Laboratoire d'Informatique Scientifique et Industrielle
EA 1232
ENSMA et Université de Poitiers

Modélisation précise des applications temps réel en vue de leur validation temporelle

Synthèse des travaux
en vue de l'obtention d'une

Habilitation à Diriger des Recherches

Emmanuel Grolleau
Maître de Conférences à l'ENSMA

3 Décembre 2009

Jury

Rapporteur	Pr. Jean-Charles Billaut	LI Tours, EPU Polytech'Tours
Rapporteur	Pr. Serge Haddad	LSV, ENS Cachan
Rapporteur	Pr. Guy Juanole	LAAS Toulouse
Directeur	Pr. Francis Cottet	LISI, ENSMA
Examinateur	Pr. Isabelle Puaut	IRISA, Université de Rennes
Examinateur	Pr. Pascal Richard	LISI, Université de Poitiers
Examinateur	Pr. Ye Qiong Song	INRIA, INPL Nancy
Examinateur	Pr. Yvon Trinquet	IRCCyN, Ecole Centrale de Nantes

à Alexandria

Remerciements

Je tiens à remercier les membres du jury pour l'intérêt qu'ils ont porté à mes travaux de recherche, et les discussions qui ont contribué à l'amélioration de ce manuscript. Je tiens aussi à remercier Francis Cottet pour avoir accepté de diriger mon Habilitation à Diriger des Recherches.

Je souhaite remercier mes doctorants pour avoir été très actifs et réactifs : j'ai eu beaucoup de chance d'être en charge de la direction de leurs travaux. Je tiens aussi à remercier de façon particulière chacun d'entre eux. Merci à Ngo Khanh Hieu pour sa bonne humeur, et sa persévérance malgré les conditions particulières de sa thèse. Merci à Karim Traoré pour ses efforts constants, les talents qu'il a sus développer et son calme dans l'urgence des échéances de concours. Merci à Ahmed Rahni pour sa rigueur et son sourire constant face à mes exigences croissantes devant ses qualités scientifiques croissantes elles aussi. Merci à François Dorin pour son engagement pédagogique et scientifique. Ce mémoire tire la majeure partie de son contenu dans leurs travaux à mes côtés.

Je tiens à remercier Francis Cottet et Annie Geniet, pour le souffle qu'ils m'ont insufflé pendant ma thèse de doctorat, j'espère appliquer la rigueur scientifique qu'ils m'ont transmise, et aussi suivre mes doctorants de façon aussi réactive que la façon dont ils m'ont suivi. En tant que doctorant, j'ai été rassuré maintes fois de voir un article, ou des chapitres de ma thèse, revenir couverts d'annotations en rouge. J'essaie à mon tour de rassurer mes doctorants en montrant la même rigueur en tant qu'encadrant.

Je souhaite remercier tous mes collègues et amis du LISI, en particulier Pascal Richard, qui m'a beaucoup appris, et Michaël Richard qui m'a beaucoup aidé aussi bien sur le plan scientifique que pédagogique. Michaël a notamment accepté d'absorber une surcharge d'enseignement importante afin de me laisser du temps pour la préparation de ce mémoire. Merci aussi à Yamine Âït-Ameur qui m'a beaucoup aidé aussi bien sur le plan administratif que scientifique. Merci aussi à Guy Pierra qui durant de nombreuses années, en tant que directeur, m'a donné sa confiance. Enfin, je tiens à remercier tous mes collègues du laboratoire, du département d'enseignement, et les doctorants, pour la convivialité de nos relations de travail.

Je tiens à remercier aussi mes collègues de l'ENSMA, pour leur confiance et leur sympathie.

Je souhaite remercier mes étudiants notamment de l'ENSMA, mais aussi de Master, des instituts polytechniques de Hanoï et d'Ho Chi Minh Ville, de SupMéca et de l'ESIP. Non seulement l'enseignement procure une joie de partager avec eux, mais les contacts que beaucoup ont gardé avec moi après l'obtention de leur diplôme par le biais d'offres de stage, de visites, des journées d'intégration des nouveaux élèves, me touche particulièrement. Je leur souhaite toute la réussite possible.

Merci à mon épouse Alexandria pour son amour constant qui est le pilier de ma bonne humeur quotidienne, et à mon fils Enzo pour sa curiosité qui fait que je me sens papa même lorsque je prépare des enseignements à la maison. Enfin je remercie mes parents, mon frère et ma sœur pour leur soutien.

Table des matières

Chapitre 1

Introduction générale

Ce document présente la synthèse de mes travaux de recherche depuis ma nomination en 2000 aux fonctions de maître de conférences à l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique (ENSMA), au département Informatique & Automatique pour la partie enseignement, et au Laboratoire d'Informatique Scientifique et Industrielle (LISI), laboratoire commun à l'ENSMA et à l'Université de Poitiers. Je suis membre de l'équipe modélisation et Analyse des Systèmes Temps Réel. Notre équipe de recherche s'intéresse principalement à la validation temporelle de systèmes temps réel.

Ma thèse de doctorat a eu lieu dans la même équipe, sous la direction d'Annie Geniet et Francis Cottet, dans la thématique ordonnancement hors-ligne, et traitait de modélisation de systèmes temps réel à l'aide de Réseau de Petri (RdP) en vue de leur validation temporelle. Cette validation temporelle repose sur une étude exhaustive de l'espace des ordonnancements valides. Elle a donné lieu à un résultat général portant sur la cyclicité des ordonnancements dans le cas monoprocesseur. Je collabore actuellement avec Joël Goossens (Université Libre de Bruxelles (ULB)) et Liliana Cucu (Institut National de Recherche en Informatique et en Automatique (INRIA), Nancy) sur la généralisation de ces résultats au cas multiprocesseur.

Je me suis ensuite intéressé à une technique méta-heuristique (colonies de fourmis) pour la recherche de séquence d'ordonnancement valide, en collaboration avec Habiba Drias (LRIA Alger).

Parallèlement à mes travaux sur l'ordonnancement hors-ligne, je me suis d'une part intéressé à l'ordonnancement en-ligne (prise en compte de facteurs pratiques : précédences, décalage de dates de réveil, etc.) en collaboration avec Pascal Richard, Michaël Richard et Francis Cottet, et ai d'autre part abordé un aspect recherche et enseignement correspondant à la proposition d'une méthode de spécification, conception, développement conformément au profil Ravenscar, orientée flots de données, à destination d'ingénieurs généralistes. Cela a donné lieu à l'encadrement du stage de master, puis de la thèse à distance, de Ngo Khanh Hieu (qui était doctorant et enseignant à l'Institut Polytechnique de Ho Chi Minh Ville). Ce travail entre dans un travail plus large que je souhaite développer dans l'équipe : la validation temporelle dans le cycle de développement orienté modèle des applications critiques. Il s'agit d'une part de proposer des méthodes facilitant un pré-validation temporelle mais aussi des méthodes permettant une aide au choix des paramètres temporels, et plus généralement au dimensionnement matériel du système.

En 2002 j'ai pris la co-direction d'un projet ENSMA mêlant recherche, enseignement,

et technologie avec Alain Farcy, du Laboratoire d'Etudes Aérodynamiques (LEA). Ce projet, toujours en cours, consiste en la mise en œuvre d'un drone convertible (vol horizontal ou vertical). J'ai eu la charge de la conception et réalisations logicielles et matérielles du système embarqué et de la station sol. La partie conception et implémentation a occupé la moitié de la thèse de doctorat de Karim Traoré dont j'ai eu le direction. Lorsque nous nous sommes intéressés à la validation de l'application, nous avons noté des lacunes importantes dans les modèles de validations en-ligne classiques ; nous nous sommes alors intéressés à un modèle de tâches temps réel prenant en compte des facteurs liés au décalage de certaines tâches entre elles, de façon indépendante des autres : les transactions. Nous avons obtenu divers résultats intéressants sur les transactions, et pu unifier la théorie portant sur ce modèle au modèle des tâches multiframes généralisées, étudié dans la littérature. Etant donnée la richesse des études à mener sur le modèle des transactions, la thèse d'Ahmed Rahni, dont j'ai eu la direction, a poursuivi les travaux entamés à la fin de la thèse de doctorat de Karim, sur l'étude des transactions. Nous continuons d'ailleurs à travailler sur ce modèle, qui semble être l'un des modèles de tâches temps réel les plus généraux.

Enfin, plus récemment, toujours dans le cadre ordonnancement en-ligne, je me suis intéressé à la validation dans le cas réparti. Dans ce contexte, je co-encadre la thèse de François Dorin, qui fait suite à la thèse de Michaël Richard. Nous développons une technique de séparation et évaluation permettant de faire ordonnancement et placement conjoint de tâches, de façon à ce que le système soit validable par analyse holistique. L'analyse holistique permet de valider globalement une application répartie, en prenant en compte les délais d'arrivée de messages dus à l'ordonnancement sur le réseau et sur les calculateurs. Cette technique permettra de dimensionner des systèmes répartis tout en assurant un placement.

Ce mémoire ne suit pas la chronologie de mes recherches, mais un découpage par concepts : la première partie introduit la problématique de l'ordonnancement, elle est suivie de la partie contributions, découpée en trois parties : ordonnancement hors-ligne, ordonnancement en-ligne et positionnement de la validation temporelle dans le cycle de développement. La troisième partie présente des perspectives de recherche.

Première partie

Introduction

1.1 Introduction au temps réel

1.1.1 Support d'exécution

On appelle système temps réel un ensemble de programmes informatiques soumis à des contraintes de temps. Ainsi, de nombreux systèmes de contrôle-commande en charge de contrôler un système physique que l'on nomme un procédé (petit périphérique dit électronique, automobile, avion, etc.) sont des systèmes temps réel. Bien que d'autres catégories de programmes puissent être qualifiés de systèmes temps réel (transactions financières, etc.) la communauté temps réel assimile souvent système temps réel et contrôle-commande de procédé. Ainsi, on présente souvent les contraintes de temps d'un système temps réel comme inhérentes à la dynamique du procédé contrôlé : le système doit réagir "suffisamment vite". L'étude de cas du drone AMADO montre qu'il exite d'autres types de contraintes inhérentes à la communication avec le matériel. Le non respect des contraintes de temps peut être plus ou moins dommageable pour le système, entraînant des effets allant d'une perte peu sensible en qualité de service, à des dommages catastrophiques pour le procédé contrôlé ou son environnement. Le système acquiert des données sur l'état d'un procédé à l'aide de capteurs, et le commande à l'aide d'actionneurs ; il est relié, pour les précédés complexes, à plusieurs autres systèmes (nous qualifions alors chaque système de sous-système), à l'aide de moyens de communication (bus de terrain, liaison sans fil, etc.). Dans ce cas, on parle de systèmes répartis. Les systèmes temps réel sont considérés comme des systèmes réactifs, car ils interagissent continuellement avec leur environnement à la différence des système transformationnels, et doivent respecter des contraintes de temps à la différence des systèmes interactifs.

Deux familles d'approches peuvent être employées pour implémenter un système temps réel : l'approche synchrone et l'approche asynchrone.

- L'approche synchrone se base sur l'hypothèse synchrone, c'est-à-dire que le système est capable de traiter un événement entrant (par exemple arrivée d'une trame par le réseau) avant l'arrivée du prochain événement. Cette hypothèse étant très restrictive, on est amené dans la réalité à utiliser des routines de traitement d'interruption afin de mémoriser un événement au moment où il arrive. Parallèlement à cela, on exécute périodiquement le traitement des derniers événements arrivés. Dans ce cas, il suffit de montrer que la durée maximale de traitement est inférieure ou égale à la période choisie, et que l'écart temporel maximum entre l'arrivée effective d'un événement et son traitement est acceptable. Dans une telle approche, la conception du système peut être parallèle, mais il est linéarisé à la compilation sous la forme d'un modèle proche d'un automate fini représentant les changements d'état du système.
- L'approche asynchrone admet un parallélisme de conception et d'implémentation, et ne formule pas d'hypothèse a priori sur les durées globales de traitement des événements. Cependant, elle nécessite une étape de validation temporelle démontrant que les contraintes de temps de chaque tâche sont toujours respectées à l'exécution. La validation temporelle d'un système temps réel consiste à vérifier que les contraintes de temps du système sont respectées durant toute la vie de l'application. On distingue systèmes temps réel à contraintes strictes (le respect des contraintes de temps doit être total), et à contraintes relatives (certaines contraintes peuvent parfois ne pas être respectées). Je me suis intéressé principalement aux systèmes

temps réel à contraintes strictes sous l'hypothèse asynchrone.

On pourrait arguer qu'avec l'augmentation de puissance et la diminution du coût des calculateurs, l'approche asynchrone pourrait disparaître. Cependant, dans le contexte des systèmes embarqués, étant donné qu'il y a des contraintes de robustesse, d'encombrement, de dissipation de chaleur, et d'énergie consommée [Bin07], la puissance des calculateurs est très faible en comparaison de celle les calculateurs équipant des ordinateurs personnels. Ainsi, pour le spatial, on utilise des microcontrôleurs endurcis d'une fréquence de l'ordre de la dizaine de MHz. Dans les industries produisant des systèmes en série, on essaie de réduire les coûts en dimensionnant au plus juste le support de calcul. De plus, le nombre de fonctions fournies par les systèmes augmente, par exemple il a plus que doublé en 10 ans dans le domaine automobile.

Les calculateurs utilisés sur les systèmes embarqués peuvent être des assemblages de microcontrôleurs, Digital Signal Processing (DSP), Application-Specific Integrated Circuit (ASIC) et ou Field-Programmable Gate Array (FPGA). On peut s'attendre, dans les années à venir, à voir se multiplier les microcontrôleurs multicœurs comme c'est le cas pour les microprocesseurs des ordinateurs personnels (pour une puissance théorique de calcul équivalente, moindre consommation d'énergie, moindre déperdition calorifique, etc.), d'où l'intérêt croissant des chercheurs de la communauté pour des modèles multiprocesseurs. Les parties sujettes à des modifications courantes sont implémentées sur microcontrôleur ou FPGA (le DSP étant nettement plus coûteux), les ASIC étant réservé aux fonctionnalités consommatrices de temps processeur (acquisition rapide du signal, acquisition numérique sonore, d'image, etc.) et peu sujettes à modifications au cours de la vie du produit.

Nos recherches concernent les approches asynchrones, pour des applications principalement développées sur microcontrôleur, dans le cas monoprocesseur, réparti, et multiprocesseur dans une moindre mesure. Les ASIC, FPGA ou autres composants pouvant être liés par des bus de communication externes ou internes à la puce contenant le microcontrôleur (cas des System-on-Chip (SoC)), sont vus comme des périphériques, ou simplement des capteurs reliés par bus de communication au calculateur.

L'implémentation des systèmes temps réel que nous considérons se base sur un exécutif temps réel fournissant des primitives de gestion du multitâche, des primitives temps réel (outils de synchronisation comme les sémaphores ou les moniteurs, de communication synchrone faiblement couplée comme les boîtes aux lettres ou files First In First Out (FIFO), ou encore de communication synchrone fortement couplée comme les rendez-vous ou les barrières). De plus en plus d'exécutifs offrent des interfaces de programmation répondant à des normes pour le temps réel comme Portable Operating System Interface (POSIX), Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen (OSEK), Ada, Avionics Application Standard Software Interface (ARINC 653), ou peuvent avoir une interface de programmation propriétaire comme VxWorks de WindRiver, ou Real-Time Executive for Multiprocessor Systems (RTEMS). La plupart des exécutifs industriels procurent des techniques issues de la recherche en ordonnancement temps réel dans les 25 dernières années. Ainsi, afin d'éviter l'inversion de priorités, avec un ordonnanceur à priorités fixes, on trouvera le protocole à priorité plafond [SRL90] (implémenté sous la forme de l'héritage immédiat de priorité [Kai82]) dans la plupart des exécutifs pour les sémaphores ou les moniteurs, ou bien le protocole à priorité héritée dans le cas de VxWorks. On peut déplorer l'absence d'ordonnanceurs à priorités dynamiques dans

les exécutifs industriels, bien que la norme Ada 2005 l'introduise. Certaines applications utilisent un dispositif simple basé sur un séquenceur, qui pourra par exemple remplacer l'exécutif, qui pourra suivre une séquence prédéfinie d'ordonnancement de tâches. Dans le cas où on utilise un ordonnanceur, on parle d'ordonnancement en-ligne, dans le cas où on utilise un séquenceur, on parle d'ordonnancement hors-ligne.

Il existe deux types d'implémentation possibles sur exécutifs : une implémentation dirigée par le temps et une implémentation dirigée par les événements :

- Implémentation dirigée par le temps : toute tâche est activée sur occurrence de temps, par exemple périodiquement, ou bien à une certaine date. Dans ce cas, les tâches du système sont dites concrètes : leurs dates d'activation sont connues à l'avance. Etant donné que de nombreux événements (arrivée de données sur un bus de communication, capteur actif, etc.) signalent leur occurrence par interruption matérielle, le mécanisme utilisé consiste à utiliser une routine de traitement d'interruption qui mémorise l'événement (par exemple dans une variable globale) jusqu'à son traitement par la tâche d'acquisition correspondante.
- Implémentation dirigée par les événements : les tâches sont soit activées par le temps, comme dans le cas précédent, soit activées directement par l'occurrence d'un événement (typiquement une interruption matérielle). La routine de traitement d'interruption déclenche alors directement la tâche d'acquisition correspondante. Dans ce cas, on ne connaît pas à l'avance les dates d'activation de la tâche, et on parle de tâche non concrète. Etant donné qu'il nous faut, dans le cadre de systèmes temps réel à contraintes strictes, maîtriser les activations au pire des tâches, les événements déclencheurs de tâches sont caractérisés temporellement. Typiquement, les déclenchements de la tâche seront espacés, soit par dispositif matériel, soit par protection logicielle, d'un écart minimal connu : on parle alors de tâche sporadiquement périodique. Dans le pire des cas, une telle tâche est considérée comme périodique, bien que dans la réalité, ses réveils puissent être espacés davantage.

L'avantage de l'approche dirigée par le temps dans l'industrie est le sentiment de maîtrise du système (bien que le système soit tout autant maîtrisé dans une implémentation dirigée par les événements), la possibilité d'utiliser un ordonnancement en-ligne ou hors-ligne, et la possibilité pour les tâches concrètes d'utiliser adroitement le décalage des tâches de façon à lisser la charge. L'avantage d'une implémentation dirigée par les événements est que le système est à même, pour une puissance de calcul équivalente, de répondre plus rapidement aux événements. Cependant, dans ce cas, seul l'ordonnancement en-ligne peut être utilisé.

1.1.2 Modèles de tâches

Au long de ce document, différents modèles de tâches temps réel sont manipulés, dans différents contextes. Cette section introduit donc les notations qui sont utilisées pour définir précisément les problèmes d'ordonnancement temps réel.

Tâches à base périodique

La plupart des tâches composant un système temps réel dirigé par le temps sont périodiques, et nous avons vu dans la section précédente qu'on assimilait aussi les tâches spora-

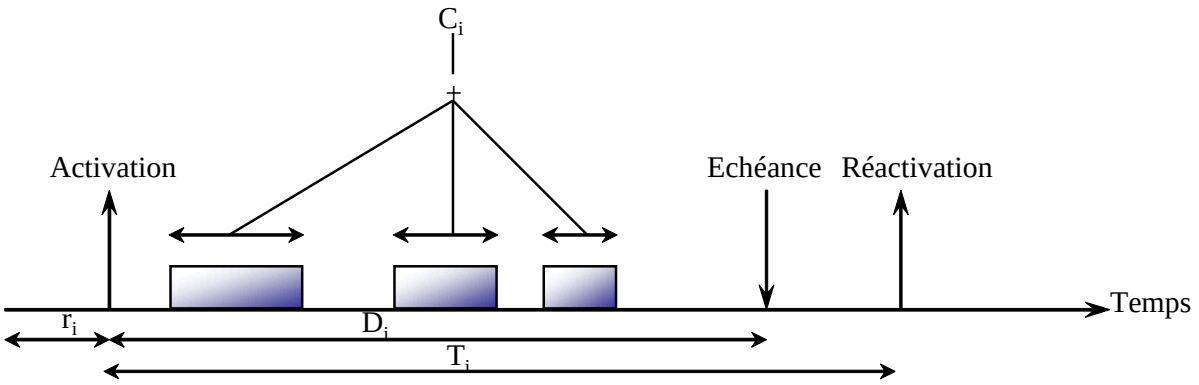

FIGURE 1.1 – Modèle de tâche concrète.

diquement périodiques à des tâches périodiques. Par conséquent, l'ordonnancement temps réel de base considère des systèmes de tâches périodiques. Ainsi, le modèle traditionnellement utilisé pour introduire le temps réel est le modèle de Liu& Layland [LL73], qui définit une tâche par sa pire durée d'exécution ou Worst-Case Execution Time (WCET), et sa période. Sa contrainte de temps est, dans ce cas de tâche basique, que la tâche se termine avant sa prochaine activation (requête) : on parle de tâche à échéance sur requête. La contrainte de temps associée à une tâche s'appelle le délai critique et représente le délai, relativement à sa date d'activation, accordé à la tâche pour s'exécuter avant son échéance. Voici les notations utilisées pour caractériser temporellement une tâche τ_i. Notons que par convention, nous utilisons des majuscules pour des délais, et des minuscules pour des dates.

Modèle de base

Le modèle de tâches de base, dit de Liu et Layland, est caractérisé par :

- r_i date du premier réveil (ou première activation) de la tâche. Ce paramètre n'a de sens que pour les tâches concrètes, puisque r_i est inconnu dans le cas de tâches non concrètes.
- C_i pire durée d'exécution ou WCET. Notons que cette pire durée est le plus souvent très supérieure à la durée moyenne.
- T_i période d'activation, dans le cas strictement périodique, la tâche est activée exactement toutes les T_i unités de temps à partir de sa première date d'activation ; dans le cas sporadiquement périodique, T_i représente le délai minimal entre eux activations successives.
- D_i est le délai critique d'une tâche, i.e. délai accordé avant échéance. Lorsque $D_i <= T_i$ on parle d'échéance contrainte, lorsque $D_i = T_i$ la tâche est à échéance sur requête, lorsque $D_i > T_i$ (i.e. la tâche est autorisée à terminer après sa prochaine date d'activation) on a affaire à une échéance arbitraire. Notons que dans ce cas, implicitement, la tâche est non ré-entrante, c'est-à-dire que la prochaine instance ne peut pas commencer à s'exécuter avant la fin de l'instance précédente.

