quantifier l'invalidité. C'est l'objet de la section suivante.

5.2. *Mesure d'invalidité*

Nous définissons une mesure d'invalidité M_p d'une configuration de tâches Γ grâce à la distance entre composantes p-continues. Le modèle géométrique d'une application invalide comporte plusieurs composantes p-continues. Nous disposons, pour chaque couple de composantes, de la distance δ_p. La mesure d'invalidité que nous définissons tient compte de cet ensemble de distances. La mesure devant quantifier l'invalidité, nous avons choisi de considérer le plus grand intervalle de discontinuité. Notre hypothèse à été que lorsque, en modifiant la spécification, on aura réussi à combler le *vide* correspondant au plus grand intervalle, on aura par la même occasion comblé les *vides* plus petits et ainsi rendu l'application valide.

Définition 14 *Soit $\mathcal{X}^p = \{X_1, ..., X_s\}$ l'ensemble des composantes p-continues de $\Omega_R(\Gamma)$. Alors :*

$$M_p(\Gamma) = \begin{cases} 0 & \textit{si } |\mathcal{X}| = 1, \\ Max_{i,j \in [1,s]^2}(\delta_p(X_i, X_j)) & sinon. \end{cases}$$

Dans la suite, nous notons $\mathcal{C}$ un contexte d'ordonnancement à p processeurs, et Γ une configuration de n tâches.

5.2.1. *Adéquation avec la spécification (section 2)*

Notre mesure devant quantifier l'invalidité, elle doit vérifier les propriétés suivantes :

– être nulle pour toutes les configurations ordonnançables ($M_p(\Gamma) = 0 \Leftrightarrow \mathcal{O}_{\mathcal{C}}(\Gamma) \neq \emptyset$) et non nulle pour les applications invalides ($M_p(\Gamma) > 0 \Leftrightarrow \mathcal{O}_{\mathcal{C}}(\Gamma) = \emptyset$) ;

– $M_p(\Gamma) = 0 \Rightarrow \forall k > p,\ M_k(\Gamma) = 0$, car une application ordonnançable dans un contexte comportant p processeurs l'est aussi lorsque le nombre de processeurs est plus grand ;

– $M_p(\Gamma) > 0 \Rightarrow \forall k < p,\ M_k(\Gamma) \geq M_p(\Gamma)$, car une application non ordonnançable sur p processeurs ne l'est pas pour un nombre de processeurs plus petit.

Montrons que $M_p(\Gamma) = 0 \Leftrightarrow \mathcal{O}_{\mathcal{C}}(\Gamma) \neq \emptyset$: si $M_p(\Gamma) = 0$ alors $|\mathcal{X}^p|$=1 et donc $\mathcal{X}^p = \{\Omega_R(\Gamma)\}$. Comme les éléments de $\mathcal{X}^p$ sont p-continus, $\Omega_R(\Gamma)$ est p-continu. Il existe donc une trajectoire p-continue correspondant à un ordonnancement valide de

Γ sur p processeurs et donc $\mathcal{O}_{\mathcal{C}}(\Gamma) \neq \emptyset$. Inversement, si $\mathcal{O}_{\mathcal{C}}(\Gamma) \neq \emptyset$ alors il existe une trajectoire ψ p-continue valide pour Γ. Comme ψ est p-continue, tous les points qu'elle définit font partie de la même composante p-continue. Or ψ est définie pour tout t de $\mathbb{R}$ donc il n'y a qu'une seule composante p-continue. Dans ce cas, $|\mathcal{X}^p|$=1 et $M_p(\Gamma)$=0.