La figure 1.1 représente une tâche τ_i concrète caractérisée par $< r_i, C_i, D_i, T_i >$. La charge d'un système est donnée par :

- $U_i = C_i/T_i$ est la charge d'une tâche, c'est-à-dire la fraction du processeur qu'il faut

lui consacrer.

Paramètres liés aux instances de tâches périodiques

- $r_{i,k}$ date de réveil de l'instance k de la tâche τ_i. Nous prenons $r_{i,0} = r_i$. Dans le cas d'une tâche concrète strictement périodique, cette date est calculable statiquement et vaut $r_{i,k} = r_i + kT_i$.
- $d_{i,k}$ échéance de l'instance k de τ_i. $d_{i,k} = r_{i,k} + D_i$.

Paramètres spécifiques

- J_i gigue de démarrage de la tâche, représente le délai pouvant retarder chaque instance d'une tâche après sa date d'activation au plus tôt. L'instance k d'une tâche peut donc être prête au plus tôt à la date $r_{i,k}$ et au plus tard à la date $r_{i,k} + J_i$. Ce paramètre est généralement utilisé pour représenter une attente de message provenant d'un autre calculateur dans le cadre d'une analyse holistique. Ainsi, un modèle intégrant ce paramètre sera susceptible d'être utilisé dans une analyse holistique de système réparti.
- B_i facteur de blocage d'une tâche, représente l'interférence (i.e. retard) qu'une tâche peut subir de la part d'une tâche moins prioritaire, suite à l'utilisation d'un protocole de gestion de ressources permettant d'éviter l'inversion de priorité.

Paramètres liés à l'ordonnancement des tâches

- $TR_{i,k}$ est le temps de réponse de l'instance k de τ_i. L'instance respecte son échéance si $TR_{i,k} \leq D_i$.
- TR_i est le temps de réponse maximal de τ_i, $TR_i = \max_{\forall k}(TR_{i,k})$. Une tâche est fiablement ordonnancée si $TR_i \leq D_i$.

Il est primordial de conserver un lien entre réalité des applications temps réel et paramètres temporels. Ainsi, il est important de garder à l'esprit le fait que C_i est une borne supérieure de durée d'exécution, et que la durée effective pourra être arbitrairement petite. De même, les contraintes de temps, choisies par le concepteur au regard du cahier des charges, ne sont pas forcément immuables et peuvent souvent être négociées. La date de réveil r_i n'a de sens que dans le cas de tâches concrètes, et cette date peut avoir été choisie par le concepteur pour aider à l'ordonnançabilité. Quant au facteur de blocage B_i, il peut être une borne supérieure non atteignable dans le cas de tâches concrètes. Le facteur J_i ne peut pas être anticipé, ainsi, un ordonnancement hors-ligne de tâches ayant une gigue de démarrage ne pourrait faire démarrer cette tâche qu'après sa gigue d'activation maximale ; si ce n'était pas le cas et que J_i représente le délai maximal d'attente d'un message en provenance du réseau, la tâche pourrait être exécutée avant arrivée du message attendu.

Des facteurs pratiques interviennent dans la quasi totalité des applications réelles, il s'agit de :

- l'exclusion mutuelle, mécanisme de protection contre l'utilisation simultanée de ressources critiques.
- les contraintes de précédence, représentant la communication synchrone faiblement couplée entre des tâches (type boîte aux lettres ou synchronisation par sémaphore privé).

D'autres facteurs pratiques sont très peu étudiés de nos jours, ainsi la communication

synchrone fortement couplée, de type rendez-vous, a été étudiée dans [BW97], mais est écartée suite à l'adoption quasi unanime par les concepteurs du profil Ravenscar [Bur99]. Il en va de même pour la préemption : la plupart des applications l'autorisent, donc sauf mention contraire dans la suite, les tâches sont préemptibles.

Tâches non périodiques

Les tâches ni périodiques ni sporadiquement périodiques sont vues en terme d'instances ; elles sont, d'après la terminologie de [Liu00] soit sporadiques et caractérisées par $< r_i, C_i, D_i >$ (noter que contrairement aux tâches sporadiquement périodiques, aucun délai minimal ne sépare deux instances successives) soit apériodiques et caractérisées uniquement par $< r_i, C_i >$. Dans les deux cas, r_i n'est connu que dynamiquement lors de l'arrivée des événements déclencheurs. Les apériodiques sont traitées avec la stratégie du meilleur effort (on tente de minimiser le temps de réponse sans nuire à l'ordonnançabilité des autres tâches). On ne peut pas valider temporellement un système constitué de tâches sporadiques, par conséquent, la technique retenue, étudiée dans la communauté dans les années 90, est un test d'acceptation en-ligne de complexité très faible, permettant d'accepter ou de refuser une tâche sporadique. Si l'on n'utilise que les temps creux laissés par l'ordonnancement des tâches périodiques ou sporadiquement périodiques, on peut retarder inutilement l'exécution du trafic non périodique. Ainsi la littérature propose deux familles de méthodes :

- Utilisation de serveur périodique dont la bande passante sera utilisée par les non périodiques. Cependant, s'il n'y a pas de trafic non périodique lors de l'exécution du serveur, on pourra se retrouver dans le même cas que lorsqu'il n'y a pas de serveurs si les requêtes non périodiques arrivent à un mauvais moment. L'idée développée dans la littérature est donc de préserver la bande passante du serveur. C'est le cas pour le serveur ajournable [LSS87, Str88], adapté à Earliest Deadline First (EDF) dans [GB95]. Le problème d'un tel serveur est qu'il possède un effet de bord ressemblant à celui de la gigue de démarrage : il existe des périodes dans lesquelles le serveur peut utiliser plus que sa capacité, ce qui nuit à l'ordonnançabilité globale. Le serveur sporadique [SSL89, GB95] évite ce phénomène. Pour un ordonnanceur à priorités dynamiques, on peut citer les serveurs à utilisation constante [DLS97, SB96].
- Le vol de temps creux : dans ce cas, on utilise la laxité des tâches (différence entre le temps nécessaire à la terminaison de l'instance courante et temps restant avant échéance). La laxité exprime le retard que l'on peut appliquer à une tâche sans nuire à son ordonnançabilité. [LRT92] propose une telle approche pour un ordonnanceur à priorités fixes, et [CS89] pour un ordonnanceur à priorités dynamiques, utilisant Earliest Deadline Last (EDL).

Pour plus de clarté, ce mémoire utilise une notation inspirée de la théorie de l'ordonnancement pour caractériser les problèmes d'ordonnancement temps réel.

Architecture	
$\emptyset$	système monoprocesseur
distr	systèmes monoprocesseurs répartis
multi	système multiprocesseur à processeurs identiques
uniform	système multiprocesseur à processeurs uniformes (chaque processeur est caractérisé par une capacité s_i)
unrelated	système multiprocesseur à processeurs non comparables (chaque tâche a un WCET différent en fonction du processeur, correspond à une architecture à processeurs dédiés)
distr,mult	on peut lier distr avec un type de système multiprocesseur (exemple : distr,uniform définit une architeture distribuée composée de systèmes multiprocesseurs uniformes)
Modèle temporel	
Type de réveil	
$0, C_i[, ...]$	tâches concrètes simultanées
$r_i, C_i[, ...]$	tâches concrètes différées
$C_i[, ...]$	tâches non concrètes
Type de réveil	
$[...,]C_i, T_i$	échéance sur requête, par exemple $0, C_i, T_i$ est un système de tâches concrètes, simultanées à échéance sur requête
$[...,]C_i, D_i \leq T_i, T_i$	échéance contrainte
$[...,]C_i, D_i, T_i$	échéance arbitraire
Facteurs pratiques	
res	présence d'exclusions mutuelles
B_i	présence d'un facteur de blocage maximal dû à un protocole de gestion de ressources
J_i	présence d'une gigue de démarrage
$\prec$	contraintes de précédence entre tâches de même période
$\prec_{gen}$	contraintes de précédence généralisées (pouvant concerner des tâches de période différente)
susp	Certaines tâches se suspendent (par exemple initialisation d'une entrée/sortie, attente de l'entrée/sortie, reprise)
nopreempt	Tâches non préemptibles (par défaut on considère des tâches préemptibles)
preemptnopreempt	Certaines tâches ont des parties non préemptibles. Remarque : les cas *nopreempt* et *preemptnopreempt* sont modélisables en général par les cas *res*

1.1.3 Algorithmes d'ordonnancement

Dans cette section, nous donnons quelques définitions de base de la théorie de l'ordonnancement temps réel, afin d'exprimer les résultats d'optimalité et de complexité connus.

Ordonnançabilité

Définition 1 *Une tâche τ_i est fiablement ordonnancée par un algorithme d'ordonnancement si et seulement si (ssi) toutes ses instances respectent leur échéance, c'est-à-dire si $TR_i \leq D_i$.*

Notons que cette définition a une vue intemporelle de l'ordonnançabilité : la tâche doit respecter toutes les échéances durant la vie de l'application, qui est potentiellement infinie.

Définition 2 *Un système $S = \{\tau_i\}_{i=1..n}$ est fiablement ordonnancé par un algorithme d'ordonnancement ssi toutes ses tâches sont fiablement ordonnancées.*

Définition 3 *Un algorithme d'ordonnancement A est plus puissant qu'un algorithme d'ordonnancement B pour un contexte de problèmes ssi il ordonnance fiablement tout système de ce contexte ordonnancé fiablement par B.*

Notons que la notion de puissance d'ordonnancement donnée dans cette définition n'est pas stricte, ainsi si A est plus puissant que B et que B est plus puissant que A alors A et B sont de puissance équivalente.

Définition 4 *Un algorithme d'ordonnancement est optimal dans une classe d'algorithmes pour un contexte de problèmes ssi il est plus puissant que n'importe quel algorithme de cette classe pour le contexte considéré.*

Ces définitions donnent une vue "algorithme" du problème, la définition suivante donne une vue "application".

Définition 5 *Un système $S = \{\tau_i\}_{i=1..n}$ est ordonnançable ssi il existe un algorithme permettant de l'ordonnancer fiablement. Il est ordonnançable dans une classe d'algorithmes ssi il existe un algorithme de cette classe l'ordonnançant fiablement.*

Propriété 1 *En présence d'un algorithme optimal, un système est ordonnançable dans une classe d'algorithmes ssi il est ordonnancé fiablement par un algorithme optimal de cette classe.*

Algorithmes d'ordonnancement

Dans le cas de l'ordonnancement hors-ligne, le modèle de tâche étudié sert à générer une séquence d'ordonnancement valide qui sera utilisée lors de l'exécution par un séquenceur, alors que l'ordonnancement en-ligne consiste à valider une politique d'attribution de priorités qui sera appliquée par un ordonnanceur pendant l'exécution du système. La politique d'attribution des priorités est soit à priorités fixes, dite Fixed Priority Policy (FPP), soit à priorités dynamiques. Les algorithmes en-ligne les plus connus sont :

- FPP
  - Rate Monotonic (RM) [Ser72, LL73] : donne une priorité inversement proportionnelle à la période, est utilisé dans le contexte des tâches à échéance sur requête. Algorithme optimal dans la classe des algorithmes FPP dans les contextes $|0, C_i, T_i|$ et $|C_i, T_i|$.

  - Deadline Monotonic (DM) [LW82] : la priorité est inversement proportionnelle au délai critique. Equivalent à RM dans le contexte de tâches à échéance sur requête. Algorithme optimal dans la classe des algorithmes FPP dans les contextes $|0, C_i, D_i, T_i|$ et $|C_i, D_i, T_i|$.
  - affectation d'Audsley [Aud91] : algorithme d'affectation de priorités utilisé dans le cas de systèmes de tâches concrètes non simultanées. Algorithme optimal dans la classe des algorithmes FPP dans les contextes $|r_i, C_i, D_i, T_i|$ et $|C_i, D_i, T_i|$.
- Priorités dynamiques
  - EDF [Jac55, Hor74, Der74, Lab74, LL73] : plus l'échéance est proche, plus la priorité est grande. Optimal dans les contextes $|r_i, C_i, D_i, T_i|$ et $|C_i, D_i, T_i|$. Condition nécessaire et suffisante (CNS) d'ordonnançabilité dans les contextes $|r_i, C_i, T_i|$ et $|C_i, T_i|$: $U = \sum_{i=1..n} U_i \leq 1$
  - Minimal Laxity (ML) ou Least Laxity (LL) [LW82, Mok83] : plus la laxité est faible (distance entre temps de traitement restant et échéance), plus la priorité augmente. Mêmes propriétés d'optimalité qu'EDF en monoprocesseur. ML est basé sur les quanta de temps (i.e. une décision d'ordonnancement peut intervenir sans occurrence d'un événement particulier comme le réveil, la terminaison, ou le changement d'état d'une tâche), ce qui introduit de nombreux changements de contexte.
  - Algorithmes Pfair (Proportionate fairness) [BCPV96, AHS05] dont certains sont optimaux dans le contexte multiprocesseur global (dans le contexte multiprocesseur, on distingue contexte global - migrations autorisées à n'importe quel moment - et partitionné) multi$|C_i, T_i|$, et pour lesquels une CNS d'ordonnançabilité est, sur m processeurs identiques, $U = \sum_{i=1..n} U_i \leq m$ et $U_i \leq 1$. Le principe des algorithmes Pfair repose sur le principe d'équité par rapport à la charge : cela consiste, idéalement, à allouer dans toute fenêtre temporelle [0,t[$U_i \times t$ unités de temps. Etant donné que l'ordonnanceur considéré est basé sur des quanta de temps, car une préemption ne peut avoir lieu qu'à la granularité de l'ordonnanceur, la propriété Pfair est définie telle que le temps consacré à la tâche τ_i sur [0,t[se situe entre $\lfloor U_i \times t \rfloor$ et $\lceil U_i \times t \rceil$. Chaque instance de tâche est alors divisée en sous-tâches devant s'exécuter dans les fenêtres correspondantes à la propriété Pfair. Trois algorithmes Pfair optimaux existent : PF [BCPV96], PD [BGP95], et PD^2 [AS00], ils sont basés sur un algorithme EDF pour chaque sous-tâche, et diffèrent par la façon de résoudre les ex-æquos. Ces algorithmes sont optimaux, cependant en termes de performances, ils se comportent mal avec l'utilisation de ressources critiques, et entraînent de nombreuses préemptions et migrations entraînant un surcoût processeur important [SHAB03].

La figure 1.2 montre une classification des algorithmes d'ordonnancement. En présence de parties non préemptibles (et donc en présence de ressources critiques), les algorithmes conservatifs ne sont pas optimaux (i.e. ne laissant pas le processeur oisif lorsqu'au moins une tâche est prête). Au contraire, on peut trouver des séquences d'ordonnancement non conservatives optimales. Les algorithmes non conservatifs se basent généralement sur une vue totale du système (connaissance des dates de réveil des tâches à venir), on parle d'algorithmes clairvoyants, au contraire des algorithmes en-ligne qui n'ont connaissance que des tâches actives.

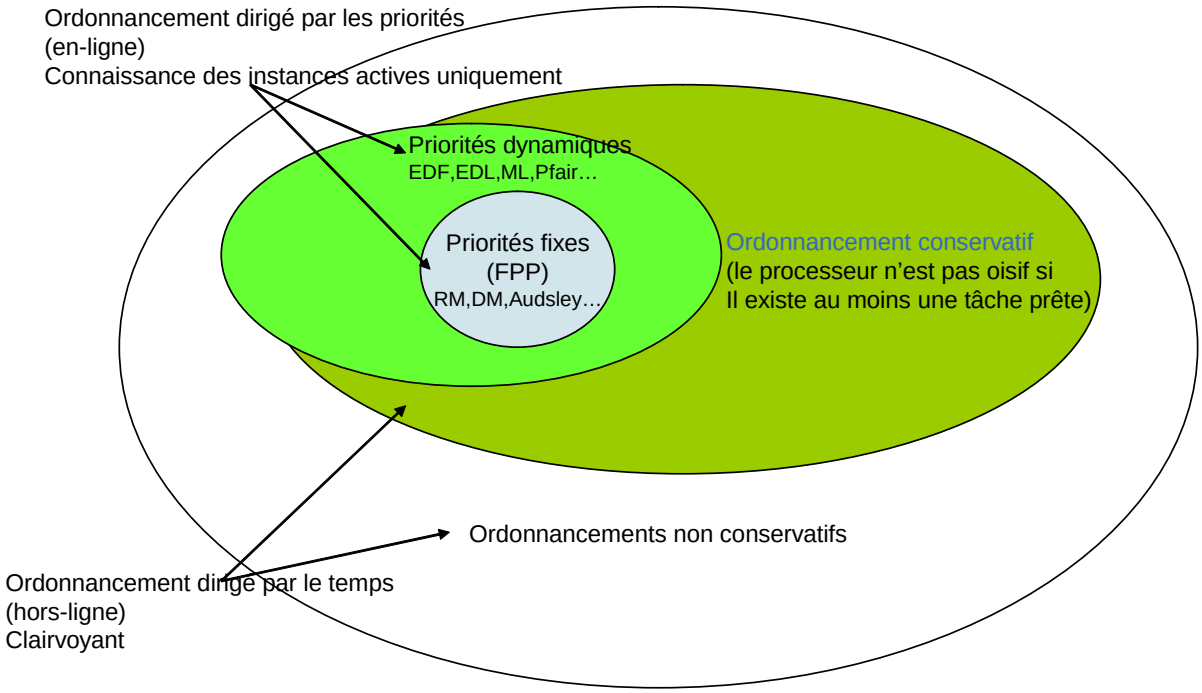

FIGURE 1.2 – Familles d'algorithmes d'ordonnancement classées par puissance d'ordonnançabilité.

1.1.4 Tests d'ordonnançabilité

Le problème de dire si un système est ordonnançable, indépendamment de l'algorithme, est un problème complexe, à part dans les contextes $multi|C_i, T_i|$, $|0, C_i, T_i|$, et $|C_i, T_i|$, où l'on dispose d'un test d'ordonnançabilité de complexité polynomiale. Dès lors que le système est concret avec des réveils différés, le problème est Co-NP-complet au sens fort [BHR90, LW82]. Dès lors que des ressources critiques sont utilisées, le problème de décider de l'ordonnançabilité d'un système de tâches est NP-difficile au sens fort (complexité au moins aussi dure que les problèmes NP-complets, mais l'appartenance à la classe NP n'est pas montrée) [Mok83], il en va de même lorsque les tâches sont non préemptibles, ou lorsque les tâches se suspendent [Ric03].

Les tests utilisés pour vérifier si un système est fiablement ordonnancé par un algorithme sont soit approchés et polynomiaux, soit exacts mais pseudo-polynomiaux ou faiblement exponentiels. Il existe 4 familles de méthodes :

- méthodes analytiques, dont un bon nombre est basé sur l'étude de la charge processeur,
- calcul de temps de réponse, basé sur le calcul de la période d'activité,
- vérification du respect des échéance, reposant sur l'étude de la fonction de demande processeur,
- simulation, reposant sur la construction de l'ordonnancement.

Notons que dans le cas des méthodes hors-ligne, le test d'ordonnançabilité est intégré dans la méthode de construction de séquence d'ordonnancement.

Méthodes analytiques

Les méthodes analytiques sont, pour certains contextes, des Conditions Nécessaires et Suffisantes (CNS) d'ordonnançabilité, et pour la plupart des Conditions Suffisantes (CS) d'ordonnançabilité applicables pour certains algorithmes dans des contextes simples (voir [ABD+95]).

Ainsi, il est trivial qu'un système n'est pas ordonnançable sur m processeurs si sa charge $U \geq m$. Dans les contextes monoprocesseurs, de tâches indépendantes concrètes à échéance sur requête $|r_i, C_i, T_i|$ et non concrètes $|C_i, T_i|$, EDF possède une CNS $U \leq 1$ [LL73]. On trouve un résultat équivalent pour les algorithmes Pfair dans les contextes m processeurs identiques, avec migration pour le contexte $multi|0, C_i, T_i|$, et $|C_i, T_i|$ où la CNS d'ordonnançabilité est $U \leq m$.

Pour les algorithmes à priorités fixes, même dans ce contexte très simple en monoprocesseur, nous ne disposons que de CS polynomiales. Ainsi dans les contextes $|0, C_i, T_i|$ et $|C_i, T_i|$, la Condition suffisante (CS) la plus connue pour l'algorithme RM est $U \leq n(2^{1/n} - 1)$ pour n tâches [LL73]. Notons que la preuve originale est fausse et a été corrigée dans [DG00]. Cette CS est rendue obsolète par la CS $\prod_{i=1..n} U_i + 1 \leq 2$ [BBB03] qui est la meilleure condition d'ordonnançabilité possible que l'on puisse obtenir pour un algorithme à priorité fixe.

Dès lors que l'échéance peut être inférieure ou égale à la période, aucune CS polynomiale n'est efficace.

Calcul de temps de réponse

Le calcul de temps de réponse ou Response Time Anayslis (RTA) s'effectue par étude de la période d'activité [JP86], pour des tâches à priorités fixes.

Définition 6 *Dans le contexte FPP, on définit l'ensemble des indices de tâches au moins aussi prioritaires que τ_i par*

$$hp(i) = \{j \neq i \text{ tel que } \tau_j \text{ est au moins aussi prioritaire que } \tau_i\}$$

Théorème 1 *[LL73] Instant critique pour une tâche : dans le contexte $|C_i, [D_i,]T_i|$, le pire temps de réponse pour une tâche τ_i advient lorsque toutes les tâches de $hp(i)$ sont activées simultanément avec la tâche étudiée, cet instant s'appelle l'instant critique.*

Note : $[D_i,]$ signifie que la présence d'un délai critique, qu'il soit inférieur, égal, ou supérieur à la période, n'a pas d'importance ici car le résultat concerne le calcul du temps de réponse et pas l'ordonnançabilité.

Le théorème 1 explique que les résultats concernant le contexte $|0, C_i, [D_i,]T_i|$ s'appliquent au contexte $|C_i, [D_i,]T_i|$, puisque pour des tâches indépendantes non concrètes, le pire cas correspond à un réveil simultané.

Définition 7 *[Leh90] En priorité fixe, une période d'activité du processeur de niveau i est un intervalle de temps durant lequel le processeur est pleinement utilisé, seulement par des tâches de priorité supérieure ou égale à i.*

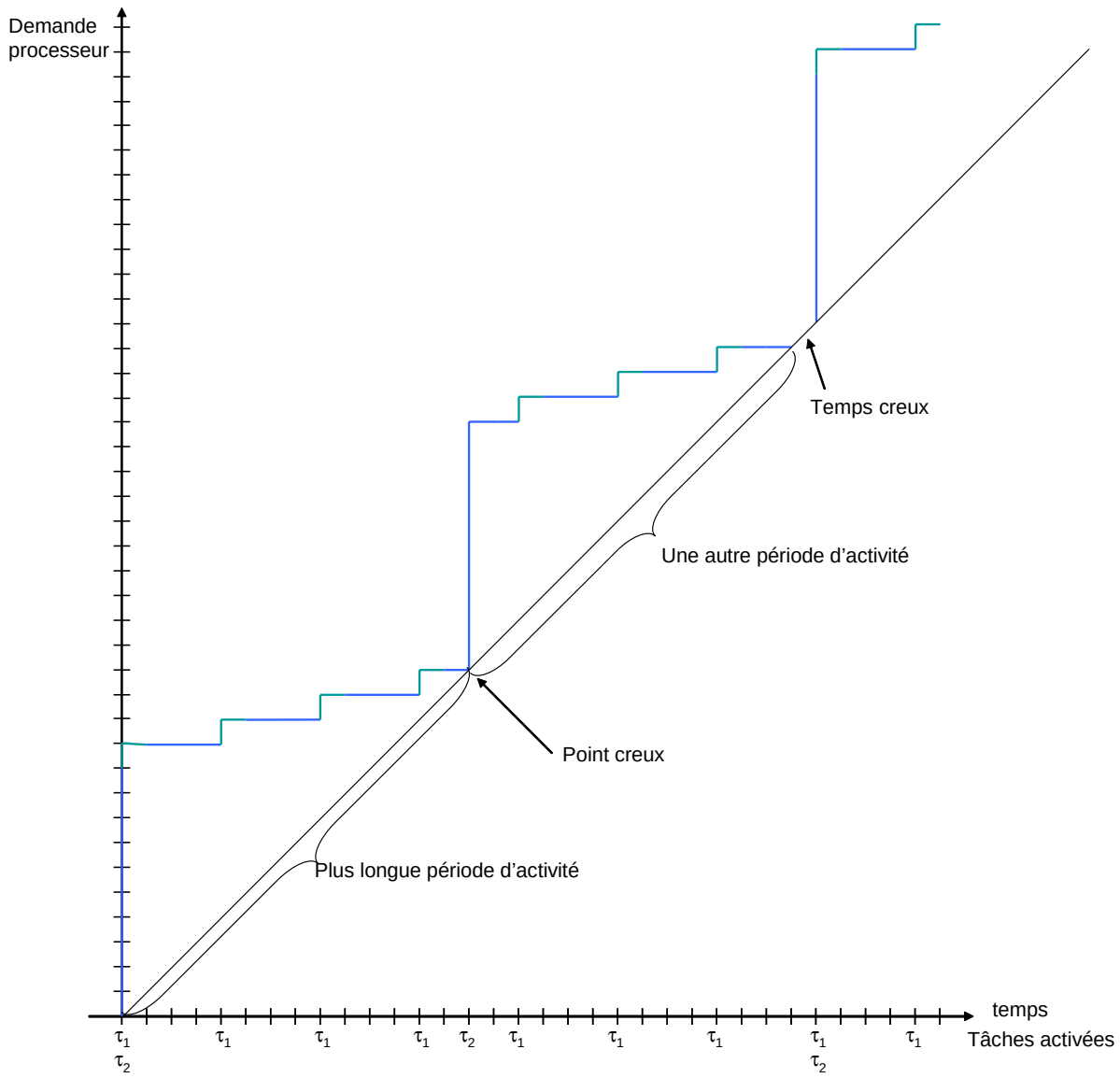

FIGURE 1.3 – Périodes d'activité.

L'allure des périodes d'activité de niveau 2 est représentée sur la figure 1.3 pour un système quelconque de tâches indépendantes dont les deux tâches les plus prioritaires sont $\tau_1 =< C_1 = 1, D_1, T_1 = 4 >$ et $\tau_2 =< C_2 = 10, D_2, T_2 = 14 >$. Notons que le délai critique n'intervient pas, et que les tâches moins prioritaires n'interviennent pas non plus.

Théorème 2 *[Aud91] Dans le contexte $|C_i, [D_i,]T_i|$ la plus longue période d'activité de niveau i démarre à un instant critique de niveau i (lorsque toutes les tâches de $hp(i)$ sont activées simultanément avec τ_i) et est donnée par l'activation des tâches à leur rythme maximum (cas des tâches sporadiquement périodiques).*

Par conséquent, le plus long temps de réponse d'une tâche se trouve dans la période d'activité initiée par un instant critique. La longueur de la période d'activité de niveau i

initiée par un instant critique est la plus petite solution non nulle de l'équation :

$$W_i(t) = \sum_{j \in \{i\} \cup hp(i)} \left\lceil \frac{W_i(t)}{T_j} \right\rceil C_j$$

Pour cette recherche de point fixe, la valeur initiale choisie pour t peut simplement être C_i ou bien être plus proche du point fixe [SH98]. Cette équation converge si et seulement si $U \leq 1$. Elle calcule la première intersection de la courbe de période d'activité avec la courbe $y = t$ qui représente la capacité de traitement du processeur. Une intersection s'appelle un point creux et termine une période d'activité. Un point creux peut d'ailleurs donner lieu à un temps creux si aucune tâche n'est à traiter ensuite. Notons que la complexité de ce calcul pseudo-polynomial est restée ouverte jusqu'à récemment, où [ER08] ont montré que le calcul de temps de réponse en FPP était NP-difficile au sens faible. Le temps de réponse n'est d'ailleurs pas approximable polynomialement [ER09], bien qu'on puisse approximer polynomialement l'augmentation de ressources (augmentation de vitesse du processeur) requise pour que le système soit ordonnançable [NRB09, BBNR07]. Notons que la complexité de l'ordonnançabilité d'un système par FPP dans ce contexte est encore ouverte.

Il reste maintenant à relier période d'activité et pire temps de réponse.

Théorème 3 *[Aud91] Dans le contexte $|r_i, C_i, [D_i,]T_i|$ le pire temps de réponse d'une tâche intervient lors de la plus longue période d'activité de son niveau de priorité.*

Dans le cas de tâches pouvant être réveillées simultanément (cas non concret et concret simultané) avec $D_i \leq T_i$, une tâche ne peut pas voir une instance interférer avec son instance suivante sans violer son échéance. Par conséquent τ_i ne peut intervenir qu'une seule fois dans une période d'activité sous peine de violer son échéance.

$$W_i(t) = C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i(t)}{T_j} \right\rceil C_j \text{ pour } W_i(t) \leq T_i$$

D'après le théorème 2, $W_i(t)$ donne la plus longue période d'activité de niveau i et d'après le théorème 3, le pire temps de réponse de τ_i se trouve dans cette période d'activité. Mais on ne peut y trouver, si cette période est plus courte que T_i, que la première instance. C'est cette instance qui possède donc le plus long temps de réponse. Etant donné que toutes les tâches représentées ont une priorité au moins aussi grande que celle de τ_i, le point creux représente la terminaison de τ_i. Puisque τ_i est activée à l'origine [JP86] :

$$TR_i = W_i(t) \text{ pour } W_i(t) \leq T_i$$

Si les échéances peuvent être plus grande que les périodes, ou tout simplement si l'on souhaite connaître le pire temps de réponse même en cas de dépassement d'échéance, il nous faut considérer toutes les instances se trouvant dans la période d'activité initiée par l'instant critique. Sachant que nous considérons des tâches non ré-entrantes, la technique proposée dans [Leh90] consiste à calculer la longueur de la période d'activité en considérant 1 instance de τ_i comme dans [JP86], ce qui permet d'obtenir le temps de réponse de la première instance $\tau_{i,0}$ puis, si cette période est supérieure à T_i, calculer la période

d'activité de la seconde instance $\tau_{i,1}$. Le temps de réponse est alors donné par la longueur de la période d'activité moins la date de réveil T_i de $\tau_{i,1}$, et ainsi de suite jusqu'à trouver un temps de réponse inférieur ou égal à T_i. Il s'agit alors de rechercher le plus petit point fixe non nul $W_i^{(k)}(t)$ de l'équation suivante en partant de $k = 1$, jusqu'à ce que le temps de réponse obtenu $TR_{i,k} \leq T_i$:

$$W_i^{(k)}(t) = kC_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i^{(k)}(t)}{T_j} \right\rceil C_j$$

$$TR_i^{(k)} = W_i^{(k)}(t) - (k-1)T_i$$

On peut être alors amené à considérer un nombre exponentiel de périodes d'activité avec un calcul pseudo-polynomial pour chacune. En effet la période d'activité maximale, atteignable dans le cas où $U = 1$, est l'hyperpériode $H = PPCM_{i=1..n}(T_i)$ (PPCM est le plus petit commun multiple). Par conséquent, pour une tâche τ_i dont la première instance de temps de réponse $\leq T_i$ est sa dernière instance dans l'hyperpériode, il nous faut construire H/T_i périodes d'activité.

Bien qu'exponentielle dans le cas d'échéances arbitraires, et pseudo-polynomiale même dans le cas le plus simple, cette méthode nécessite des temps de calcul très faibles même sur des études de cas imposantes. C'est la méthode utilisée dans la méthode Rate Monotonic Analysis (RMA) [HKO+93], car elle s'adapte à de nombreux facteurs pratiques.

Dans le cas de partage de ressources, on utilise un protocole de gestion de ressources pour éviter l'inversion de priorité (protocole à priorité héritée [SRL90], protocole à priorité plafond [SRL90], protocole à priorité plafond immédiat [Kai82], protocole d'allocation de la pile [Bak91]). L'inversion de priorité se produit lorsqu'une tâche de priorité faible est en section critique, et qu'une tâche de priorité intermédiaire s'exécute, alors qu'une tâche de priorité plus élevée est en attente de la ressource détenue par la tâche de priorité faible. Les protocoles de gestion de ressource introduisent un mécanisme d'héritage permettant d'éviter l'inversion de priorité, et de borner le nombre de sections critiques de priorité inférieure susceptibles de retarder une tâche. Nous ne détaillons pas ces protocoles ici. La pire interférence que peut subir une tâche à cause de la gestion des ressources est représentée par le facteur de blocage B_i. Ce facteur représente une borne supérieure de l'interférence maximale due à des tâches moins prioritaires qu'une tâche peut subir pendant une période d'activité de son niveau de priorité. Il est important de remarquer d'une part que B_i est une borne supérieure pas forcément atteignable, et que cette interférence ne peut avoir lieu qu'une seule fois par période d'activité, ce qui entraîne la modification suivante de la formule de calcul de pire temps de réponse en présence de ressources critiques :

$$W_i^{(k)}(t) = kC_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i^{(k)}(t)}{T_j} \right\rceil C_j$$

On peut voir le facteur B_i comme une marche initiale de la fonction de la période d'activité représentant l'interférence maximale due aux tâches moins prioritaires. En présence de ressources critiques (ou de tâches non préemptibles, puisque le principe est le même), le calcul de la période d'activité donne donc une borne supérieure de la période d'activité. Etant donné que le problème d'ordonnançabilité de systèmes de tâches partageant des

ressources est NP-difficile [Mok83], le problème du calcul exact de pire temps de réponse l'est aussi.

Lorsque des gigues d'activation J_i sont présentes, les travaux de Tindell permettent d'adapter la technique de RTA.

Théorème 4 *[Tin94a] Dans un système de tâches avec des gigues d'activation, l'instant critique d'une tache τ_i coïncide avec le réveil simultané de toutes les tâches de $hp(i)$ avec une activation retardée d'une valeur correspondante à leur gigue maximale.*

Ce théorème s'explique par le fait que l'interférence due aux tâches de plus haute priorité que la tâche analysée est maximale lorsque leurs réveils sont les plus fréquents. Ainsi, on considère un instant où les tâches plus prioritaires subissent une gigue maximale, de façon à ce que les activations des prochaines instances arrivent le plus tôt possible. La formule de calcul de la période d'activité à l'instant critique devient alors :

$$W_i^{(k)}(t) = kC_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{J_j + W_i^{(k)}(t)}{T_j} \right\rceil C_j$$

Le pire temps de réponse de la tâche étudiée est, quant à lui, augmenté de sa gigue d'activation puisque dans le pire des cas, elle a été elle-même activée avec sa gigue d'activation maximale, l'instant critique considéré a donc eu lieu à la date J_i relativement au départ de la première instance de τ_i.

$$TR_i^{(k)} = W_i^{(k)}(t) + J_i - (k-1)T_i$$

De nombreuses autres adaptations de la RTA existent, et la partie contribution montre que nous avons nous-même contribué à son adaptation à différents facteurs pratiques.

Marco Spuri [Spu96] a généralisé la RTA à l'algorithme EDF. Il montre qu'un instant critique correspond à un réveil simultané de toutes les tâches autres que la tâche étudiée, et que le pire temps de réponse ne peut advenir que lorsque l'échéance de la tâche étudiée coïncide avec une autre échéance. Il faut donc considérer toutes les périodes d'activité pouvant être construites en faisant coïncider l'échéance de la tâche étudiée avec chacune des échéances des autres tâches, ce qui donne un nombre pseudo-polynomial de périodes d'activités à calculer.

Analyse de la fonction de demande

L'analyse de la fonction de la demande processeur, dite Demand Bound Function (DBF), s'applique sur des ordonnancements dirigés par les échéances (typiquement EDF). Il s'agit de vérifier, pour les algorithmes optimaux, comme EDF, si les échéances sont respectées. La théorie a été proposée dans [JS93] et [BHR90]. La fonction $dbf(t_1, t_2)$ représente la charge des tâches activées après la date t_1 dont l'échéance survient avant t_2. La figure 1.4 représente l'évolution de la DBF du système constitué de deux tâches $|r_1 = 0, C_1 = 1, D_1 = 4, T_1 = 4|$ et $|r_2 = 0, C_2 = 10, D_2 = 14, T_2 = 14|$ entre les instants 0 et t. Nous pouvons y voir la différence entre dbf et période d'activité : la dbf représente le travail qui a du être traité pour respecter les échéances(la courbe doit donc toujours être inférieure à la puissance de traitement du processeur, représentée par $y = t$) alors que la

période d'activité (représentée en pointillés sur la figure) représente le travail nécessaire pour terminer le traitement des tâches actives.

La hauteur de la fonction $dbf(t_1, t_2)$ donne la valeur du temps que le processeur doit consacrer aux tâches activées entre les dates t_1 et t_2 pour respecter leurs échéances. La DBF est donnée par [BMR90, BHR90] :

$$dbf(t_1, t_2) = \sum_{i=1}^{n} \eta_i(t_1, t_2) C_i$$

où $\eta_i(t_1, t_2)$ représente le nombre d'instances de τ_i réveillées dans l'intervalle et ayant leur échéance dans l'intervalle.

Théorème 5 *[BHR90] Un système de taches est ordonnançable dans le contexte* $|r_i, C_i, D_i, T_i|$ *ssi* $U \leq 1$ *et* $\forall 0 \leq t_1 < t_2 \leq$ *durée de simulation des ordonnancements, on a* $dbf(t_1, t_2) \leq t_2 - t_1$.

Différents auteurs, dont nous même, avons travaillé ou travaillons sur le problème de la durée de simulation. A cet endroit du manuscrit, je citerai l'hyperpériode H pour les systèmes simultanés, et le résultat le plus ancien concernant la durée de simulation : $max_{i=1..n}(r_i) + 2H$, utilisé dans [BHR90], et correspondant au contexte $|r_i, C_i, D_i \leq T_i, T_i|$. La cyclicité, même monoprocesseur, dans le contexte des échéances arbitraires $|r_i, C_i, D_i, T_i|$ est encore un problème ouvert sur lequel je travaille actuellement en collaboration avec Joël Goossens et Liliana Cucu.

L'analyse de la DBF repose sur le constat qu'un dépassement d'échéance ne survient jamais lorsque le processeur est oisif. Il est prouvé que le premier dépassement d'échéance s'il existe, se produit dans la plus longue période d'activité. L'analyse d'ordonnançabilité est alors restreinte dans la plus longue période d'activité en vérifiant que toutes les échéances des tâches sont respectées dans cette période d'étude qui débute à l'instant $t = 0$ dans le cas tâches concrètes simultanées et tâches non concrètes. [BHR90] montre que la longueur de la période d'activité est limitée dans le cas U < 1 par

$$\frac{U}{1-U} \max_{i=1..n}(T_i - D_i)$$

Théorème 6 *[BHR90]Un système de tâches dans les contextes* $|0, C_i, D_i, T_i|$ *et* $|C_i, D_i, T_i|$ *est ordonnançable ssi pour toute date d'échéance d rencontrée dans la période d'activité,* $dbf(0, d) \leq d$.

La période d'activité est vue ici comme la période d'activité du système entier, et est calculée comme on le fait dans le cas des priorités fixes pour le plus bas niveau de priorité. Ainsi, la figure 1.4 montre que le système est ordonnançable par EDF : il suffit de vérifier que les échéances tombées aux dates 4, 8, 12 et 14 sont respectées ($dbf(0, t) \leq t$) entre 0 et la fin de la prémière période d'activité.

Le nombre d'instances à considérer $\eta_i(0, t)$ est donné par :

$$\eta_i(0, t) = max\left(0, \left\lfloor \frac{t - r_i - D_i}{T_i} \right\rfloor + 1\right)$$

Le même type d'adaptation de formule que pour la RTA est applicable pour prendre en compte les facteurs pratiques.

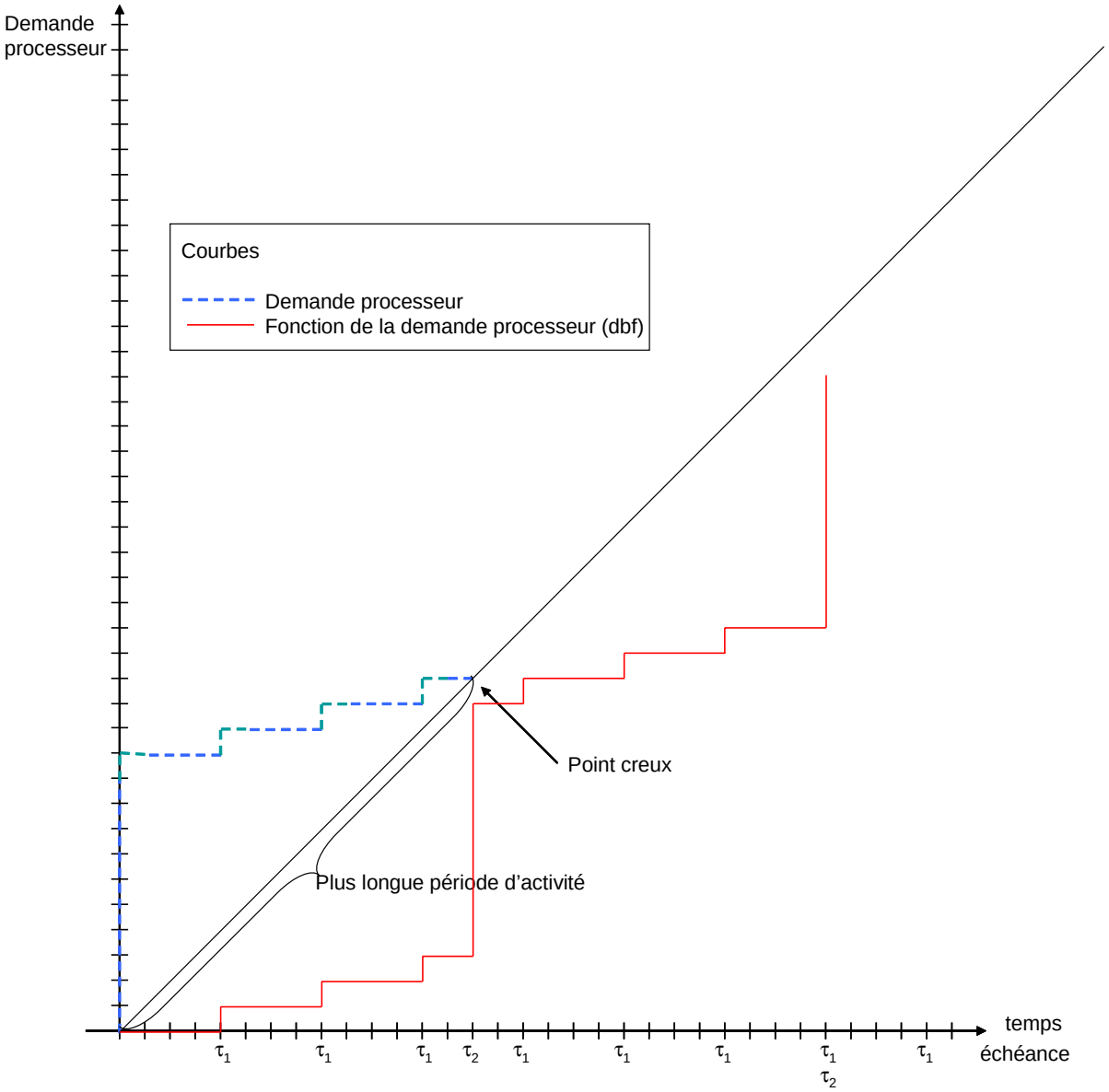

FIGURE 1.4 – Fonction de la demande processeur (dbf).