De la même manière, on montre que $M_p(\Gamma) > 0 \Leftrightarrow \mathcal{O}_{\mathcal{C}}(\Gamma) = \emptyset$: Si $M_p(\Gamma) > 0$ alors $|\mathcal{X}^p| >$1. Comme $\mathcal{X} = \{X_1, ..., X_s\}$ avec $\forall i \in [1, |\mathcal{X}^p|], \forall j \neq i, X_i \cap X_j = \emptyset$, il n'existe aucune trajectoire p-continue sur [0,+∞[et donc aucun ordonnancement valide ($\mathcal{O}_{\mathcal{C}}(\Gamma) = \emptyset$). Inversement, si $\mathcal{O}_{\mathcal{C}}(\Gamma) = \emptyset$, alors il n'existe aucune trajectoire p-continue sur [0,+∞[. Soit ψ une trajectoire de Γ, ψ n'étant pas p-continue, il existe $I = [t_1, t_2]$ / ψ n'est pas p-continue sur I. On a donc P_1= ($t_1 - 1$, $\psi(t_1 - 1)$) et P_2= ($t_2 + 1$, $\psi(t_2 + 1)$) qui ne sont pas p-continus. Suivant la définition des composantes p-continues, on a nécessairement au moins deux composantes p-continues X_1 et X_2 avec $P_1 \in X_1$ et $P_2 \in X_2$. Et enfin $|\mathcal{X}^p| \geq 2$ et donc $M_p(\Gamma) > 0$.

La mesure que nous avons définie correspond donc bien à la spécification : la mesure d'invalidité associée à une application ordonnançable est nulle.

Montrons maintenant que $M_p(\Gamma) = 0 \Rightarrow \forall k > p,\ M_k(\Gamma) = 0$: supposons $M_p(\Gamma) = 0$; alors $|\mathcal{X}^p|$=1. Il existe donc une trajectoire $\psi \in T_p(\Omega_R(\Gamma))$ p-continue sur l'intervalle $[0, +\infty[$. Comme toute trajectoire p-continue est k-continue pour tout $k > p$, ψ est aussi k-continue sur $[0, +\infty[$. Il existe donc une seule composante k-continue : $|\mathcal{X}^k|$=1. Nous avons donc $\forall k > p,\ M_k(\Gamma) = 0$.

Une application qui n'est pas ordonnançable sur p processeurs l'est « encore moins » lorsque l'on diminue le nombre de processeurs.

Montrons que $M_p(\Gamma) > 0 \Rightarrow \forall k < p,\ M_k(\Gamma) \geq M_p(\Gamma)$: supposons $M_p(\Gamma) > 0$ alors $\mathcal{X}^p = \{X_1^p, ..., X_r^p\},\ r > 1$. Soit $k < p$, d'après la propriété des composantes continues, on a : $\mathcal{X}^k = \{X_1^k, ..., X_l^k\},\ l > 1$ et $\forall i \in [1, l], \exists j \in [1, r],\ X_i^k \subseteq X_j^p$. La distance maximale entre deux composantes k-continues est donc au minimum égale à la distance maximale entre deux composantes p-continues (cf. figure 18). Nous avons donc $M_k(\Gamma) \geq M_p(\Gamma)$.

5.2.2. *Propriétés supplémentaires de la mesure*

On note Θ une trajectoire p-continue. Par définition des distances et de la mesure, on a : $M_p(\Gamma) = 0 \Rightarrow \forall t \in \mathbb{R},\ P_1 = (t, \Theta(t)),\ P_2 = (t + 1, \Theta(t + 1)),\ \delta_p(P_1, P_2) = 1$. Soit p = inf $\{k,\ M_k(\Gamma) = 0\}$, le plus petit nombre de processeurs pour lequel l'application est ordonnançable. Alors on a la propriété suivante :

$$\forall k < p,\ M_k(\Gamma) = \left\lceil \frac{p}{k} \right\rceil .$$

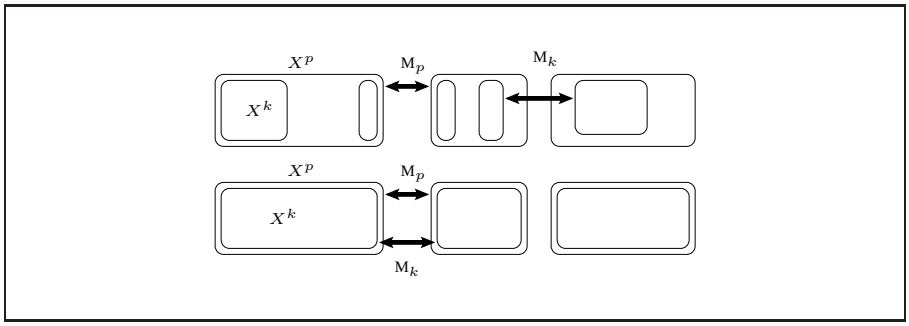

Figure 18. *Mesures avec k et p processeurs*

Preuve : soit p=inf $\{k,\ M_k(\Gamma) = 0\}$, et soit $k < p$, alors on considère tous les couples de points de la trajectoire Θ consécutifs P_1 et P_2 tels que : $P_1 = (t, \Theta(t))$ et $P_2 = (t+1, \Theta(t+1))$ soient k-discontinus. Comme P_1 et P_2 sont p-continus, on a :

$$P_1 = (t, (x_i)_{i\in[1,n]}) \text{ et } P_2 = (t+1, (y_i)_{i\in[1,n]}) \Rightarrow \delta_p(P_1, P_2) = 1.$$

Donc :

$$\left\lceil \frac{\sum_{i=1}^{n} |y_i - x_i|}{p} \right\rceil = 1.$$

Comme p=inf $\{k,\ M_k(\Gamma) = 0\}$, on a $\sum_{i=1}^{n} |y_i - x_i| = p$: en effet, si $\sum_{i=1}^{n} |y_i - x_i| = p - 1$ alors $M_{p-1}(\Gamma)$=0 et donc $p \neq inf\{k, M_k(\Gamma) = 0\}$. On a alors, $\delta_k(P_1, P_2) = max(\lceil \frac{p}{k} \rceil, 1, 1) = \lceil \frac{p}{k} \rceil$. Et donc : $\forall P_1 = (t, \Theta(t)), P_2 = (t+1, \Theta(t+1))$, $\delta_k(P_1, P_2) = \lceil \frac{p}{k} \rceil (> 1)$ ou $\delta_k(P_1, P_2) = 1$, suivant que les points sont k-continus ou non. Ainsi, $M_k(\Gamma) = Max_{t\in\mathbb{R}}(\delta_k((t, \Theta(t)), (t+1, \Theta(t+1))))$. Et finalement $M_k(\Gamma) = \lceil \frac{p}{k} \rceil$.

Cette propriété permet de simplifier l'algorithme de calcul de la mesure.

5.2.3. *Implantation*

Sur la base de ce modèle géométrique, nous avons développé le logiciel GemSmarts. Ce logiciel prend en paramètre les spécifications temporelles des tâches ainsi que les contraintes de processeurs et de ressources. Il construit l'objet discret de $(n+1)$ dimension associé à l'application intégrant le partage de ressource et calcule, pour chaque nombre de processeur possible, la mesure d'invalidité de l'application.

Sur la capture d'écran de GemSmarts présentée figure 19, nous voyons, dans la partie basse, les spécifications temporelles des six tâches de l'application avec, au-dessus, une séquence d'ordonnancement valide. Dans la partie haute est affiché la projection dans un espace à 3 dimensions (deux tâches et le temps) du modèle géométrique de l'application. A droite de l'image figurent les mesures pour chaque nombre de processeur possible (de 1 à 6). A gauche se trouvent les outils permettant de faire pivoter, de translater ou d'agrandir l'image centrale.