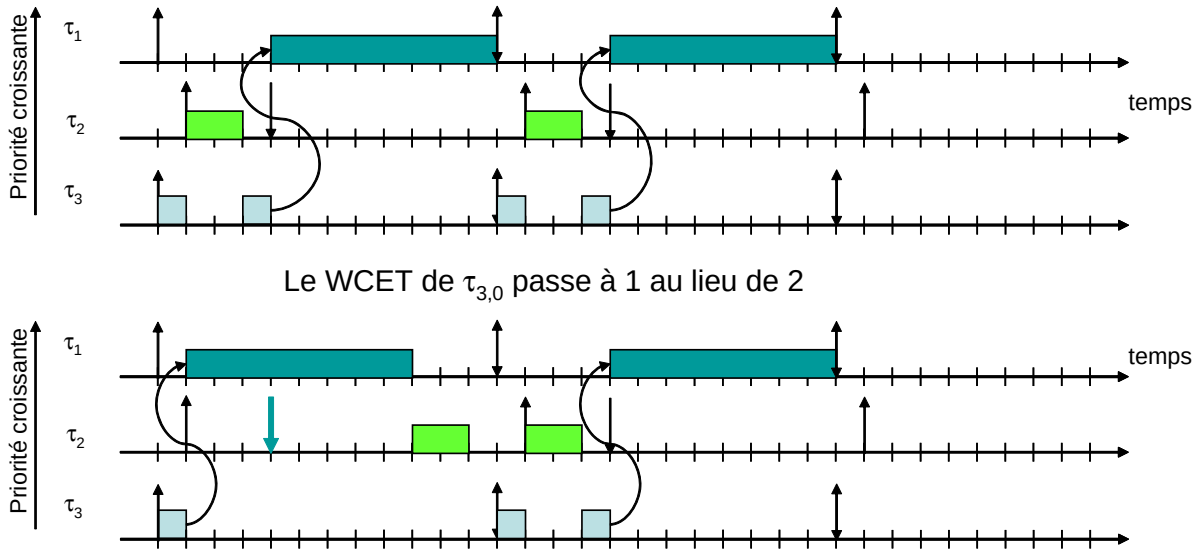

FIGURE 1.5 – Anomalie d'ordonnancement en présence de précédences.

Simulation

Une simulation consiste à construire la séquence d'ordonnancement comme elle serait construite par l'algorithme d'ordonnancement, ceci jusqu'à atteindre le régime permanent (cyclicité des ordonnancements) ou bien pour certains algorithmes spécifiques sur l'intervalle de validation (durée inférieure à la durée nécessaire pour atteindre un cycle). La question de la durée de simulation est encore ouverte dans un certain nombre de contextes. Je développe les problèmes encore ouverts et les problèmes fermés dans la partie contributions de ce mémoire.

Une simulation peut être utilisée soit pour la validation, notamment dans le cas où les tâches sont concrètes non simultanées, soit pour des études de performance pire cas. Cependant, la présence de facteurs pratiques peut rendre la simulation non représentative dans le cas où il y a des anomalies d'ordonnancement. Une anomalie a lieu lorsque la simulation conclue au respect de toutes les échéances mais que diminuer des WCET ou bien augmenter une période conduit à une violation d'échéance.

La figure 1.5 montre un exemple d'anomalie d'ordonnancement en présence de contraintes de précédence incohérentes avec les priorités : si on a $\tau_i \prec \tau_j$ et que la priorité de τ_j est supérieure à la priorité de τ_i, on peut avoir une anomalie. On peut aussi trouver des anomalies en présence de ressources critiques, de tâche qui se suspendent, et même dans le cas multiprocesseur dans le cas où les périodes ne sont pas strictes (voir figure 1.6).

Méthodes hors-ligne

Les méthodes hors-ligne reposent soit sur des approches de complexité exponentielle, de type séparation & évaluation à la Xu& Parnas [XP93, Xu93], algorithme de Shepard et Gagne [SG91] qui n'est pas optimal [AS97], approche co-design (matériel-logiciel) de [Cav97], soit sur des approches énumératives [CGGC00, Lar04], et peuvent utiliser des approches méta-heuristiques afin d'éviter l'explosion combinatoire [DS95, LGD05]. Le but est de construire une séquence d'ordonnancement valide, qui sera ensuite jouée par

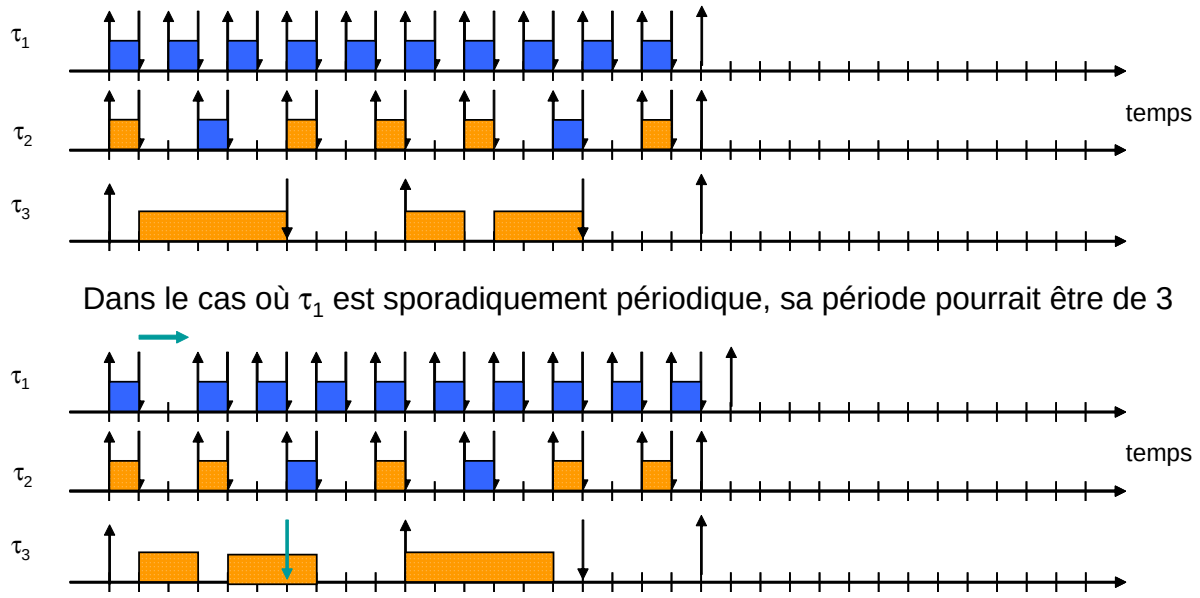

FIGURE 1.6 – Anomalie d'ordonnancement en multiprocesseur en présence de tâches sporadiquement périodiques.

un séquenceur. Cette séquence doit être générée sur la durée de simulation. Cette problématique rejoint le problème de cyclicité. Ainsi, à ma connaissance, le laboratoire est le seul, en se basant sur notre résultat de cyclicité générale, à proposer des méthodes hors-ligne prenant des tâches concrètes non simultanées en considération. En effet, les autres approches se basent sur les instances des tâches, et de ce fait considèrent un nombre d'instances H/T_i correspondant à la tâche τ_i. Cela est un facteur très limitant pour les autres approches, car une approche hors-ligne sera considérée par un concepteur lorsqu'il a un système dirigé par le temps, et donc des tâches concrètes. Dans ce cas, il est fréquent d'utiliser une technique de décalage des réveils afin d'éviter l'instant critique.

L'ordonnanceur est, pour les approches hors-ligne, complètement clairvoyant. Il n'existe pas à ma connaissance d'approches prenant en considération des tâches non-concrètes. Certains facteurs pratiques sont simples à prendre en compte (ressources critiques, précédence, non-préemptibilité) mais les facteurs qui aboutissent à une date de réveil dynamique d'une tâche (gigue d'activation, suspension) n'ont pas été explorés, le problème étant dans ce cas assez proche du problème des tâches non-concrètes : on ne peut pas être clairvoyant dans ce cas.

La partie contribution expose les travaux que nous avons menés et les résultats que nous avons obtenus dans les approches hors-ligne et en-ligne, puis leur application dans le positionnement de la validation temporelle dans le cycle de développement logiciel.

Deuxième partie

Contributions

Chapitre 2

Ordonnancement hors-ligne

Ce chapitre traite de nos contributions dans le domaine de l'ordonnancement hors-ligne. La première section présente brièvement mes résultats de thèse en ce qui concerne l'extraction de séquences optimales, la seconde section présente une méthode méta-heuristique pour l'ordonnancement monoprocesseur, la troisième partie aborde nos contributions au problème général de la cyclicité, et nous terminons le chapitre par un bilan et des perspectives de recherche.

2.1 Ordonnancement exhaustif optimal

2.1.1 Modélisation

Ma thèse de doctorat a eu lieu au LISI d'octobre 1996 à décembre 1999. Elle portait sur l'ordonnancement hors-ligne exhaustif de systèmes temps réel en environnement monoprocesseur et multiprocesseur. Elle a eu lieu sous la direction d'Annie Geniet et de Francis Cottet. L'idée de base est de modéliser un système à l'aide d'un réseau de Petri fonctionnant à vitesse maximale (il a ainsi la puissance d'une machine de Turing [Sta90]) puis d'en extraire le graphe des marquages correspondant aux comportements temporellement valides du système.

Nous partons d'un langage simple de description de tâches temps réel : date initiale de réveil, périodicité, échéance relative, et d'une description de sa structure en blocs de durée, et primitives de communication et de synchronisation. Ceci pourrait être une abstraction extraite de la conception. Notons qu'il faut cependant connaître la durée des traitements, ce qui implique un positionnement de notre démarche plutôt en fin de cycle de développement. Les contextes dans lesquels nous avons travaillé sont $|r_i, C_i, D_i \leq T_i, T_i|res, \prec$ et $multi|0, C_i, D_i \leq T_i, T_i|res, \prec, premptnopreempt$. Un système ainsi décrit est alors modélisé par un RdP (voir figure 2.1), utilisant les constructions standard (synchronisation, exclusion mutuelle), combinées avec un système d'horlogerie. Le temps est implicitement donné par le comportement du RdP, puisqu'il s'agit, étant donné que le RdP fonctionne à vitesse maximale, de forcer tout le système d'horlogerie à fonctionner (le temps est ainsi décompté à chaque tir d'un ensemble de transitions, car la transition RTC produisant le temps tire) parallèlement au traitement d'une unité de temps d'une (cas monoprocesseur) ou plusieurs tâches (cas multiprocesseur). La liberté de choisir à un instant donné d'ordonnancer une tâche prête ou une autre réside dans l'indéterminisme du RdP. Ainsi,

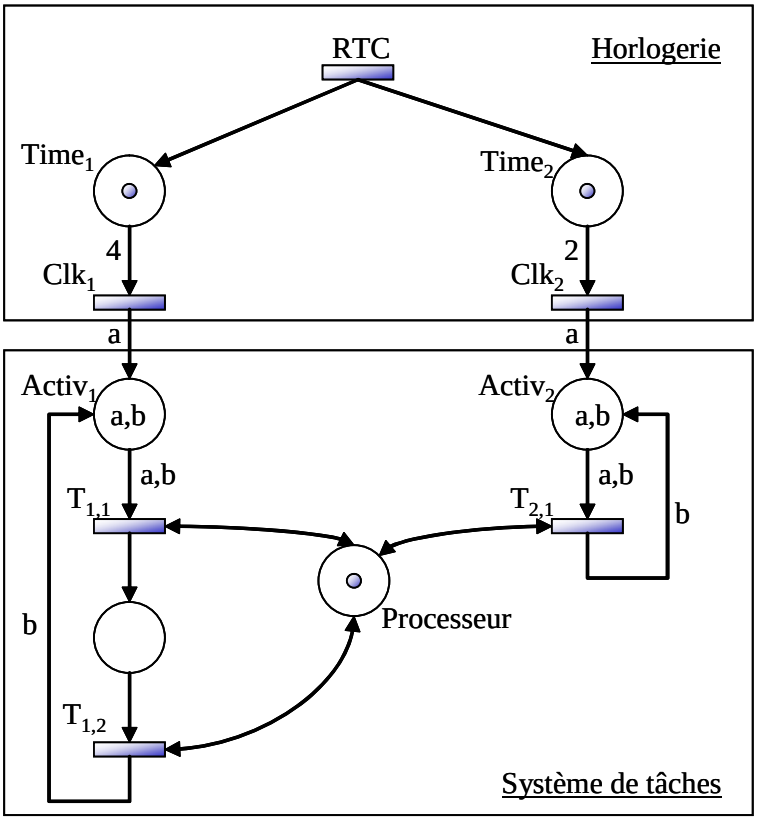

FIGURE 2.1 – Modèle de 2 tâches indépendantes $\tau_1 =< r_1 = 0, C_1 = 2, D_1, T_1 = 4 >$ et $\tau_2 =< r_2 = 0, C_2 = 1, D_2, T_2 = 2 >$ en monoprocesseur.

chaque unité de temps de traitement de tâche étant modélisée par une transition nécessitant pour être franchissable un jeton "processeur", les unités de temps des tâches sont en exclusion mutuelle (utilisation de la ressource processeur), et la construction du graphe des marquages donne naturellement l'ensemble des ordres possible d'exécution des tâches du système. On peut voir sur la figure 2.1 que le RdP utilisé est coloré : la couleur a signifie "activée" et la couleur b signifie "instance précédente terminée". Les contraintes de temps (échéances) sont prises en compte par des contraintes de marquages terminaux (par exemple, le jeton de la place marquant la terminaison d'une tâche doit être présent lorsque son horlogerie contient un nombre de jetons égal à son échéance relative). Le graphe des marquages terminaux donne alors l'ensemble des séquences valides du système modélisé. Ce graphe a une forme de diamant, c'est-à-dire que c'est un graphe sans cycle orienté, avec une source et une destination (qui en fait est identique à la source), tel que le nombre d'arc entre la source et la destination est identique quel que soit le chemin choisi. Cela correspond au fait que pour tout ordonnancement valide, chaque tâche a été exécutée le même nombre de fois. Ainsi pour un système de tâches simultanées à échéances contraintes, on trouve $C_i \times H/T_i$ arc étiqueté par τ_i sur chaque chemin, car toute séquence d'ordonnancement valide est périodique de période H dans ce cas, et l'état initial est identique à l'état final : toutes les tâches sont dans l'état "prêt" à l'instant H et à l'instant 0, et aucune instance n'est encore en cours d'exécution, sinon son échéance aurait été manquée.

Nous avons montré qu'afin de trouver des séquences dans notre contexte (utilisation

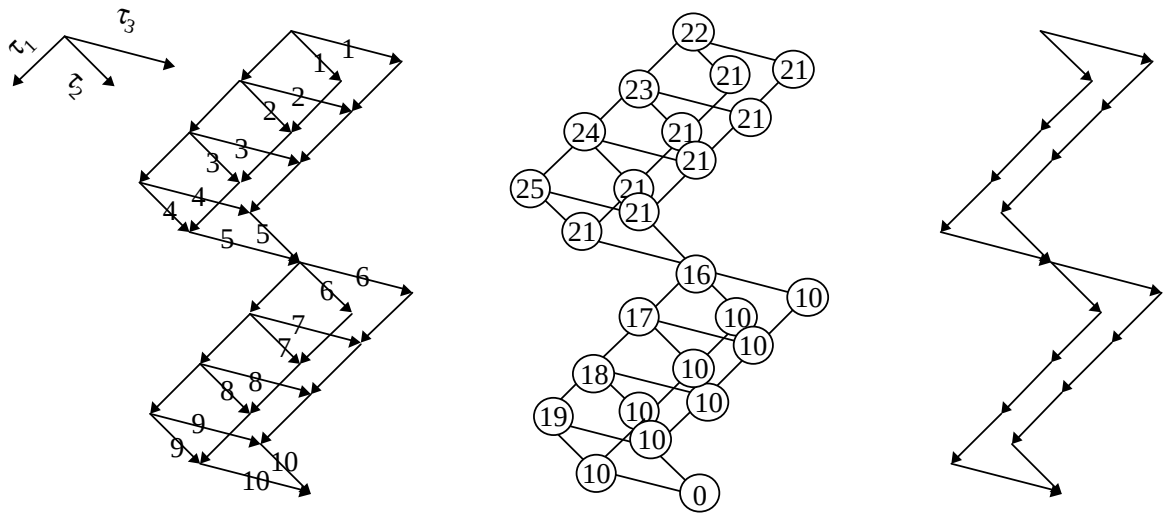

FIGURE 2.2 – Graphe d'ordonnancement avec arcs pondérés, puis nœuds pondérés, et sous-graphe extrait.

de ressources, de parties non préemptibles et de précédences), il nous fallait introduire une tâche oisive, permettant d'insérer des temps creux. En effet, sans cela, nous n'obtiendrions que des séquences conservatives. Dans le cas de tâches simultanées, il est possible de connaître le nombre de temps creux présents dans une hyperpériode : $C_0 = H(1-U)$.

2.1.2 Génération et extraction de séquences optimales

Lorsque le graphe des marquages est construit, il est simple d'en extraire des séquences d'ordonnancement optimales au vu d'un ensemble de critères. Par exemple, si l'on souhaite extraire le sous-graphe qui correspond aux séquences d'ordonnancement minimisant le temps de réponse moyen d'une tâche donnée, il suffit de valuer les arcs qui correspondent à la terminaison de la tâche dont on cherche à minimiser le temps de réponse par le temps de réponse correspondant. L'extraction du sous-graphe minimisant le temps de réponse moyen se fait alors en effectuant un parcours topologique inversé. Cela se fait en une passe en partant du noeud le plus bas du graphe, et en remontant niveau par niveau : seuls les noeuds faisant partie de chemins de coût minimal sont conservés. La figure 2.2 représente le graphe d'ordonnancement obtenu pour un système de 3 tâches $\tau_1 =< r_1 = 0, C_1 = 3, D_1 = 5, T_1 = 5 >$, $\tau_2 =< r_2 = 0, C_2 = 1, D_2 = 5, T_2 = 5 >$ et $\tau_3 =< r_3 = 0, C_3 = 2, D_3 = 10, T_3 = 10 >$. Sa profondeur est $H = PPCM(5, 5, 10) = 10$. Le coût des arcs est choisi ici pour exécuter au plus tôt les tâches τ_2 et τ_3. La mise en œuvre est simple, car le graphe est un diamant : il suffit de pondérer chaque arc par sa hauteur s'il est étiqueté par τ_2 ou τ_3. Les chemins de coût minimal sont donnés par une pondération des nœuds par un parcours topologique en partant du bas : un nœud est étiqueté par le chemin de coût minimal entre lui et la destination. Le sous-graphe obtenu en sélectionnant à chaque hauteur les nœuds de coût minimal est le sous-graphe contenant les séquences optimales pour le critère "exécution au plus tôt des tâches τ_2 et τ_3".

Nous avons identifié et classifié des critères d'optimisation en deux catégories :

- les critères graphiques, permettant d'extraire un sous-graphe suivant la méthode exposée :
  - exécution au plus tôt, appelée aussi importance (pourrait s'appeler priorité mais à cause des exclusions mutuelles, la différence de l'importance est qu'on peut être amené à ordonnancer d'autres tâches avant la tâche "importante" pour éviter de violer une échéance),
  - minimisation du temps de réponse moyen ou maximal, favorise les tâches longues de l'ensemble choisi,
  - minimisation du taux de réaction (TR_i/D_i) moyen ou maximal, favorise les tâches de petit délai critique de l'ensemble choisi,
  - minimisation du retard ($TR_i - D_i$) moyen ou maximal (remarque : étant donné que nous travaillons sur le graphe de séquences valides, ce critères est toujours ≤ 0), ne favorise aucun type de tâche dans l'ensemble choisi,
- les critères arborescents ne peuvent pas s'exprimer en terme de graphe, car un même nœud peut faire partie de "bons" et de "mauvais" chemins au regard du critère. L'implémentation de la minimisation de tels critères nécessite un passage du graphe des ordonnancements à l'arbre des ordonnancements, ce qui n'est pas applicable sur des études de cas, même petites :
  - gigue temporelle : exprime la différence entre les temps de réponse des instances successives. Cette propriété est particulièrement intéressante pour certaines tâches faisant de l'échantillonnage, calculant des intégrales ou bien des dérivées entre des points successifs. Il n'y a pas de moyen efficace d'extraire de telles séquences, par conséquent j'ai été amené à collaborer avec Laurent David sur des techniques de diminution de la gigue temporelle en utilisant le décalage des réveils.
  - répartition des temps creux : utile pour le traitement en tâche de fond de tâches apériodiques, ce critère, comme la gigue, ne peut pas être calculé efficacement.

La construction du graphe des marquages déclenchant de nombreux retours arrière, il nous a fallu détecter des conditions de coupe au plus tôt lors de la construction du graphe. Ceci a été fait en utilisant des conditions nécessaires (CN) d'ordonnançabilité avant ajout de chaque nœud, afin de vérifier la viabilité de celui-ci.

Enfin la taille de ce graphe étant exponentielle en fonction du nombre de tâches n (par exemple, dans le cas simultané, le graphe s'inscrit dans un hyperpavé à $n+1$ dimensions, sa longueur dans chaque dimension étant $C_i \times H/T_i$), des optimisations sont mises en place de façon à réduire ce graphe. Ainsi, nous avons mis en place des contraintes de successeurs, exprimant le fait que lorsque deux parties de tâches sont indépendantes, il n'est pas nécessaire d'étudier les comportements entrelacés. Des contraintes supplémentaires vérifiées lors de l'exécution ont aussi été implémentées afin de réduire la taille du graphe des marquages dès la construction.

Initialement les systèmes de tâches considérés étaient composés de tâches périodiques simultanées (i.e. elles démarrent toutes au même instant). La forme de diamant étant la base d'utilisation de critères de recherche de séquences optimales, il nous semblait important de la conserver même dans le cas où les tâches sont différées. Nous nous sommes heurtés à un problème ouvert dans le cas général, qui était le problème de la cyclicité des ordonnancements. Les seuls résultats connus concernaient les algorithmes à priorités fixes avec des tâches indépendantes, et ne pouvaient pas être appliqués sur notre méthode

d'ordonnancement exhaustif. De plus, même si nous avions pu généraliser ces résultats à un contexte plus large, le graphe des marquages n'avait pas eu un unique état final, et le nombre d'occurrences des tâches par chemin aurait été différent.