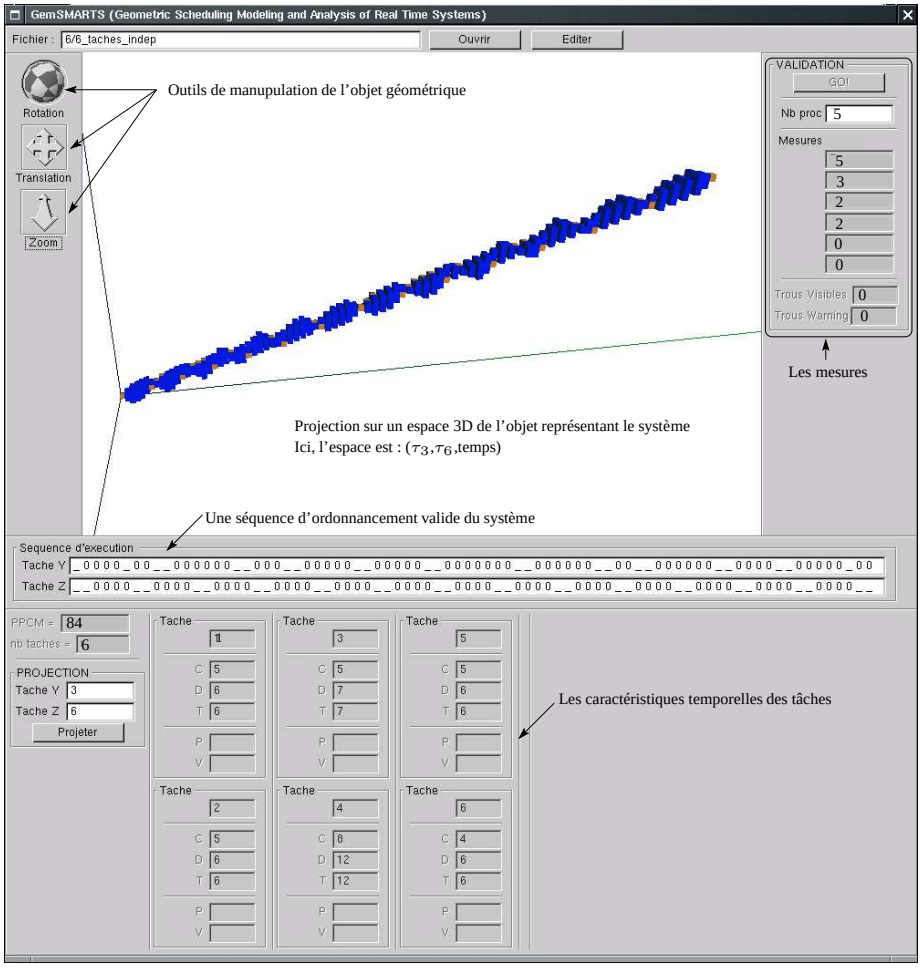

Figure 19. *GemSmarts : une capture d'écran de notre logiciel pour une configuration de 6 tâches ordonnançables sur une architecture disposant de 5 processeurs*

Les temps de calculs n'excédant que très rarement la seconde, ce logiciel fait preuve d'une grande interactivité et permet une analyse rapide de l'ordonnançabilité des applications modélisées.

5.3. *Etude du comportement de la mesure*

Dans cette section, nous essayons de dégager des relations entre la modification de certains paramètres des tâches et la diminution ou l'augmentation de notre mesure d'invalidité sur un exemple simple.

L'application que nous étudions est composée de six tâches dont quatre partagent une ressource. Les taux d'utilisation et de charge de ce système sont de 71 % (cf. figure 20).

tâche	C	D	T	P	V
τ_1	5	6	6	1	4
τ_2	5	6	6	2	5
τ_3	5	6	6	1	4
τ_4	8	12	12	5	8
τ_5	5	6	6	-	-
τ_6	4	6	6	-	-

M_1	M_2	M_3	M_4	M_5	M_6
sans ressource					
5	3	2	2	**0**	**0**
avec ressources					
14	7	5	5	4	4

Figure 20. *Un exemple d'application ; les mesures sont données avec et sans prise en compte du partage de ressources*

Dans le premier tableau de la figure 20, nous montrons les caractéristiques temporelles des tâches du système ainsi que les différentes mesures lorsque l'on considère des tâches indépendantes ou que l'on tient compte des partages de ressources. On note que le système de tâches indépendantes est ordonnançable sur une architecture disposant de cinq processeurs. Les partages de ressources impliquent une non-ordonnançabilité du système quel que soit l'architecture considérée ($M_6 = 0$). Dans la suite nous étudions les évolutions des mesures en fonction des variations de périodes et de délais critiques que nous appliquons aux tâches.

La figure 21a présente l'évolution des différentes mesures M_1, M_2, M_3, M_4, M_5 et M_6 en fonction de l'augmentation de la période de la tâche τ_1. Cette série de tests a pour but de déterminer une relation entre la diminution du taux d'utilisation et la mesure en présence de partages de ressources. Nous avons choisi de ne tester que les périodes multiples de la période initiale puisque, dans le cas réel, la périodicité des capteurs ne peut être modifiée et on ne peut modifier la tâche de lecture associée qu'en restant synchronisé avec le capteur. Comme nous pouvons le remarquer, les mesures ont un comportement oscillant : on ne peut donc pas relier directement l'augmentation de la période avec la diminution de la mesure. Cette oscillation est principalement dûe aux partages de ressources qui peuvent se passer plus ou moins bien suivant la synchronisation des tâches impliquées. Lorsque le système de tâches ne partage pas de ressource, la modification de la période entraine soit une diminution de la mesure soit une stagnation de celle-ci si le taux de charge du système est trop important sur la première période.

Le second graphique 21b, présente l'évolution de la mesure en fonction de l'augmentation du délai critique de la tâche τ_1. Il permet de se rendre compte que les mesures ont un comportement cohérent face à la diminution de la charge : elles sont décroissantes. Sur cet exemple elles ne sont pas strictement décroissantes en raison des partages de ressources qui interfèrent avec la diminution de la charge.

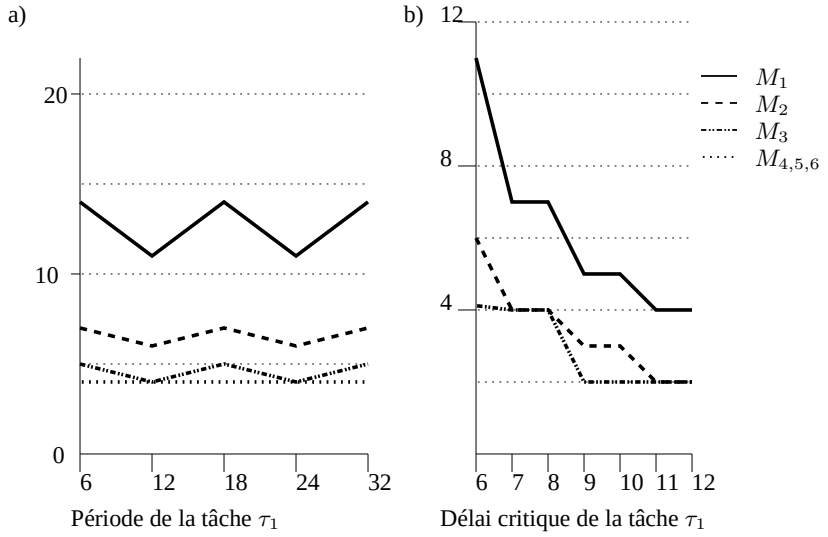

Figure 21. *Evolution de la mesure lorsque l'on fait varier la prériode ou le délai critique d'une tâche*

Le tableau 1 présente une partie des résultats obtenus lors de notre expérimentation préliminaire. Il montre qu'une forte augmentation des périodes et délais critiques des tâches permet de rendre l'application valide pour trois processeurs.

Cette étude préliminaire montre que la mesure que nous avons définie n'est pas monotone et qu'elle n'est pas directement corrélée avec la diminution du taux d'utilisation du système. Lorsque le système est indépendant, cela signifie que la charge est trop importante, il faut donc modifier les délais critiques. L'étude des variations de la mesure dans le cas indépendant permet donc de connaitre l'influence des périodes et des délais critiques sur l'invalidité de l'application. Lorsque le système partage des ressources critiques, cela signifie que, lorsque l'on modifie les périodes, on modifie également le positionnement relatif des sections critiques ce qui entraine soit une plus grande invalidité en cas de synchronisation des sections critiques, soit une moins grande invalidité lorsqu'il y a un possible décalage de celles ci. On peut donc définir les périodes des tâches les plus adaptées pour un partage de ressources. Cette mesure est également cohérente avec les résultats classiques : elle est inversement reliée à la charge du système et permet de detecter certaines causes de non-ordonnançabilité.