Nous avons donc cherché, à travers la vue particulière que nous offrait le graphe des ordonnancements valides, à comprendre le comportement périodiques des ordonnancements de tâches à départ différé. [LM80] montre que les algorithmes d'ordonnancement à priorités fixes dans le contexte $|r_i, C_i, D_i \leq T_i, T_i|$ sont périodiques de période H à partir de la date $r + H$ avec $r = max_{i=1..n}(r_i)$ la date de réveil la plus tardive des tâches. Cela signifie qu'après une montée en charge d'au plus $r + H$ le système se comporte périodiquement sur une période H. Ce résultat ne concerne pas les ressources critiques ni les précédences, et ne peut pas être utilisé dans le cadre d'une approche hors-ligne. Nous avons montré que tout ordonnancement n'injectant pas de temps creux de façon forcée possédait un dernier temps creux acyclique, qui pouvait avoir lieu avant le démarrage du système. La date de ce temps creux dépend uniquement des dates de réveil, périodes, et pire durées des tâches, et pas de l'algorithme d'ordonnancement employé. Ce dernier temps creux acyclique, dont la date d'arrivée est notée t_c, $t_c \in [-1..r + H - 1]$, marque l'entrée dans le régime permanent qui a une période H. La borne supérieure, que l'on peut obtenir pour certains systèmes, correspond au résultat précédent, mais le cas le plus fréquent correspond à une entrée immédiate dans le cycle. Ce résultat, très général pour la théorie de l'ordonnancement, donne la durée exacte de la montée en charge, avant entrée dans le régime permanent du système. Le problème de déterminer t_c de façon analytique est encore un problème ouvert.

L'utilisation de ce résultat pour notre problème a permis de généraliser au cas des tâches différées les résultats obtenus sur le graphe en forme de diamant correspondant aux systèmes de tâches simultanées. En effet, la forme de diamant est présente, et correspond en fait à un double diamant : l'état initial mène vers un seul état, quelle que soit la séquence considérée, qui correspond à l'état suivant le dernier temps creux acyclique. L'état obtenu H unités de temps plus tard est le même.

Le modèle a ensuite été élargi au contexte $multi|0, C_i, D_i \leq T_i, T_i|$: l'avantage d'utiliser un tel modèle est que le passage à l'échelle n'a nécessité que la modification du nombre de jetons contenant le nombre de processeurs. Le graphe des marquages a une forme de diamant comme dans le cas monoprocesseur, et on peut donc appliquer les mêmes techniques. Cependant, l'applicabilité sur des cas réels de la version multiprocesseur laisse peu d'espoir face à l'explosion combinatoire.

Afin de démontrer qu'une telle méthode peut être employée sur des cas réels dans le cas monoprocesseur, j'ai développé un logiciel prototype (voir figure 2.3) construisant des séquences d'ordonnancement optimales dans le cas monoprocesseur, ce logiciel s'appelle PeNSMARTS [1]. Il est composé d'un moteur de construction de graphes des marquages, et d'extraction suivant les critères proposés durant ma thèse, avec compression au vol des marquages (chaque place du modèle est bornée), d'un traceur de séquences, d'un outil de construction visualisation et animation de réseaux de Petri colorés, et d'un visualisateur de graphes des marquages. Il a été repris et étendu pour prendre en compte des tâches modélisant aussi les blocs conditionnels par Stéphane Pailler lors de sa thèse de doctorat. PeNSMARTS a subi plusieurs évolutions, notamment lors de stages de Licence

1. Petri Net Scheduling Modeling and Analysis of Real-Time Systems

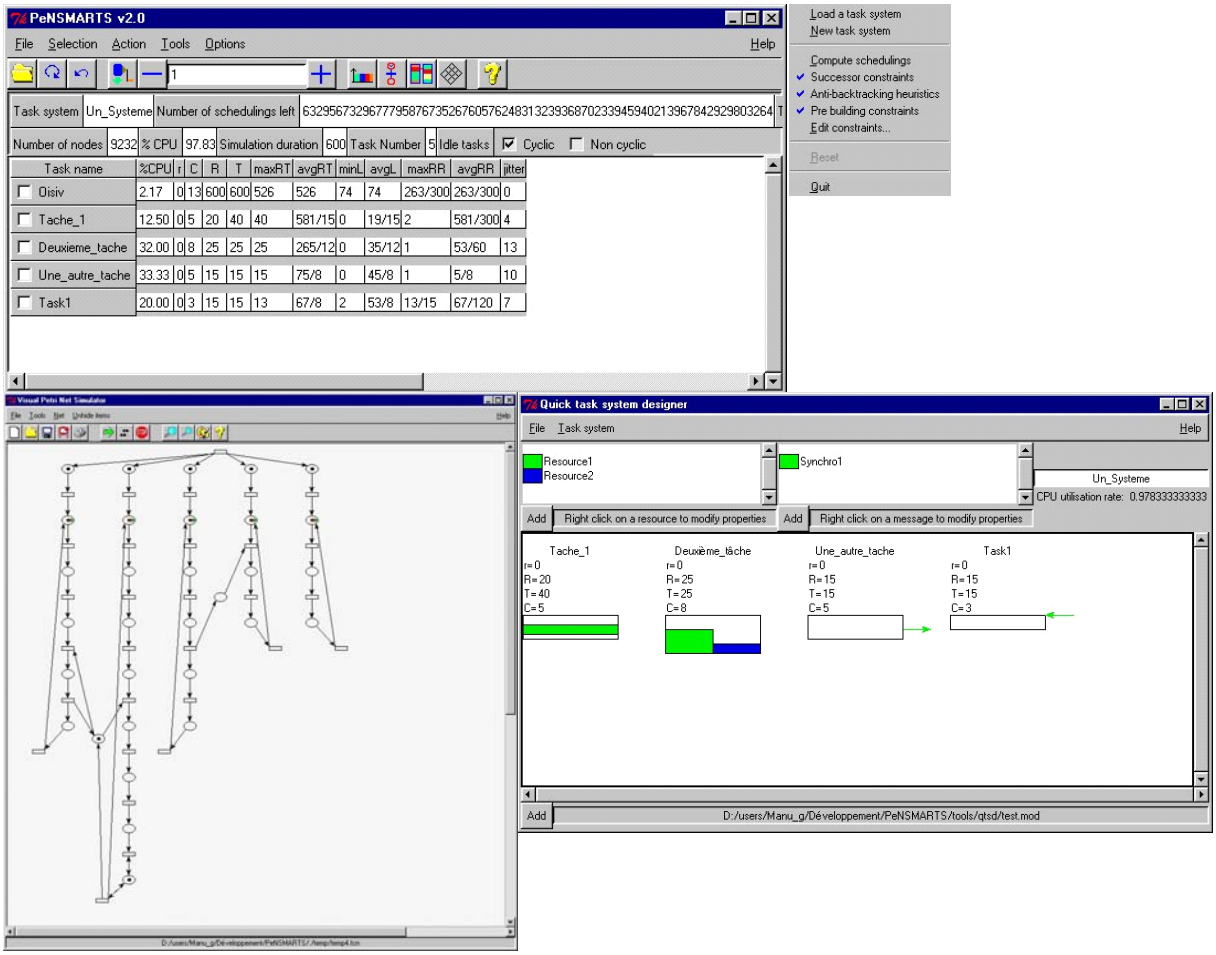

FIGURE 2.3 – Capture d'écran de PeNSMARTS, de Color Petri Net Simulator, et Quick Task Systems Designer.

ou de Master pour y intégrer la prise en compte de réseaux de Petri temporels, et plus récemment y intégrer l'aspect multiprocesseur.

2.1.3 Collaborations et publications

Ces recherches ont été principalement effectuées pendant ma thèse de doctorat, sous la direction d'Annie Geniet et de Francis Cottet, pour laquelle j'ai obtenu le 2nd prix de thèse de la région Poitou-Charentes en 2000. J'ai pu collaborer dans le cadre d'une méthode réunissant approche hors-ligne et langage synchrone avec Yamine Aït-Ameur et Frédéric Boniol. Les publications issues de ces travaux sont [GCG02, CGGC00, Gro99, AABD+04, GCG01, GCG00c, GCGC99, GCGC98b, GCG00b].

2.1.4 Points forts de la démarche et perspectives

La méthode que nous avons proposée est à ma connaissance la seule méthode hors-ligne prenant en compte des tâches différées, ceci grâce à l'étude menée sur le problème de cyclicité dans le cas général. Cette augmentation du contexte, par rapport aux autres

méthodes hors-ligne, est primordiale, car sur les applications réelles susceptibles d'utiliser un ordonnancement hors-ligne, toutes les tâches sont concrètes, par conséquent, le décalage des dates de réveil en vue d'éviter l'instant critique apporte un gain important en ordonnançabilité. De plus, nous avons outillé cette méthode afin de montrer jusqu'où on pouvait résister à l'explosion combinatoire avec ce type de méthode.

D'autres travaux ont été menés par la suite sur l'ordonnancement exhaustif au laboratoire, notamment dans la thèse de Gaëlle Largeteau, sous la direction de Dominique Geniet et Francis Cottet. L'idée de base est de modéliser chaque tâche par un automate fini, puis d'obtenir l'ensemble des séquences valides par produit synchronisé. Les résultats obtenus sont identiques au graphe des marquage terminaux, mais le fait d'utiliser des outils génériques pour gérer les synchronisation de gros automates rend la mise en œuvre très gourmande en temps et en espace par rapport à PeNSMARTS qui possède des optimisations ad hoc. Par la suite, ils ont étudié les graphes d'ordonnancement en forme de diamant avec une approche informatique graphique, ne manipulant plus le graphe des ordonnancement en tant que graphe, mais en tant que figure géométrique. Différents travaux sont menés au laboratoire sous la direction d'Annie Geniet pour prendre en compte différents paramètres dans les applications ordonnancées de façon exhaustive, comme les tâches à durées conditionnelles, ou les sporadiques.

2.2 Méthode méta-heuristique

Afin de pouvoir comparer l'efficacité une approche optimale à une approche méta-heuristique, j'ai été amené à proposer et tester une méthode méta-heuristique sur le problème de l'ordonnancement hors-ligne. Cette méthode se base sur la méthode qui sert de base à la plupart des approches orientées séparation & évaluation pour la résolution du problème d'ordonnancement hors-ligne optimal.

2.2.1 Méthode optimale de base

Xu et Parnas ont proposé une méthode d'ordonnancement optimal hors-ligne, basée sur une approche de type séparation & évaluation. Leur technique ordonnance de façon optimale des systèmes de tâches simultanées pouvant se synchroniser et communiquer. Le contexte utilisé dans [XP93] est $|0, C_i, D_i \leq T_i, T_i|res, \prec$, et $multi|0, C_i, D_i \leq T_i, T_i|res, \prec$ dans [Xu93]. Cependant leur approche ne s'étend pas trivialement au cas non simultané, et ne cherche pas à optimiser d'autre critère que le respect d'échéance. Leur méthode part d'une séquence d'ordonnancement EDF car la séquence produite est optimale dans le contexte $|0, C_i, D_i, T_i|$. Etant donné qu'on est en présence de ressources critiques, EDF n'est pas optimal, et on cherche alors à améliorer la séquence à chaque étape de la construction, soit jusqu'à ce qu'on s'aperçoive que les choix faits jusqu'au noeud en cours ne peuvent pas mener à un ordonnancement valide, dans ce cas, on remonte et on coupe, soit jusqu'à ce que l'on trouve un ordonnancement valide. La méthode de Xu & Parnas, bien qu'optimale uniquement au regard du respect des échéances, est nettement plus rapide que la nôtre, cependant elle fonctionne dans le contexte $m|r_i = 0, C_i, D_i \leq T_i, T_i|prec, res$ pour [Xu93] et $|r_i = 0, C_i, D_i \leq T_i, T_i|prec, res$ pour [XP93]. Nous nous sommes inspirés du cas monoprocesseur pour y adapter une technique méta-heuristique.

D'un système de tâches périodiques simultanées, Xu & Parnas décomposent chaque tâche τ_i en H/T_i instances, qui sont appelées les *jobs* j_i. Chaque *job* est vu comme une suite de segments $j_i = (s_{i,1}, ..., s_{i,n_i})$ de façon à n'avoir des précédences (envois/réceptions de messages, etc.) qu'en début ou fin de segment (i.e. elles pourront être exprimées sous la forme $s_{i,j} \prec s_{k,l}$), et des exclusions mutuelles portant sur des suites de segments entiers. Cette technique de découpage, dite "en forme normale" est classique dans les techniques d'ordonnancement en-ligne prenant en compte des contraintes de précédence. Implicitement deux segments successifs appartenant à la même tâche ont une contrainte de précédence. Chaque segment se voit muni d'une date de réveil correspondant à la date de réveil du *job* correspondant pour le premier segment d'un *job*, et correspondant à $r_{i,j} + C_{i,j}$ pour le *job* $s_{i,j+1}$. Les dates de réveil sont ajustées pendant le fonctionnement de l'algorithme de sorte à ce que la date de réveil d'un segment corresponde au maximum pour chaque prédécesseur à sa date de réveil plus son WCET (donc $r_{i,j}$ correspond à la date d'activation au plus tôt de $s_{i,j}$). Etant donné que les tâches sont transformées en *jobs*, les délais critiques sont transformés en échéances, et appliquées au segments de façon symétrique à la méthode employée pour les dates de réveil pour obtenir une échéance $d_{i,j}$ pour chaque segment. Les exclusions mutuelles sont exprimées à l'aide de l'opérateur non réflexif *exclude*, ainsi une exclusion mutuelle entre deux suites de segments s'exprime $(s_{i,j}, s_{i,j+1}, ...s_{i,j+m}) \oplus (s_{k,l}, s_{k,l+1}, ...s_{k,l+p})$ et $(s_{k,l}, s_{k,l+1}, ...s_{k,l+p}) \oplus (s_{i,j}, s_{i,j+1}, ...s_{i,j+m})$.

La séquence initiale est obtenue en appliquant l'algorithme EDF tout en respectant les contraintes de date de réveil, précédence et exclusion. L'idée générale est de repérer le segment le plus tardif s_{late} et de l'ordonnancer plus tôt en respectant les contraintes. On identifie pour cela 2 ensembles de segments dans la période d'activité contenant le segment que l'on veut ordonnancer plus tôt. Le premier contient des segments moins urgents que s_{late}, qui, s'ils sont placés avant, le sont parce-qu'ils détenaient une ressource voulue par s_{late}, et tels qu'on n'a pas posé de contrainte les obligeant à préempter s_{late}. Pour chaque section critique concernée par cet ensemble, on crée un fils de l'ordonnancement courant dans lequel on introduit des contraintes de précédence forçant s_{late} à précéder chaque segment de la section critique. Le second ensemble de segments est constitué de segments plus urgents que s_{late} mais qui ont pu être retardés par des segments moins urgents les excluant. On crée alors un fils pour chaque segment concerné de ce second ensemble où on oblige avec des contraintes de précédence ce segment à précéder les segments l'excluant, et on oblige chaque segment s'exécutant entre ce segment et s_{late} a être plus prioritaire que le segment en question en introduisant une contraint *preempts*. Les nœuds fils sont construits en utilisant EDF en respectant les contraintes (date de réveil, précédences, exclusion, et *preempts* qui rend un segment plus prioritaire qu'un autre au détriment des échéances). L'arbre de recherche est construit en profondeur d'abord, en sélectionnant le nœud ouvert le plus prometteur.

Cet algorithme est optimal, nous pouvons remarquer qu'il n'est pas conservatif, car il peut injecter des temps creux alors qu'il y a des tâches qui auraient été prêtes au vu des contraintes originelles. Ces temps creux sont injectés dans le cas où des contraintes "precède" sont ajoutées artificiellement pour éviter qu'une section critique ne commence avant un segment qui manquerait dans ce cas son échéance.

2.2.2 Introduction aux méthodes basées sur les colonies de fourmis

Ce qui est intéressant lorsqu'on souhaite appliquer un algorithme méta-heuristique en se basant sur une méthode à la Xu & Parnas est que l'on peut évaluer la qualité d'une solution, et qu'on est capable d'améliorer localement une solution (en améliorant la latence du segment de latence maximale). Nous avons donc mis en place une méthode méta-heuristique utilisant un algorithme basé sur les colonies de fourmis. Le but était d'évaluer cette méthode relativement récente sur un problème d'ordonnancement temps réel.

Les algorithmes d'optimisation basés sur les colonies de fourmis ont été proposés pour la première fois par Marco Dorigo en 1991 comme une approche multi-agents pour la résolution de problèmes d'optimisation combinatoire tel que le problème du voyageur de commerce [DCG98]. Nous avons donc, comme nombre de chercheurs, travaillé pour étendre le domaine d'application de ces nouveaux algorithmes à d'autres problèmes d'optimisation combinatoire.

Les algorithmes d'optimisation basés sur les colonies de fourmis sont inspirés par l'observation du comportement des fourmis réelles. Les fourmis sont des insectes vivants dans des colonies et le comportement d'une colonie est beaucoup plus intéressant que celui d'une seule fourmi, et en particulier le fait d'observer comment les fourmis peuvent trouver le plus court chemin entre la fourmilière et la source de nourriture. Les fourmis, quand elles se déplacent entre la fourmilière et une source de nourriture déposent une substance chimique appelée la phéromone, et forment ainsi une piste de phéromone sur ce chemin. Les fourmis peuvent sentir de la phéromone et elles choisissent un chemin entre plusieurs chemins, avec une probabilité équivalente à la quantité de phéromone présente sur l'un des chemins. La piste de la phéromone permet aux fourmis de revenir en arrière vers la fourmilière et aux autres fourmis de trouver la source de nourriture. Ce comportement explique comment les fourmis trouvent le plus court chemin, après l'apparition d'un obstacle qui interrompt le chemin initial : les fourmis qui sont en tête ne peuvent pas continuer à suivre la piste de phéromone. Donc elles ont le choix de prendre la branche d'un côté ou de l'autre de l'obstacle avec une équi-probabilité. La moitié des fourmis choisissent la branche supérieure et l'autre moitié la branche inférieure. Les fourmis qui ont pris le chemin le plus court vont plus rapidement reconstituer la piste de la phéromone interrompue, par rapport à celles qui ont pris le chemin le plus long. Ainsi le plus court chemin va avoir beaucoup plus de phéromone dans le temps du fait du passage d'un plus grand nombre de fourmis. Au bout d'un certain temps, toutes les fourmis vont choisir le nouveau plus court chemin.

Un problème d'optimisation combinatoire peut être représenté par un ensemble de composants, un ensemble de connexions entre composants, un ensemble de coûts associés aux connexions, et un ensemble de contraintes fonction du temps. Un état s du problème est une séquence d'éléments. Deux états sont voisins si un état peut être atteint à partir d'un autre état à l'aide d'une connexion. Notons $N(s)$ l'ensemble des voisins de s. Un état du problème est une solution au problème si la séquence satisfait les contraintes, et a un coût calculé à partir des coûts associés aux connexions. Les algorithmes basés sur les colonies de fourmis, dits Ant Colony Optimization (ACO), reprennent le comportement naturel des fourmis pour résoudre des problèmes d'optimisation à la différence près que

les fourmis artificielles ont une mémoire interne, peuvent mettre à jour la phéromone après construction d'une solution locale, et peuvent être aidées par des compétences supplémentaires telles l'anticipation, la recherche locale ou les retours arrières [DCG99]. L'avantage des approches ACO sur des approches de type recuit simulé ou algorithmes génétiques est qu'elles s'adaptent à des modifications des composants et connexions entre composants du problème.

L'algorithme ACO de base est Ant system (AS) [Dor91, DCG98] que nous décrivons succinctement. Dans AS, à l'instant t, une fourmi située sur un composant i choisit le composant j avec la probabilité :

$$P_{ij}(t) = \frac{(T_{ij}(t))^{\alpha} (\eta_{ij}(t))^{\beta}}{\sum_{\forall \text{ composant atteignable } k} (T_{ik}(t))^{\alpha} (\eta_{ik}(t))^{\beta}}$$

où T_{ij} est la phéromone entre i et j et η_{ij} est l'information heuristique (par exemple, inversement proportionnelle au coût associé au passage de i à j). Les coefficients α et β sont choisis de façon à donner plus ou moins de poids à la phéromone ou à l'information heuristique. A chaque construction de solution par un nombre choisi de fourmis, une unité de temps s'écoule, et la phéromone est mise à jour sur chaque connexion en fonction d'un facteur d'évaporation et de l'inverse du coût des solutions parcourues par les fourmis utilisant cette connexion.

L'algorithme AS nécessite un grand nombre d'itérations pour trouver de bonnes solutions sur de grandes instances de problèmes. Par conséquent de nombreuses variantes ont été proposées :

- AS avec stratégie élitiste, où la meilleure solution trouvée depuis le début dépose de la phéromone supplémentaire à chaque itération [BHS99].
- AS avec stratégie élitiste *ASrank* [BHS99], qui correspond à une stratégie élitiste avec en plus une augmentation de phéromone de la part d'un certain nombre des meilleures fourmis de l'itération courante.
- Max-Min Ant System (MMAS) modifie AS dans le sens que seule la meilleure fourmi de l'itération courante est autorisée à ajouter de la phéromone, la phéromone est limitée entre une valeur *min* et une valeur *max* afin d'éviter la stagnation et les valeurs de phéromone sont initialisées à la valeur *max*.
- Ant Colony System (ACS) [DG97] utilise un paramètre permettant de choisir entre exploration de nouvelles solutions et exploitation de la phéromone, a une stratégie élitiste, et fait diminuer localement la phéromone pendant la construction d'une solution par une fourmi sur un composant exploré. Ainsi, la recherche est relativement guidée par la phéromone, mais la diminution locale de phéromone diminue les risques de voir beaucoup de fourmis explorer le même chemin. C'est l'algorithme que nous avons choisi d'employer pour une première étude sur le problème d'ordonnancement temps réel car il offre de bonnes possibilités de convergence, confirmées depuis notre étude dans [NJ09].
- Multiple Ant Colony Optimization (MACO) [SS02] consiste à utiliser plusieurs colonies de fourmis utilisant leur propre type de phéromone. Plusieurs colonies de fourmis peuvent alors explorer le problème d'optimisation en parallèle, ce qui rend cette technique hautement parallélisable.