τ_1		τ_2		τ_3		τ_4		$Mesure$					
D	T	D	T	D	T	D	T	M_1	M_2	M_3	M_4	M_5	M_6
6	6	6	6	6	6	12	12	14	7	5	4	4	4
6	**12**	6	6	6	6	12	12	**11**	**6**	**4**	4	4	4
6	**18**	6	6	6	6	12	12	14	7	5	4	4	4
6	**24**	6	6	6	6	12	12	**11**	**6**	**4**	4	4	4
6	6	6	**12**	6	6	12	12	**11**	**6**	**4**	4	4	4
6	6	6	6	6	**12**	12	12	**11**	**6**	**4**	4	4	4
6	6	6	6	6	6	12	**24**	14	7	5	4	4	4
6	**12**	6	**12**	6	**12**	12	12	**11**	**6**	**4**	4	4	4
7	**12**	6	6	6	6	12	12	**7**	**4**	**4**	4	4	4
8	**12**	6	6	6	6	12	12	**7**	**4**	**4**	4	4	4
9	**12**	6	6	6	6	12	12	**5**	**3**	**2**	**2**	**2**	**2**
10	**12**	6	6	6	6	12	12	**5**	**3**	**2**	**2**	**2**	**2**
11	**12**	6	6	6	6	12	12	**4**	**2**	**2**	**2**	**2**	**2**
12	**12**	6	6	6	6	12	12	**4**	**2**	**2**	**2**	**2**	**2**
8	**12**	**8**	**12**	**8**	**12**	12	12	**6**	**3**	**2**	**2**	**2**	**1**
12	**12**	**12**	**12**	**12**	**12**	**18**	**24**	**3**	**2**	**0**	**0**	**0**	**0**

Tableau 1. *Evolution des mesures lors de modifications des périodes et délais critiques de tâches dépendantes*

En plus d'un comportement cohérent face aux modifications de caractéristiques temporelles, l'aide visuelle que permet un modèle géométrique apportée par le logiciel permet de localiser l'invalidité : il est possible, en observant les étranglements des objets et les zones de discontinuité, de connaître les tâches sur lesquelles les contraintes portent et d'en déduire les ressources bloquantes ou les délais insuffisants.

Du fait de la vitesse de calcul du modèle, il est possible d'effectuer plusieurs modélisations successives rapidement. Une étude d'un système combinant observation des mesures et analyse de l'objet géométrique peut être menée de la façon suivante :

1) modélisation du système sans prendre en compte les ressources ;

2) étude des variations des mesures et en déduire l'influence des périodes et des délais ;

3) observation de l'objet géométrique pour localiser les étranglements et discontinuités et en déduire précisement les tâches trop contraintes ;

4) modification des caractéristiques temporelles ;

5) modélisation du système en intégrant les ressources critiques ;

6) si besoin observer les variations de la mesure pour en déduire les périodes les plus adaptées ;

7) localisation des discontinuités pour connaître les ressources problèmatiques ;

8) déplacement des sections critiques et/ou modification des caractéristiques temporelles.

Les premiers résultats obtenus sont encourageants et permettent déjà d'effectuer une grande partie du diagnostic sur des systèmes de taille réel. Dans la suite de nos travaux, nous comptons effectuer une étude systèmatique de notre mesure pour déterminer plus finement la corrélation entre les valeurs des mesures et les caractéristiques temporelles des tâches afin de guider le concepteur en lui précisant quelles contraintes sont à modifier et dans quelles proportions. Suivant les résultats de cette analyse, nous espérons arriver à automatiser les phases 4, 6 et 8.