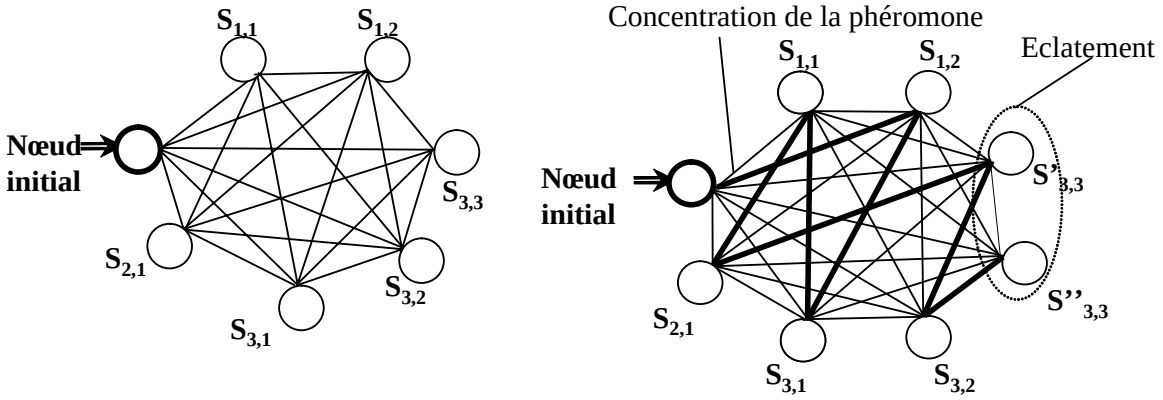

FIGURE 2.4 – Modification du graphe de recherche par ACSRTS.

2.2.3 Notre algorithme ACO

Nous avons adapté ACS à la création de séquences d'ordonnancement valides dans le contexte $|0, C_i, D_i \leq T_i, T_i|res, \prec$. L'algorithme obtenu s'appelle Ant Colony System for Real-Time Scheduling (ACSRTS). Nous partons d'un système de tâches concrètes simultanées, avec des précédences et des exclusions mutuelles, et nous découpons chaque tâche τ_i en H/T_i *jobs*, eux mêmes découpés en segments entre lesquels nous représentons les contraintes de précédence et les exclusions comme cela est fait dans [XP93]. Le problème initial est alors vu comme la recherche d'un chemin entre les segments respectant les contraintes, et minimisant le retard maximal. Cependant, l'approche de [XP93] montre qu'il peut être nécessaire, pendant l'exploration, de découper des segments. Par conséquent, notre graphe de recherche est dynamique.

Le principe de cheminement d'une fourmi le long d'une solution est le suivant :

1. initialement, la date de l'ordonnancement est $t = 0$, la fourmi se trouve sur le nœud initial ;
2. la fourmi considère les segments éligibles (prédécesseurs déjà traités, pas d'exclusion avec un segment en cours, date de réveil $\leq t + c$ avec c durée de la plus longue section critique) et utilise la règle de transition pseudo-aléatoire ACS (l'information heuristique se base sur l'algorithme EDF) pour choisir le segment à exécuter, elle met alors à jour la phéromone localement (afin de diminuer les chances pour les autres fourmis de choisir le même segment) ; la valeur c permet à la fourmi d'avoir une chance de choisir de ne rien faire afin de ne pas commencer une section critique qui pourrait ensuite retarder un segment pas encore prêt à l'instant t ;
3. la séquence en cours de construction se voit ajouter tout ou partie du segment choisi : si le porchain réveil d'un segment éligible a lieu après la fin du segment, c'est tout le segment qui est ajouté, sinon le segment est décomposé en 2 (voir figure 2.4) afin de laisser à la fourmi une chance de choisir de préempter le segment en cours par le segment nouvellement éligible.
4. on continue ainsi jusqu'à arriver à parcourir tous les segments.

Lorsque les fourmis ont créé chacune une séquence d'ordonnancement, on applique la stratégie élitiste en augmentant la phéromone déposée sur les arcs qui correspondraient

aux fils explorés dans l'approche de Xu& Parnas permettant d'améliorer la meilleure solution trouvée. Nous avons alors implémenté l'algorithme et avons recherché des paramètres ACS permettant une convergence rapide vers une solution valide, puis comparé ACSRTS à l'algorithme de Xu& Parnas. Les résultats sont difficilement comparables : ainsi sur des instances très contraintes, les performances temps/qualité de la solution sont très comparables, par contre sur des instances peu contraintes, l'algorithme de Xu& Parnas trouve très rapidement une solution optimale. Cependant sur une étude de cas réaliste, ACSRTS a montré des performances nettement supérieures. Une approche métaheuristique pourrait être un bon complément à une approche optimale dans le cas où l'instance de problème est grande et contrainte.

2.2.4 Collaborations et publications

Cette étude a été menée en collaboration avec le Pr. Habiba Drias, du LRIA (Laboratoire de Recherche en Intelligence Artificielle) d'Alger, via le co-encadrement du stage de magistère de Yacine Laalaoui, étudiant de l'INI (Institut National d'Informatique) d'Alger. Cela a donné lieu a une publication en 2005 [LGD05].

2.2.5 Points forts de la démarche et perspectives

A notre connaissance, cette approche a été la première adaptation d'une technique ACO à l'ordonnancement temps réel. L'une des difficultés de ce type de méthodes réside dans la façon d'exprimer le graphe de recherche. Nous avons démontré la faisabilité d'une telle approche, et exploité la spécificité de ce type de méthode en travaillant sur un graphe dynamique.

Il semble qu'ACSRTS puisse être assez simplement passé à l'échelle multiprocesseur, et les performances comparées avec les algorithmes exacts devraient être améliorées.

Toutes les approches basées sur l'algorithme de Xu& Parnas sont très limitées pour les applications réelles à cause de la contrainte $r_i = 0$. Il faudrait donc utiliser notre résultat de cyclicité des ordonnancements afin de les adapter au cas où certaines tâches sont différées.

2.3 Cyclicité des ordonnancements

Le problème de la cyclicité est central dans de nombreux problèmes d'ordonnancement :

- ordonnancement hors-ligne,
- validation par simulation d'algorithmes en-ligne, dans le cas où les tâches sont non simultanées.

Le premier résultat connu concerne le contexte $|r_i, C_i, D_i \leq T_i, T_i|$ et a été montré par Leung et Merrill.

Théorème 7 *[LM80] Dans le contexte $|r_i, C_i, D_i \leq T_i, T_i|$, l'algorithme DM se comporte périodiquement avec une période $H = PPCM_{i=1..n}(T_i)$ au plus après $r + H$ unités de temps avec $r = max_{i=1..n}(r_i)$. Par conséquent, la durée de simulation est $[0..r + 2H[$.*

2.3.1 Résultat principal

Nous avons adopté une démarche totalement différente, puisque fonctionnant avec tout algorithme d'ordonnancement y compris non conservatif (insérant des temps creux) à condition qu'il ne force pas d'injection de plus de $H(1-U)$ temps creux toutes les H unités de temps. Le résultat que nous avons démontré est le suivant :

Théorème 8 *[CGG04] Dans le contexte $|r_i, C_i, D_i \leq T_i, T_i|res, \prec$, toute séquence d'ordonnancement valide générée par un algorithme déterministe ne forçant pas l'injection de temps creux au-delà de $H(1-U)$ toutes les H unités de temps, est périodique de période H après le dernier temps creux acyclique, arrivant à la date $t_c \in [-1..r+H[$. Le régime périodique est la première fenêtre temporelle de taille H sans temps creux acyclique, et démarre exactement après la date t_c. La date t_c est indépendante de l'algorithme d'ordonnancement choisi.*

Par algorithme déterministe nous entendons un algorithme prenant la même décision d'ordonnancement face à deux états identiques du système. Le fait que l'algorithme ne force pas d'injection de temps creux au-delà de $H(1-U)$ toutes les H unités de temps s'explique de la façon suivante : un système de charge $U = \sum_{i=1..n} \frac{C_i}{T_i}$ voit les tâches utiliser exactement UH unités de temps processeur dans une fenêtre de taille H. Le reste, inutilisé par le système, se traduit par des temps creux, pendant lesquels le processeur est oisif. Par conséquent le nombre de temps creux que l'on peut trouver dans une fenêtre périodique de taille H correspond à $(1-U)H$. Si nous ne nous étions intéressés qu'aux algorithmes en-ligne, le théorème 8 aurait pu être exprimé ainsi :

Corollaire 1 *Dans le contexte $|r_i, C_i, D_i \leq T_i, T_i|res, \prec$, toute séquence d'ordonnancement valide générée par un algorithme déterministe conservatif est périodique de période H après le dernier temps creux acyclique, arrivant à la date $t_c \in [-1..r+H[$. Le régime périodique démarre exactement après la date t_c. Le régime périodique est la première fenêtre temporelle de taille H sans temps creux acyclique, et démarre exactement après la date t_c. La date t_c est indépendante de l'algorithme d'ordonnancement choisi.*

Ce corollaire exprime plus clairement le fait que tout algorithme en-ligne classique est concerné par ce résultat. Cependant, dès lors que des ressources critiques sont en jeu, les algorithmes conservatifs ne sont pas optimaux. Une approche hors-ligne telle que celle que nous avons développée doit donc autoriser l'injection de temps creux (i.e. le non conservatisme) afin d'être optimale. Cependant, il n'est pas nécessaire d'injecter plus du nombre de temps creux que le nombre de temps creux figurant dans la régime périodique pour être optimal. Par conséquent, nous avons démontré le résultat de cyclicité au plus large de façon à ce qu'il concerne les algorithmes pouvant être optimaux même dans le cas de partage de ressource. C'est le cas de l'approche implémentée par PeNSMARTS, qui peut forcer l'injection d'au plus $H(1-U)$ temps creux par hyperpériode. Notre résultat de cyclicité est donc suffisamment large pour s'appliquer à PeNSMARTS. Cependant, il ne peut pas être appliqué trivialement aux approches de séparation et évaluation type Xu & Parnas. En effet, le nombre de temps creux injectés n'est pas contrôlé dans leur approche (les temps creux peuvent être injectés dans une séquence via la création de contraintes de précédence, pouvant imposer qu'un segment non réveillé précède un segment prêt). Il n'est pas exclu cependant qu'une étude plus poussée du comportement de l'algorithme permette à cette approche de respecter le cadre du résultat de cyclicité.

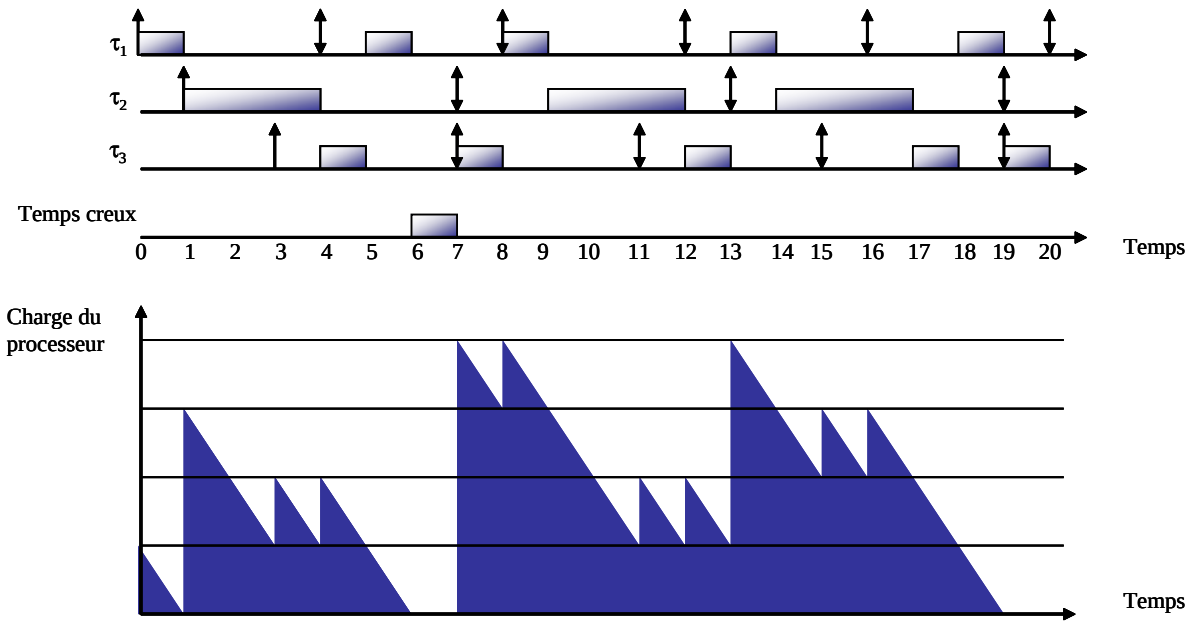

FIGURE 2.5 – Une séquence d'ordonnancement, l'occupation du processeur et la mise en évidence d'un temps creux acyclique.

Notre résultat de cyclicité est exact, alors que la borne de Leung & Merrill est une borne supérieure, de plus, nous prenons en compte les ressources critiques et les contraintes de précédences.

Le concept de temps creux acyclique est inédit à notre connaissance dans la théorie de l'ordonnancement temps réel. Afin d'illustrer ce concept, la figure 2.5 représente l'ordonnancement par l'algorithme RM d'un système de trois tâches sur un processeur. La charge du système est $U = 1$, pourtant un temps creux apparaît. Il faut remarquer qu'il apparaît dans le régime transitoire de la séquence, et marque l'entrée dans le régime périodique de période $H = 12$, comme cela est démontré par le théorème 8. Le diagramme d'occupation du processeur de la figure 2.5 illustre le fait que l'algorithme d'ordonnancement choisi n'influe pas sur la cyclicité. Celle-ci n'est influencée que par les paramètres r_i, C_i et T_i.

Le moyen de trouver le régime périodique d'un ordonnancement est alors le suivant : construire la séquence jusqu'à l'instant H, en conservant en mémoire le nombre de temps creux présents sur la dernière fenêtre de taille H. Cette fenêtre glisse au fur et à mesure que la séquence est construite. S'arrêter dès que l'on a une fenêtre ne contenant que $H(1 - U)$ temps creux. On peut remarquer qu'il est possible (et même fréquent) de se trouver dans le régime périodique dès l'origine. Dans ce cas, $t_c = -1$. On peut remarquer que la méthode est exponentielle (puisque nécessitant la construction de la séquence), mais cela n'est pas problématique, puisque si l'on a besoin de connaître la durée de simulation, c'est sans doute parce-qu'on a besoin de construire cette simulation. Le problème de la complexité combinatoire de la détermination de t_c est un problème ouvert, même dans le cas de tâches indépendantes.

2.3.2 Idée de preuve

La preuve est construite comme suit : nous considérons d'abord des systèmes de tâches tels que $U = 1$ ordonnancés de façon conservative. Dans ce cas, si un temps creux apparaît, il est forcément acyclique et ne peut donc pas faire partie du régime périodique. Si l'on regarde l'état du système juste après un tel temps creux, l'occupation du processeur est donnée exactement par le réveil des tâches à cet instant. La fenêtre de taille H partant du début du temps creux considéré voit au plus $UH = H$ (puisque $U = 1$) unités de temps demandées par le réveil des tâches du système. S'il y a moins de H unités de temps de traitement générées par le réveil des tâches, il y a un autre temps creux acyclique, puisque le système a moins de H unités de temps à traiter dans la fenêtre de taille H. S'il y a exactement H unités de temps de traitement générées par le réveil des tâches, alors le système traite entre la date suivant le dernier temps creux acyclique et H unités de temps plus loin toute la demande H qui a été générée. Par conséquent H unités de temps après le denier temps creux acyclique, on trouve un point creux, qui correspond au même état qu'après le dernier temps creux acyclique.

Lorsque la charge $U \leq 1$, le problème est identique si l'on considère l'ajout d'une tâche oisive modélisant les temps creux avec les paramètres $< r_0, C_0 = H(1-U), D_0 = T_0 = H, T_0 = H >$. Le but est d'obtenir un système de tâches équivalent, tel que les temps creux périodiques sont modélisés par la tâche oisive. Une subtilité réside cependant dans le choix de la date de réveil r_0 de la tâche oisive. En effet, on pourrait être tenté de fixer $r_0 = 0$, cependant, dans l'optique de minimiser la durée d'étude (et donc de minimiser la date d'occurrence du dernier temps creux acyclique), le choix de cette date n'est pas judicieux. Nous avons donc montré qu'il fallait fixer $r_0 = t_c + 1$ afin que l'adjonction de cette tâche fictive n'augmente pas la durée de la monté en charge. La figure 2.6 illustre ce phénomène : un système de tâches donné par $\tau_1 =< r_1 = 0, C_1 = 1, D_1 = 4, T_1 = 4 >$ et $\tau_2 =< r_2 = 4, C_2 = 3, D_2 = 6, T_2 = 6 >$ est ordonnancé suivant DM (note : n'importe quel algorithme conservatif donnerait les mêmes temps creux). La charge de ce système est $U = 1/4 + 3/6$, par conséquent le nombre de temps creux par hyperpériode est $H(1-U) = 12 \times 1/4 = 3$. Le théorème 8 établit la position du cycle, qui suit le temps creux, et fixe $t_c = 2$. Si l'on fixe, comme sur le cas (a), la date de réveil de la tâche oisive à l'origine, cela a pour effet de repousser l'entrée dans le cycle du système, qui n'est alors pas équivalent au système d'origine. Le cas (b) montre que lorsque $r_0 = t_c + 1$, le système augmenté de la tâche oisive est bien équivalent au système d'origine.

Nous avons montré que l'existence de ressources critiques, ou de parties non-préemptibles de code, n'avait aucune influence sur le cycle. Enfin, dans le cas où il existe des précédences, nous avons montré que les précédences elles-mêmes avaient un comportement périodique, et qu'ainsi les résultats de cyclicité s'étendaient aussi. La figure 2.7 montre un exemple d'ordonnancement pour le système suivant :

	r_i	C_i	D_i	T_i	Précédences
τ_1	0	1	4	4	$\tau_2 \prec \tau_1$
τ_2	1	1	4	4	
τ_3	2	1	4	4	$\tau_4 \prec \tau_3$
τ_4	0	1	4	4	

Nous sommes dans un cas où on trouve des temps creux bien que la charge restante à traiter soit non-nulle : ces temps creux sont liés à des chaînes de précédences telles que la

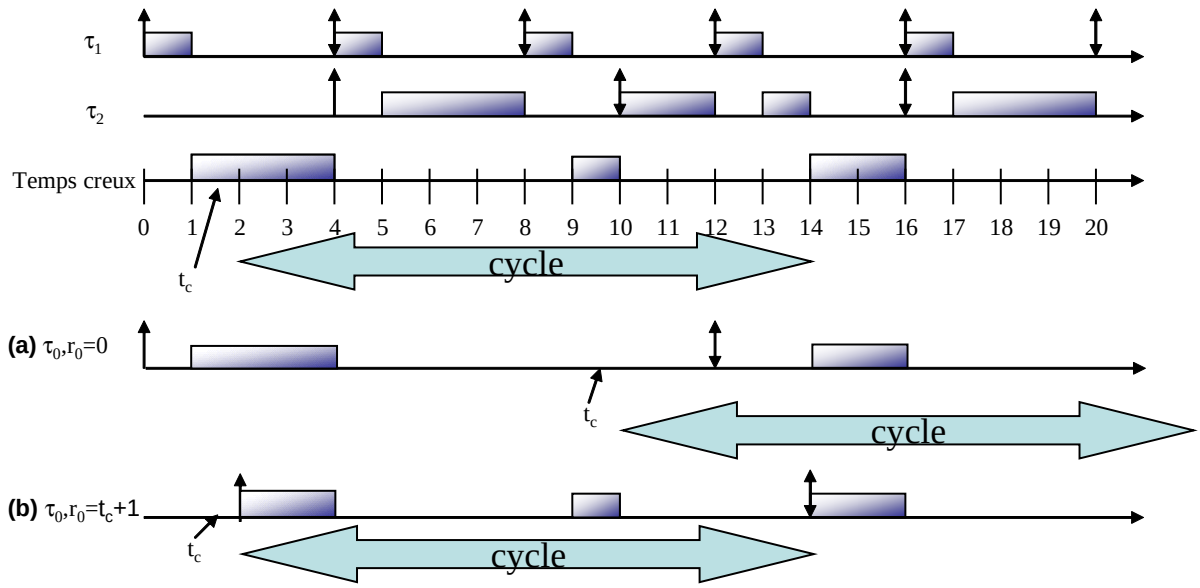

FIGURE 2.6 – Modélisation des temps creux cyclique par tâche oisive (a) en la faisant démarrer à l'instant 0 (b) en la faisant démarrer à l'instant $t_c + 1$.

tâche prédécesseur n'est pas encore réveillée alors que la tâche successeur l'est. Nous avons montré que ces chaînes de précédence étaient périodiques (trait gras sur le diagramme de charge de la figure 2.7). Par conséquent, après le dernier temps creux acyclique, la charge n'est donnée que par les tâches bloquées dans des chaînes de précédence, qui sont périodiques, et les tâches activées en cet instant. H unités de temps plus tard, la charge n'est donnée que par les mêmes tâches bloquées dans des chaînes de précédence, et les tâches activées en cet instant, qui sont les mêmes que H unités de temps plus tôt. Ce qui généralise le résultat.

2.3.3 Collaborations et publications

Cette étude a été menée en collaboration avec le Pr. Annie Geniet, pendant et après ma thèse. La preuve formelle de ce théorème est complexe et a subi de nombreuses modifications et simplifications jusqu'à sa publication dans Theoretical Computer Science [CGG04]. Les autres publications liées sont [GCG00a, GCGC98a].

2.3.4 Points forts de la démarche et perspectives

Différents travaux ont été menés depuis notre publication principale dans le domaine, notamment par Goossens et Cucu. Ils ont adopté une approche orientée algorithme d'ordonnancement particulier. Ainsi ils ont donné une borne supérieure de la cyclicité des ordonnancements à priorités fixes dans [CG06]. Supposons n tâches ordonnées par priorité croissante.

$$\text{Soit } S_i = r_i \text{ si } i = 1$$

$$S_i = \max\left\{r_i, r_i + \left\lceil \frac{S_{i-1} - r_i}{T_i} \right\rceil T_i\right\} \text{ sinon}$$

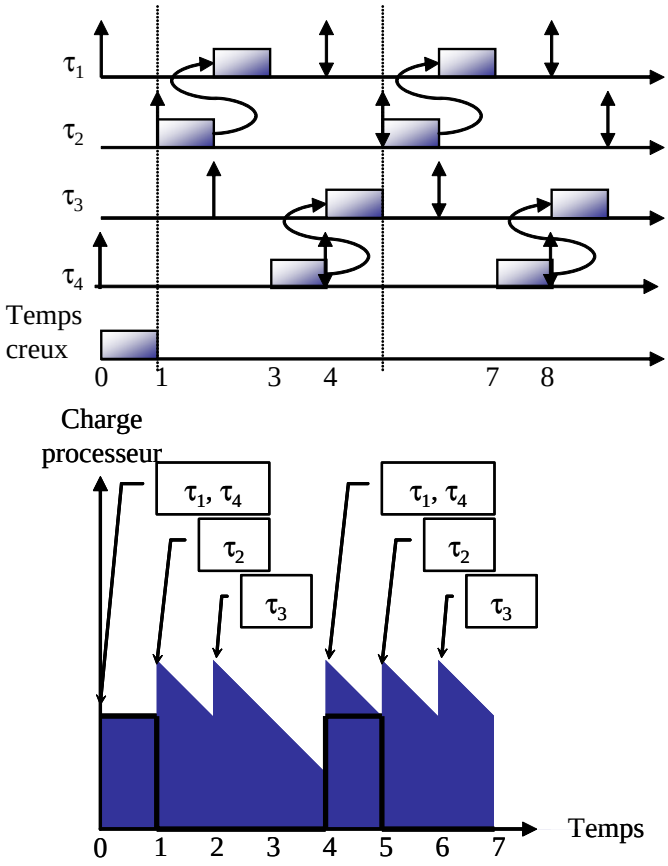

FIGURE 2.7 – Cyclicité en présence de contraintes de précédence assurées par synchronisation.

Théorème 9 *[CG06] Dans le contexte $|r_i, C_i, D_i \leq T_i, T_i$, toute séquence d'ordonnancement valide générée par un algorithme à priorités fixes se comporte périodiquement sur $[S_n..S_n + H[$.*

Ce théorème présente l'avantage de permettre un calcul polynomial d'une borne supérieure de l'entrée dans le cycle. Cependant ce n'est qu'une borne supérieure. Sachant que d'après le théorème 8, $t_c \leq S_n$, le théorème 9 pourrait être utilisé pour borner polynomialement t_c de façon spécifique au système étudié.

Les mêmes auteurs ont étendu le contexte de leur résultat aux échéances arbitraires :

$$\text{Soit } X_i = r_i \text{ si } i = 1$$

$$X_i = \max\{r_i, r_i + \left\lceil \frac{X_{i-1} - r_i}{T_i} \right\rceil T_i\} + H_i \text{ sinon avec } H_i = PPCM_{j=1..i}(T_j)$$

Théorème 10 *[CG07] Dans le contexte $|r_i, C_i, D_i, T_i|$, toute séquence d'ordonnancement valide générée par un algorithme à priorités fixes se comporte périodiquement sur $[X_n..X_n + H[$.*

Ce théorème est le seul résultat concernant les systèmes à échéances arbitraires. Il est assez pessimiste quant à l'espoir de trouver un cycle rapidement. Je collabore actuellement avec Joël Goossens et Liliana Cucu à l'amélioration de ce résultat, et à la généralisation des résultats de cyclicité au cas multiprocesseur dans le cadre de l'ordonnancement global (le cas ordonnancement partitionné multiprocesseur est une extension triviale du cas monoprocesseur), en utilisant l'approche des temps creux acycliques. Ainsi, la figure 2.8 montre l'ordonnancement EDF sur 2 processeurs de 3 tâches dont une avec échéance arbitraire. Nous voyons que bien qu'elles soient simultanées, le régime périodique ne s'instaure pas immédiatement, contrairement au cas monoprocesseur. Nous voyons aussi que le régime périodique est la première fenêtre de taille H ne contenant qu'un seul temps creux (i.e. $H(m-U)$ temps creux pour m processeurs). Nous travaillons actuellement sur la preuve de cette conjecture.

2.4 Bilan et perspectives

On peut discuter de l'apport des approches en-ligne par rapport aux approches hors-ligne, et de la programmation basée sur le temps par rapport à la programmation basée sur les événements. Les approches hors-ligne ne prennent en compte que des tâches concrètes, par conséquent, elles ne peuvent pas s'appliquer sur les applications programmées sur les événements. Certes, de nos jours, nombre d'applications très critiques (notamment dans le domaine du transport) n'utilisent presque que des tâches basées sur le temps (i.e. strictement périodiques), mais cela diminue la réactivité du système. Ainsi, dans la programmation basée sur le temps, les événements déclenchent une interruption matérielle, ce qui déclenche l'exécution d'une routine de traitement d'interruption (routine de traitement d'interruption (ISR)). Cette routine stocke l'événement dans une variable (typiquement une variable globale) ou encore dans un tampon permettant de stocker plusieurs événements à traiter. Cette variable ou ce tampon est scruté périodiquement par la tâche d'acquisition correspondante $\tau_{a,timetriggered}$. Entre l'arrivée effective de l'événement et son traitement par la tâche d'acquisition, une durée maximale

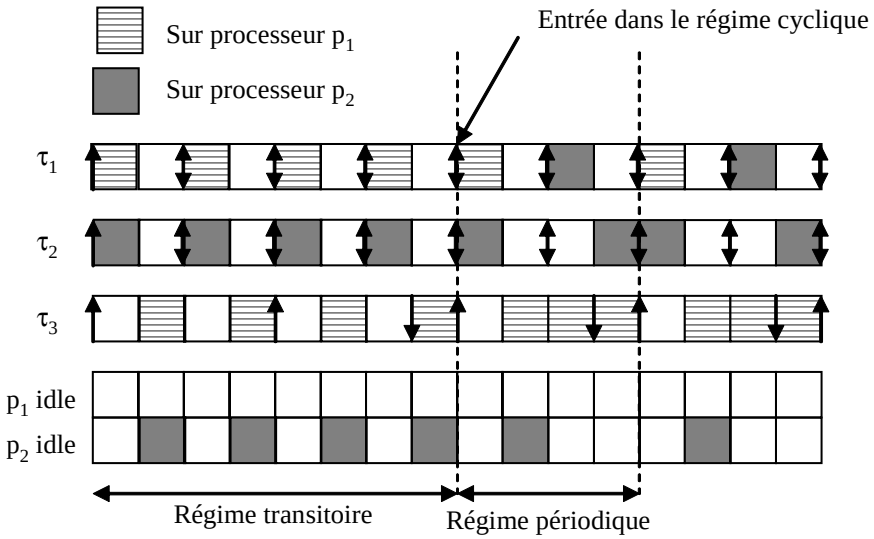

FIGURE 2.8 – L'ordonnancement d'un système simultané à échéances arbitraires de tâches sur 2 processeurs.

$TR_{ISR} + T_{a,timetriggered} + TR_{a,timetriggered}$ peut donc s'écouler. Si l'on compare cela avec une implémentation dirigée par les événements, la routine de traitement d'interruption déclenche directement la tâche d'acquisition $\tau_{a,eventtriggered}$: dans ce cas le système prend en compte l'événement $TR_{ISR} + TR_{a,eventtriggered}$ unités de temps après son occurrence. Ce comportement a un impact important sur la période de la tâche d'acquisition considérée. Supposons par exemple que les contraintes temporelles de la tâche soient liées aux occurrences de l'événement. Par exemple, supposons que l'événement soit un capteur de détection de passage : il est dans l'état haut lorsqu'il est sensibilisé par le passage d'un objet, dans l'état bas sinon. Supposons que le capteur soit sensibilisé pendant au moins 5 ms au passage d'un objet, et qu'au moins 5 ms sépare le passage de 2 objets successifs. La tâche d'acquisition ne doit pas manquer de passage d'objet, il ne faut donc pas manquer de front haut ni de front bas. Dans le cas dirigé par le temps, il faut donc que $TR_{ISR} + T_{a,timetriggered} + TR_{a,timetriggered} \leq 5ms$. Si l'on néglige le temps de traitement de l'ISR, et que l'on est dans un système à échéance sur requête, la période de la tâche doit être $T_{a,timetriggered} \leq 2,5ms$. Dans le cas dirigé par les événements, on doit avoir $T_{a,timetriggered} \leq 5ms$. En supposant les temps de traitement identiques, l'acquisition de cet événement consomme deux fois moins de temps processeur dans le cas dirigé par les événements. Ainsi, il m'est arrivé de discuter avec des industriels d'une étude de cas où toutes les tâches étaient dirigées par le temps, sauf une devant réagir rapidement à un événement, qui était dirigée par les événements.

Il serait donc intéressant, dans les approches hors-ligne, de prendre en compte une partie de trafic sporadiquement périodique, en intégrant par exemple des approches hors-ligne et des études portant sur le traitement des sporadiques.

Une autre utilisation possible des approches hors-ligne consiste à séquencer des applications développées en approche synchrone, et ainsi remplacer la synchronisation d'automates par le séquencement optimal des automates.

Enfin, la plupart des applications s'exécutant sur de petits calculateurs, comme les

applications spatiales, possèdent différents modes de fonctionnement (pendant le lancement, en exploitation, fenêtre de transmission des télémesures, entrée dans l'atmosphère). Le problème des changements de mode n'a été que très peu étudié et pourrait être un terrain d'investigation intéressant.

Chapitre 3

Ordonnancement en-ligne

Ce chapitre traite de nos apports dans le cadre de l'ordonnancement en-ligne. Ces apports ont eu lieu principalement dans deux domaines : l'étude de facteurs pratiques, notamment via l'étude les contraintes de précédence généralisées, et l'étude des transactions, aussi bien en priorités fixes qu'en priorités dynamiques. Ce modèle est issu d'une étude de cas réelle, dont j'ai dirigé le développement du système embarqué. La première section décrit brièvement l'étude de cas, afin de justifier le modèle des transactions qui est présenté en section 2. La troisième section présente notre étude des précédences généralisées, et le chapitre est conclu par un bilan et des perspectives de recherche sur l'ordonnancement en-ligne.

3.1 Etude de cas

3.1.1 Contexte du projet

Un drone Unmanned Aerial Vehicle (UAV) est un aéronef capable de voler et d'effectuer une mission sans présence humaine à bord. C'est un système qui est réalisé dans le but d'assurer une ou plusieurs missions. Il existe plusieurs catégories de drone selon la taille, les performances ou le type de mission.

Le projet drone miniature Aéronef Miniature Autonome de Détection et d'Observation (AMADO) entre dans le cadre du concours de drone miniature lancé en 2002 par la Délégation Générale pour l'Armement (DGA) et l'Office National d'Études et de Recherches Aérospatiales (ONERA). Ce concours permet de montrer la faisabilité technique et l'intérêt opérationnel des drones miniatures utilisés comme aide au fantassin dans sa progression en milieu hostile. Le concours est restreint aux drones miniatures ; ce sont des drones qui n'excèdent pas 70 cm dans leur plus longue envergure. Cette exigence impose une miniaturisation importante des composants du drone. De plus, le drone doit pouvoir évoluer hors de la vue du pilote, d'où l'exigence de pilotage aux instruments. Les autres exigences sont les suivantes : capacité de vol autonome, décollage et atterrissage automatiques, compacité de la station sol, transmission d'images en temps réel, qualité d'image fournie, capacité d'orientation du capteur pour viser un objectif, facilité et rapidité de mise en œuvre, autonomie énergétique. L'ENSMA et l'Université de Poitiers ont répondu conjointement, sous l'impulsion d'Alain Farcy du Laboratoire d'Etudes Aérodynamiques (LEA) et moi-même pour le LISI.

Initialement, le projet implique trois laboratoires poitevins :

- Le LEA, sous la responsabilité d'Alain Farcy, définit le comportement dynamique du mini drone, la répartition des masses des composants embarqués, la réalisation et la simulation du pilote automatique. Il met à la disposition de ce projet la soufflerie bois T5 de l'ENSMA, la balance dard 6 composantes "faibles efforts", le banc d'essai motorisation électrique, pour la mesure des performances et l'optimisation de la chaîne propulsive électrique complète du drone miniature (stockage de l'énergie, contrôleur, moteur, hélice).
- Le Signal, Image, Communication (SIC), sous la responsabilité de Christian Chatellier, est chargé de la compression des images et de la mise en place des dispositifs pour transmettre celles-ci vers la station sol. Le SIC met à la disposition de ce projet des générateurs modulateurs et des analyseurs de signaux. Enfin, au niveau image, le SIC dispose d'un banc de mesure et de prise de vue 3D, de caméras couleurs tri CCD, d'une caméra linéaire couleur haute résolution et d'appareils photo numériques haute résolution.
- Le LISI, sous ma responsabilité, se charge du développement de l'application au niveau de la station sol et au niveau du système embarqué. Le LISI met à disposition ses compétences en conception et validation de systèmes temps réel, ainsi que la plate-forme DARTSVIEW de conception et génération de code.

Par la suite, deux autres laboratoires poitevins, le Laboratoire de Mécanique des Solides (LMS) apportant des compétences matérielles sur cartes micro-contrôleur, et le Laboratoire d'Automatique et d'Informatique Industrielle (LAII) apportant des compétences en identification, lois de commande, ont rejoint le projet.

Suite aux exigences du concours, le drone doit se déplacer d'un point à un autre de façon autonome en transmettant une séquence vidéo à une station au sol. Il doit être en mesure de se diriger et donc d'entreprendre un virage stabilisé. Le système proposé par l'équipe est un engin hybride à voilure fixe (aile delta), possédant une capacité de vol stationnaire. La propulsion est électrique, monomoteur à hélice. L'aile delta autorise un large domaine de vol. Elle est équipée d'un récepteur radio, d'une caméra, d'un récepteur GPS, d'une centrale inertielle, d'un microcontrôleur MPC555 dans sa version initiale, et d'un modem bidirectionnel half-duplex. Le drone est pilotable en trois modes :

- Le mode assisté, le pilote envoie des consignes (vitesse verticale, vitesse et roulis) de la station sol vers le drone. Le système est alors chargé de piloter en fonction de ces consignes en transformant le couple vitesse et vitesse verticale en tangage et puissance du moteur.
- Le mode mission sol permet à l'utilisateur de définir un parcours et des actions à effectuer sur certains points de ce parcours, et sans intervention humaine, le drone devra suivre ce parcours et réaliser les actions correspondantes (les informations envoyées sont aussi des consignes).
- Le mode manuel, le pilote envoie directement au drone des commandes de vol. Ces commandes sont envoyées via une radiocommande.

Au niveau de la station sol, il s'agit d'assister le pilote du drone en lui présentant de manière adéquate les informations provenant du matériel embarqué (vidéo, attitude, position, vitesse, mode en cours, état des capteurs). Les informations échangées entre le drone et la station sol sont de quatre ordres : les informations de pilotage manuel pur (commandes), les consignes en mode assisté et en mission sol, les informations relatives

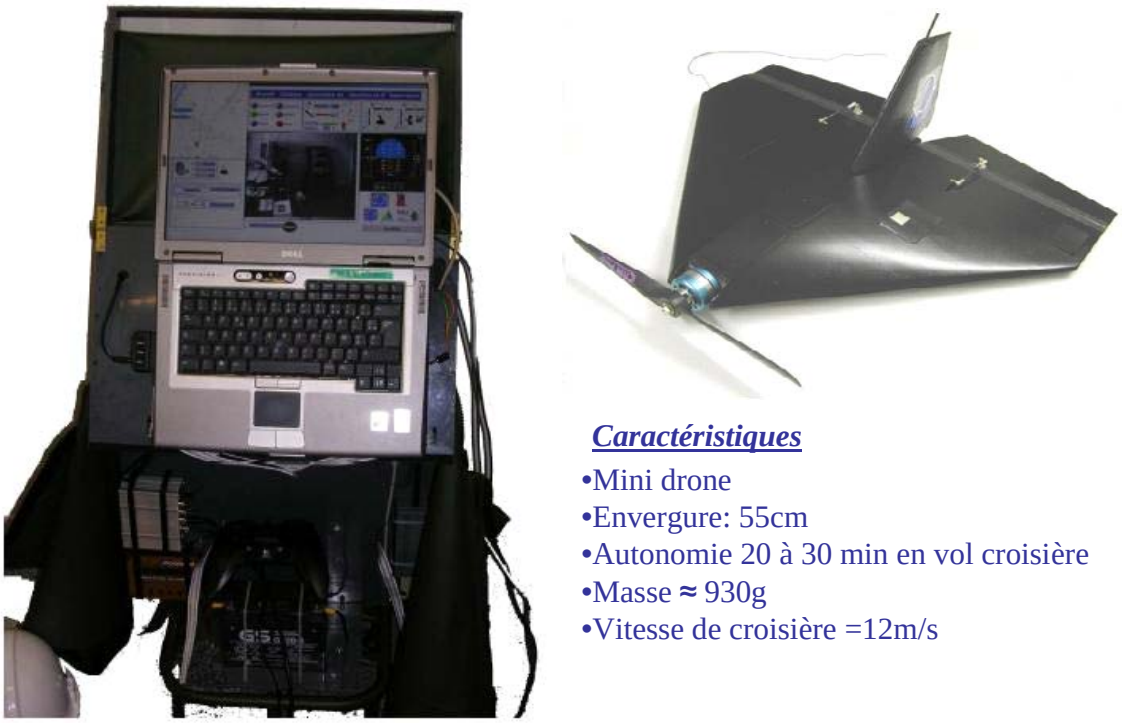

FIGURE 3.1 – La station sol et le drone AMADO.

à la position et l'attitude, et les images qui constituent le quatrième type d'informations.

Le rôle de la station sol (voir figure 3.1) est d'une part d'envoyer des commandes de vol et des consignes ou des plans de vol au drone et d'autre part d'afficher et d'enregistrer toutes les données du vol (vidéo, trajectoire, attitude, etc.).

3.1.2 Problématique ordonnancement

L'implémentation du système embarqué sur le drone, que Karim Traoré et moi-même avons mise en place, est très consommatrice en temps de calcul dans sa version monoprocesseur principalement à cause de calculs matriciels intensifs du modèle en quaternions permettant de déterminer l'attitude du drone en fonction des informations de la centrale inertielle. Le système ne pouvait pas être validé avec les modèles d'ordonnancement classiques, à cause d'une trop grande approximation de l'application réelle par modèle de tâches classiques. L'implémentation, effectuée sur un exécutif OSEK tournant sur un microcontrôleur de fréquence 40 MHz, utilise notamment 2 communications série (antenne GPS, MODEM de communication avec la station sol), et une communication CAN (communication avec une centrale inertielle). L'architecture matérielle employée ne mémorise pas les données reçues, par conséquent, les tâches d'acquisition série et CAN doivent pouvoir lire et stocker les données reçues avant qu'elles ne soient écrasées par les prochaines. Ainsi, au niveau du réseau CAN ayant un débit de 1 Mbps, la centrale inertielle envoie 3 trames de 8 octets de données à suivre toutes les 20 ms. Il faut évidemment lire et stocker chaque trame avant l'arrivée de la prochaine, sous peine de perdre les informations

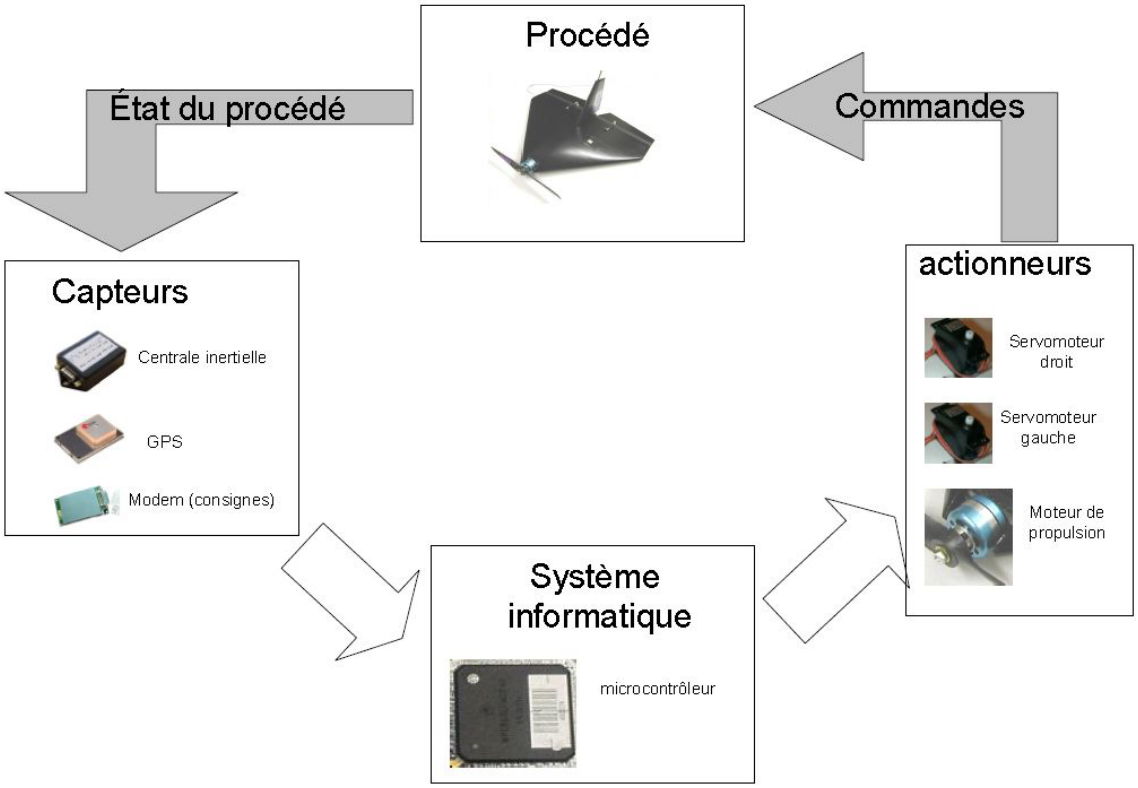

FIGURE 3.2 – Principe de fonctionnement drone-station sol.