6. Conclusion

Notre étude a été effectuée en trois temps : dans un premier temps, l'étude d'un modèle sans temps, exhaustif : le modèle MARTA. Les principaux avantages de ce modèle sont l'existence d'outils d'analyses simples et efficaces ainsi que sa grande puissance d'expression. Ce modèle peut en effet représenter des applications centralisées multiprocesseurs identiques comportant des tâches dépendantes, à durées fixes, à départs différés et non nécessairement à échéance sur requête. MARTA a été enrichi pour tenir compte des tâches à durées variables et des contraintes matérielles (Geniet *et al.*, 2001) : il peut représenter des applications fonctionnant sur une architecture centralisée ou distribuée, multiprocesseur (uniforme ou non). Le fait que de tels systèmes puissent être modélisés par des langages *centre* permet de prouver la cyclicité de leur fonctionnement sans pour autant donner une borne à cette cyclicité dans le cas général. Ce modèle très expressif permet d'analyser l'ensemble des séquences d'ordonnancement valides pour un grand nombre de contextes d'exécution. Toutefois, le but de notre étude étant aussi l'analyse de l'invalidité, MARTA se révéla insuffisant. En effet, le modèle MARTA d'une application invalide est *vide*.

Pour palier ce problème, nous avons donc observé la structure des automates obtenus par MARTA à partir de laquelle nous avons conçu le modèle GRETA. La topologie des automates obtenus par MARTA pour une tâche est une surface munie de deux arêtes (la première pour l'inactivité avant l'activation, la seconde pour l'inactivité entre l'échéance et la réactivation.

Enfin, après avoir défini les notions de p-continuité et de composante p-continue, nous avons défini une mesure d'invalidité sur la base d'une distance entre composantes p-continues. Cette mesure dépend du nombre de processeurs et correspond à la taille du plus grand intervalle de p-discontinuité.

Nous avons conçu deux logiciels de validation temporelle. Le premier (RatAn), basé sur le modèle MARTA, permet de construire l'ensemble des séquences d'ordonnancement valides d'une application. Le second (GemSMARTS), basé sur le modèle GRETA, construit un objet géométrique et intègre le calcul de notre mesure. Plusieurs perspectives sont envisageables. Dans un premier temps, il serait intéressant d'étudier plus systématiquement le comportement de notre mesure en fonction des modifications des paramètres temporels des tâches. Dans la série de tests effectués, nous avons constaté que, conformément à nos prévisions, les meilleures valeurs sont obtenues

lorsque les tâches sont à échéance sur requête (le délai critique est le plus grand possible et donc la tâche a moins de chance de manquer son échéance). La modification du taux d'utilisation intervient peu dans l'évolution de la mesure, alors que la diminution du taux de charge propose de bien meilleurs résultats.

Dans un second temps, la mesure pourrait être raffinée par la prise en compte, par exemple, du nombre d'intervalles de discontinuité.

Le but de notre projet est, à long terme, la définition et l'implantation d'un ensemble d'outils d'aide à la spécification et à la rétroconception permettant, de manière efficace, de concevoir des applications temps-réel valides. Une partie de ces outils a été réalisée (RatAn, GemSMARTS), il est maintenant nécessaire de les améliorer et de les compléter. Il est par exemple envisageable d'y intégrer des outils d'analyses permettant de valider les algorithmes « en-ligne » classiques (RM, ED...) ainsi que de déterminer les ordonnancements permettant d'optimiser certains critères tels que la gigue, les temps de réponses, l'énergie consommée etc.

7. Bibliographie

Alur R., Dill D., « A Theory of Timed Automata », *Theoretical Computer Science*, vol. 126, 1994, p. 183-235.

Arnold A., Nivat M., « Comportements de processus », *AFCET Les Mathématiques de l'informatique*, 1982, p. 35-68.

Autebert J., *Théorie des langages et des automates*, Masson, 1994.

Baruah S. K., Gehrke J. E., Plaxton C. G., « Fast Scheduling of Periodic Tasks on Multiple Resources », *Actes de International Parallel Processing Symposium*, 1995, p. 280-288.

Buttazzo G., *Hard Real-Time Computing Systems*, Kluwer Academic Publishers, 1997.

Carpenter J., Funk S., Holman P., Srinivasan A., Anderson J., Baruah S., « A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms », 2004, p. 30-1 - 30-19.

Choquet-geniet A., « Un premier pas vers l'étude de la cyclicité en environnement multiprocesseur », *Actes de Real Time System 2005*, Teknea, 2005, p. 289-302.

Choquet-Geniet A., Geniet D., Cottet F., « Exhaustive Computation of the Scheduled Task Execution Sequences of a Real-Time Application », *Actes de Symposium on Formal Techniques in Real Time and Fault Tolerent Systems'96*, vol. 1135 of *Lecture Notes in Computer Science*, Springer-Verlag, October, 1996, p. 246-262.

Chrétienne P., Les réseaux de Petri temporisés, PhD thesis, Université de Paris VI, 1983.

CNRS, « Le temps réel », *Groupe de Réflexion Temp Réel, Technique et Science Informatiques*, vol. 7, n° 5, 1988, p. 493-500.

Geniet D., « Validation d'Applications Temps-Réel à Contraintes Strictes à l'Aide de Langages Rationnels », *Actes de Real Time Systems 2000*, Teknea, 2000, p. 91-106.

Geniet D., Largeteau G., « Validation Temporelle de Systèmes de Tâches Temps-Réel Strictes à Durées Variables à l'Aide de Langages », *Actes de Modélisation des Systèmes Réactifs 2001*, Hermes-Science publication, Octobre, 2001, p. 243-258.

Juanole G., « Modélisation et évaluation du protocole MAC du réseaux CAN », *Actes de ETR'99*, 1999, p. 187-200.

Kwak H.-H., Lee I., Sokolsky O., « Parametric approach to the specification and analysis of real-time scheduling based on ACSR-VP. », *Sci. Comput. Program.*, vol. 42, n° 1, 2002, p. 49-60.

Largeteau G., Chauvière B., Geniet D., « Une Approche Géométrique de la Validation Temps-Réel », *Actes de Real Time Systems 2004*, Teknea, 2004, p. 15-34.

Largeteau G., Geniet D., « Validation d'applications temps réel distribuées à contraintes strictes », *Actes de Real Time Systems 2002*, Teknea, 2002.

Lay D., *Algèbre linéaire*, de Boeck Université, 2005.

Lee T., Cheng A. M. K., « Multiprocessor Scheduling of Hard-Real-Time Periodic Tasks with Task Migration Constraints », *International workshop on RTCS and Application*, 1994.

Liu C., Layland J., « Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment », *Journal of the ACM*, vol. 20, n° 1, 1973, p. 46-61.

Mok A., Fundamental Design Problems for the Hard Real-Time Environments, PhD thesis, M.I.T, 1983.

Nissanke N., Leulseged A., Chillara S., « Probabilistic performance analysis in multiprocessos scheduling », *Computing and Control Engineering Journal*, vol. 13, n° 4, 2002, p. 171-179.

Ramchandani C., Analysis of Asynchronous Concurrent Systems by Petri Nets, Technical report, M.I.T, project MAC, 1974.

Srinivasan A., Baruah S. K., « Deadline-based scheduling of periodic task systems on multiprocessors. », *Inf. Process. Lett.*, vol. 84, n° 2, 2002, p. 93-98.

Stankovic J. A., « Real-time and embedded systems », *ACM Computing Surveys*, vol. 28, n° 1, 1996, p. 205-208.

Thimonier L., Generating Functions and Random Words, Thèse d'état, Univ. Paris 11, 1988.

Article reçu le 1^{er} avril 2005
Accepté après révision le 18 janvier 2007

Gaëlle Largeteau *est maître de conférences à l'université de Poitiers. Après une thèse dans le domaine du temps-réel soutenue à Poitiers, elle s'est intéressée à la modélisation géométrique et plus particulièrement à la géométrie discrète. Elle effectue ses recherches au sein du Laboratoire Signal, Image, Communications de l'Université de Poitiers.*

Dominique Geniet *est maître de conférences HDR à l'université de Poitiers. Après une thèse dans le domaine du parallélisme soutenue à Orsay, il s'est intéressé à la validation temporelle de programmes temps-réel et à l'int égration de ce processus de validation dans le cycle de vie du logiciel. Il effectue ses recherches au sein du Laboratoire d'Informatique Scientifique et Industrielle de l'Université de Poitiers.*