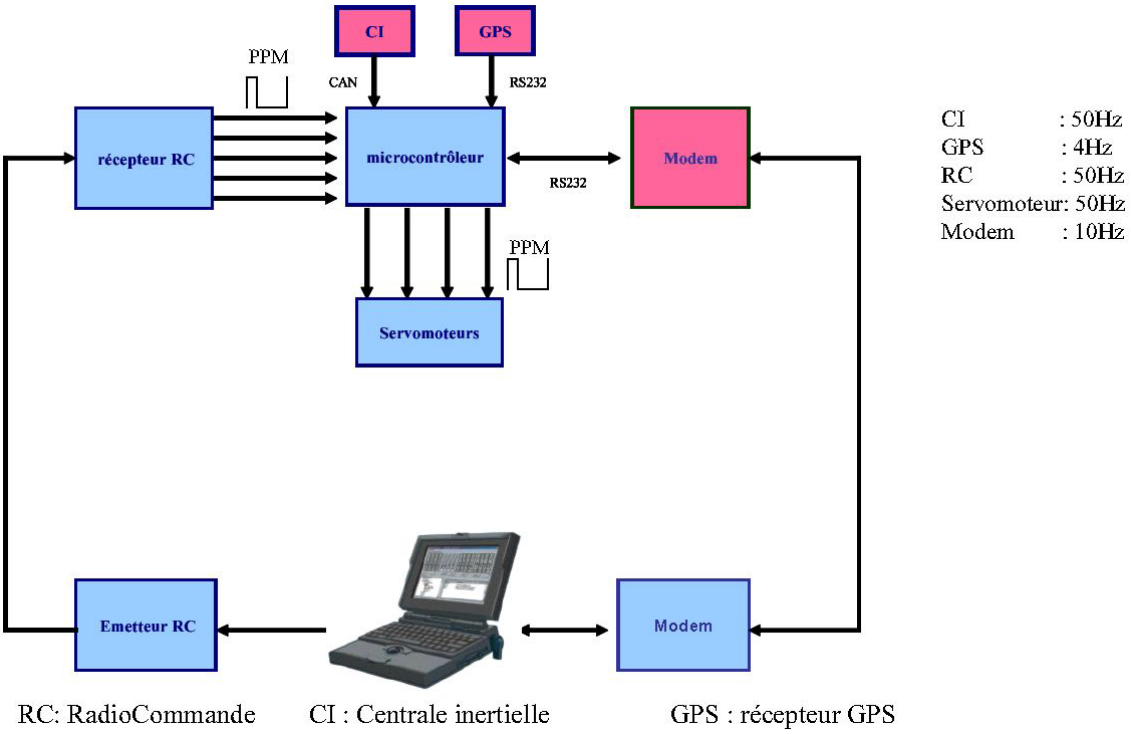

FIGURE 3.3 – Capteurs/actionneurs sur le drone AMADO.

d'attitude reçues. Il en va de même pour les informations reçues par liaison série : les octets sont à stocker un par un, jusqu'à réception complète des données. Ainsi, le GPS envoie des trames au format National Marine Electronics Association (NMEA), un protocole utilisé pour la communication entre équipements, initialement marin, contenant plus d'une centaine d'octets, toutes les 250 millisecondes, au rythme d'un octet toutes les 173 μsecondes. Il en va de même pour le MODEM, qui envoie à la même vitesse une trentaine d'octets toutes les 10 millisecondes. Si l'on modélise un tel système avec des tâches classiques périodiques, il nous faut prendre en compte le surcoût processeur lié au traitement des données entrantes. Cependant, ce surcoût, s'il est lissé uniformément sur le temps, n'est pas négligeable et représente une source de pessimisme. En effet, cela revient à considérer que les arrivées de données sont continuelles, alors qu'elles sont périodiques : pour le GPS, une centaine d'octets sont à traiter au rythme d'un toutes les 173 μsecondes pendant 17.3 millisecondes, puis il n'y a pas d'arrivée pendant 232.7 millisecondes, cependant, une tâche de vérification et de traitement des données a été déclenchée par l'arrivée du dernier octet de la trame NMEA. Si nous cherchons à modéliser le tout avec des tâches sporadiquement périodiques, il nous faudra considérer un instant critique qui est irréaliste et entraîne du pessimisme lors de la validation.

Nous nous sommes donc intéressés au modèle des transactions : ce modèle considère des ensembles de tâches, pouvant être décalées dans le temps, activées par un événement. Un événement extérieur déclenchant une suite d'action correspond à une transaction. Par exemple, dans notre étude de cas, la transaction modélisant les traitements effectuées pour recevoir, et traiter les informations GPS est activée par le premier octet d'une trame NMEA. Une transaction est sporadiquement périodique, ce qui est la cas pour les communications avec les périphériques de notre étude de cas. Il existe de nombreux cas, lorsque plusieurs trames CAN sont attendues en série, ou lorsqu'un protocole de type série est utilisé, dans lesquels le modèle des transactions peut être utilisé. L'avantage d'un tel modèle est qu'il prend en compte le décalage entre les tâches, et évite ainsi de considérer qu'elles peuvent toutes interférer comme lorsqu'on les modélise avec des tâches classiques.

3.1.3 Etat du projet

La figure 3.4 montre différentes cellules sur lesquelles le système embarqué a été porté (ou sera porté prochainement pour le *trainer*). Développé initialement sur une plaque permettant d'évaluer l'impact des composants sur les autres, la première version a été implantée dans une cellule en kevlar. Lors d'essais en vol avec cette cellule, nous avons dû utiliser une catapulte, la vitesse minimale étant trop élevée pour un lancer manuel. Malheureusement, un décollage a été tragique pour la cellule. La cellule carbone a ensuite été créée pour des essais statiques. La cellule carbone-kevlar a permis de valider le système en soufflerie : virages stabilisés, commandes dans différents modèles de vol, et a été l'objet de différents essais afin de permettre à l'équipe identification du LAII de mettre en place un modèle qu'ils étudient actuellement. La cellule de vol verticale a été développée afin d'étudier la maniabilité en vol vertical. Le trainer, d'envergure plus importante, et de pilotage simple, attend actuellement la carte bi-processeur sur laquelle nous portons le système embarqué. Cette cellule permettra des tests en vol de différents capteurs/actionneurs en ayant moins de contraintes de poids que les petites cellules précédentes.

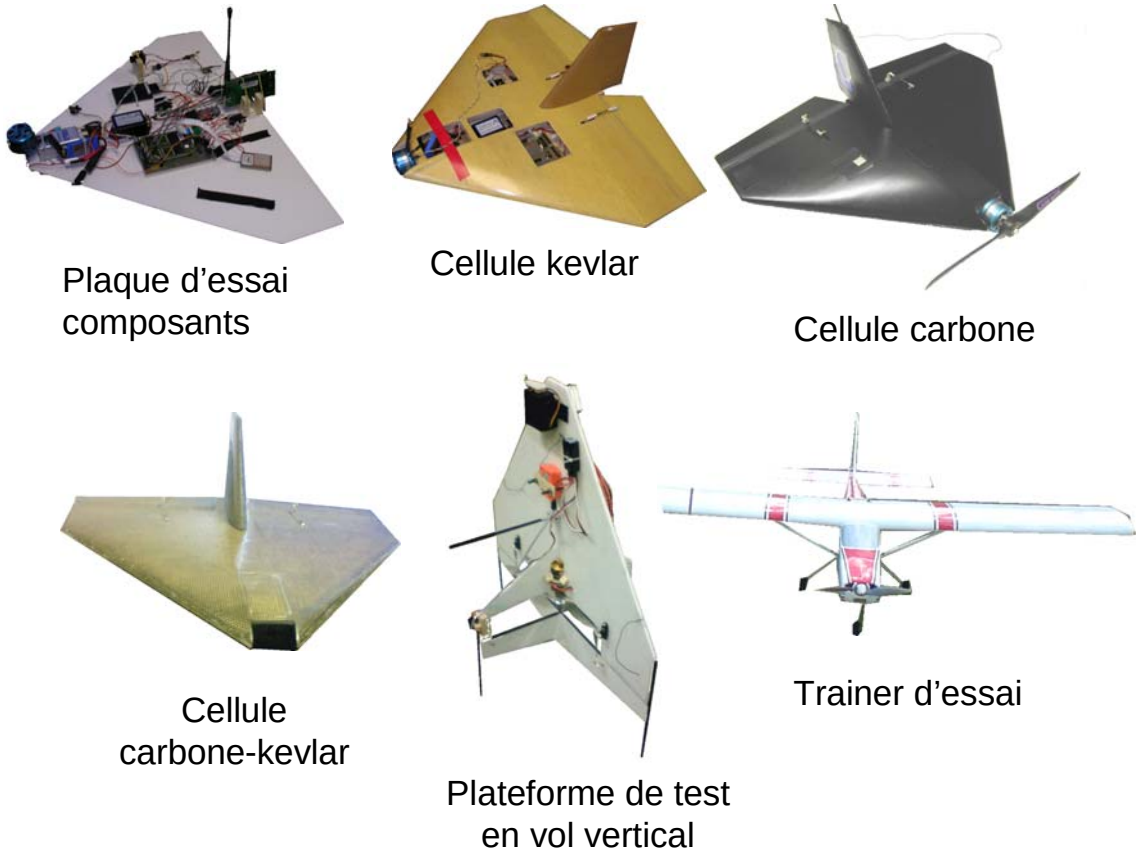

FIGURE 3.4 – Capteurs/actionneurs sur le drone AMADO.

3.2 Apports au modèle des transactions

3.2.1 Définition des transactions

[PH98] Un système est vu comme un ensemble de transactions Γ_i, chaque transaction est sporadiquement périodique (elle est déclenchée par un événement) avec un écart minimal séparant 2 activations successives appelé période T_i. Chaque transaction Γ_i contient $|\Gamma_i|$ tâches $\tau_{ij}, i = 1..|\Gamma_i|$. Chaque tâche τ_{ij} est caractérisée par une date d'activation O_ij calculée relativement au début de la transaction (i.e. de l'événement déclencheur), et peut avoir une gigue d'activation J_{ij} qui permet d'exprimer l'imprécision que peut avoir son activation : la tâche τ_{ij} est activée entre O_{ij} et $O_{ij} + J_{ij}$ unités de temps après l'événement déclencheur de la transaction. Ce modèle d'activation permet de représenter parfaitement les tâches liées à l'acquisition de données arrivant en série. En effet, le déclenchement de la transaction représente alors l'arrivée du premier message de la série, ainsi que la première lecture (i.e. $O_{i1} = 0$), puis la seconde tâche s'exécute au mieux au bout d'un temps $O_{i2} =$*délai minimal entre l'arrivée de deux éléments*, et au plus au bout d'un temps $O_{i2} + J_{i2}$. La gigue d'activation représente l'incertitude qui est liée à la régularité des données arivant sur le bus de communication. La dernière tâche de la

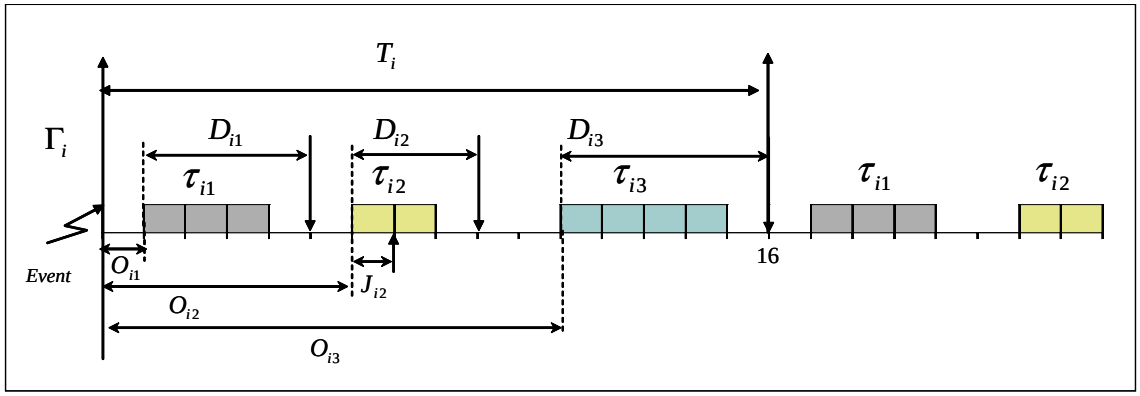

FIGURE 3.5 – Une transaction constituée de 3 tâches.

transaction correspond à l'arrivée de la dernière trame/paquet/octet/etc. composant le message, et correspond généralement au traitement du message (vérification, conversion dans une structure de données interne, mise à disposition pour le reste du système).

Formellement, un système de transactions (voir figure 3.5) est donc défini par :

- $\Gamma = \{\Gamma_1, ..., \Gamma_{|\Gamma|}\}$
- $\Gamma_i =< \{\tau_{i1}, ..., \tau_{i|\Gamma_i|}\}, T_i >$ avec T_i période de la transaction, durée minimale séparant 2 activations successives,
- $\tau_{ij} =< C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij} >$
  - C_{ij} est le WCET de la tâche
  - O_{ij} son *offset*, représentant sa date de réveil, donnée relativement au début de la transaction,
  - D_{ij} son délai critique, exprimé relativement à sa date de réveil,
  - J_{ij} sa gigue d'activation : si la transaction est activée à la date t, la tâche est activée entre les instants $t + O_{ij}$ et $t + O_{ij} + J_{ij}$,
  - B_{ij} représente le facteur de blocage dû à l'utilisation d'un protocole de gestion de ressources.

Remarquons qu'il n'est pas nécessaire que les acquisitions soient implémentées sous forme de tâches pour les modéliser sous forme d'une transaction. Ainsi, on peut imaginer un système dans lequel les données sont collectées et mémorisées par des routines de traitement d'interruption, puis une tâche de traitement est exécutée à la fin de la réception du message. Dans ce cas, chaque ISR peut être modélisée par une tâche d'une transaction.

Théorème 11 *[Tin92, Tin94b, Tin94a] Un instant critique se produit lorsqu'une tâche de chaque transaction est activée simultanément, retardée par sa gigue maximale.*

La difficulté liée au modèle des transactions est qu'il existe plusieurs instants critiques possibles : en effet, chaque combinaison d'activations simultanées de tâches peut créer un instant critique, et il faudrait étudier $\prod_{i=1..|\Gamma|} |\Gamma_i|$ instants critiques potentiels. Même à l'échelle d'une application comme celle du drone, cela donne plusieurs dizaines de milliers d'instants critiques potentiels.

Toutes les études menées sur le modèle des transactions, que ce soit en FPP ou en priorités dirigées par les échéances (EDF), ont concerné le calcul du pire temps de réponse, jusqu'à notre étude utilisant la fonction de la demande processeur en EDF. Dans le cas

des FPP, nous avons nous aussi considéré une approche basée sur le calcul de pire temps de réponse qui utilise le calcul de période d'activité.

3.2.2 Etude en priorités fixes

Méthode exacte

En priorités fixes, chaque tâche est caractérisée par une priorité P_{ij}. Puisque nous étudions les tâches une par une, nous notons τ_{ua} la tâche analysée (*under analysis*). L'ensemble $hp_i(\tau_{ua})$ est l'ensemble des indices des tâches appartenant à la transaction Γ_i de priorité au moins aussi grande que celle de τ_{ua}.

Théorème 12 *[PH98] La contribution pire cas d'une transaction Γ_i sur une tâche τ_{ua} est obtenue quand la première activation d'une tâche $\tau_{ic}, ic \in hp_i(\tau_{ua})$, dite candidate, coïncide avec l'instant critique après avoir été retardée par sa gigue maximale.*

Le théorème 12 apporte l'information supplémentaire par rapport au théorème 11 que les instants critiques à considérer concernent des tâches au moins aussi prioritaires que la tâche analysée.

Théorème 13 *[Tin94a] Lors de l'étude d'une période d'activité, les instances déclenchées avant celle-ci peuvent être ignorées à condition d'étudier la période d'activité correspondant à celles-ci.*

Ce théorème montre que lorsqu'on étudie une période d'activité initiée par un candidat par transaction, il n'est pas nécessaire d'étudier les instances de tâches réveillées avant l'instant critique candidat. En effet, supposons q'une instance réveillée avant la période d'activité puisse participer à la période d'activité, alors en étudiant la période d'activité dans laquelle elle est candidate, nous retrouverons la période d'activité actuelle.

Lorsqu'on étudie un scénario d'instant critique, celui-ci correspond à l'activation d'une tâche candidate τ_{ic} par transaction, les tâches candidates subissent leur gigue maximale. Les autres tâches pouvant retarder la tâche analysée sont :

- les tâches de $hp_i(\tau_{ua})$ pouvant être avec une certaine valeur de gigue activées à l'instant critique. Leur pire interférence (retard appliqué) sur τ_{ua} correspond alors à un réveil à l'instant critique, suivi de réveils au plus tôt ; on nomme Set_1 l'ensemble des instances concernées ;
- les tâches de $hp_i(\tau_{ua})$ réveillés dans la période d'activité à partir de l'instant critique avec une gigue nulle ; cet ensemble d'instances constitue Set_2.

L'ensemble Set_1 est statique pour un instant critique candidat considéré, alors que l'ensemble Set_2 évolue en fonction de la longueur de la période d'activité. La figure 3.6 illustre ces ensembles, dans le cas où on considère un instant critique initié par la tâche candidate τ_{i1} : celle-ci est retardée de sa gigue maximale 4, mais étant donné que la tâche τ_{i2} a une gigue de 20 unités de temps, 2 instances de τ_{i2} forment avec une instance de τ_{i1} l'ensemble Set_1. Lors de l'étude d'un instant critique initié par une tâche candidate τ_{ic} dans chaque transaction Γ_i, l'instant critique a lieu $O_{ic} + J_{ic}$ unités de temps après l'activation de la transaction. Le décalage entre l'instant critique et la première instance de τ_{ij} étant réveillée à ou après l'instant critique, appelé phase, est noté

$$\Phi_{ijc} = (O_{ij} - (O_{ic} + J_{ic}))modT_i$$

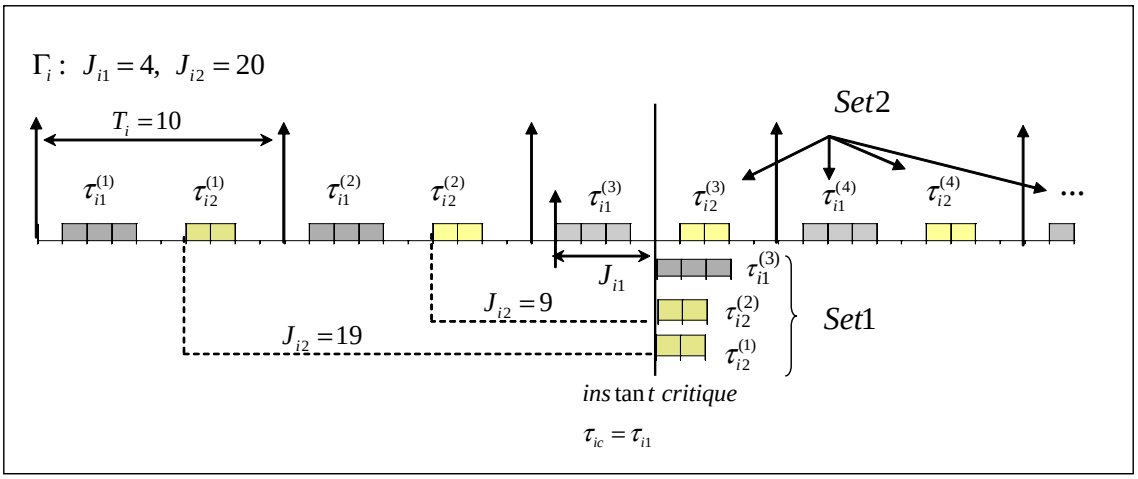

FIGURE 3.6 – Instances d'une transaction entrant dans la considération d'un instant critique candidat.

On peut à partir de la phase calculer le nombre d'instances de la tâche τ_{ij} présentes dans Set_1 et dans $Set_2(t)$, et par là l'interférence apportée à la période d'activité initiée par l'instant critique correspondant à une activation de τ_{ic}.

$$I_{ijc}^{Set_1} = \left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor C_{ij}$$

$$I_{ijc}^{Set_2}(t) = \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil C_{ij}$$

D'où la contribution apportée par τ_{ij} à une période d'activité de longueur $\geq t$ initiée par l'instant critique correspondant à la tâche candidate τ_{ic} :

$$W_{ijc}(t) = I_{ijc}^{Set_1} + I_{ijc}^{Set_2}(t)$$

De là, la contribution apportée par la transaction Γ_i à cette période d'activité :

$$W_{ic}(t) = \sum_{\forall j \in hp_i(\tau_{ua})} W_{ijc}(t)$$

Le calcul de la longueur d'une période d'activité candidate est ensuite obtenu de façon classique comme dans [Leh90] (voir théorème 4) en remplaçant l'interférence des tâches par l'interférence des transactions, et en calculant les temps de réponse des instances de τ_{ua} jusqu'à en trouver une dont le temps de réponse est $\leq T_{ua}$. Ce calcul est un calcul exact de pire temps de réponse. Comme pour la RTA classique, on peut prendre en compte le facteur de blocage B_i lorsque des ressources sont en jeu et qu'on utilise un protocole de gestion de ressources.

Le pire temps de réponse d'une tâche est alors la valeur maximale de tous les instants critiques candidats obtenus. On calcule donc un nombre exponentiel de périodes d'activité. Palencia et Harbour [PH98] ont proposé une méthode approchée utilisant des enveloppes d'interférence de transaction, enveloppes affinées par Nolin et Turja.

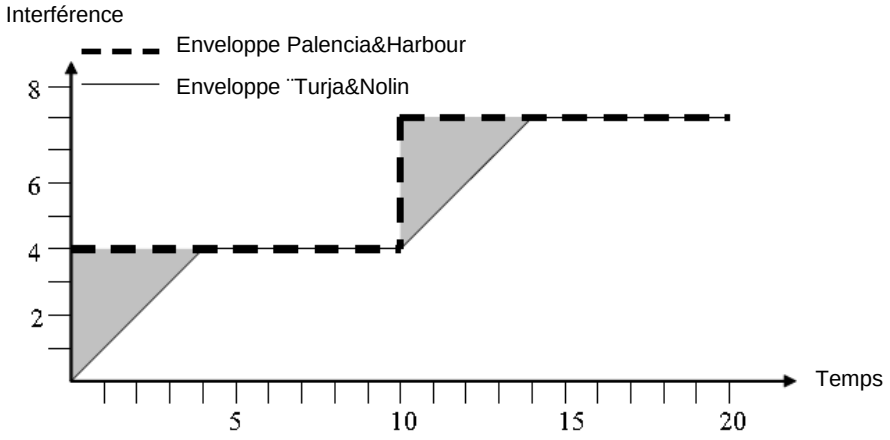

FIGURE 3.7 – Interférence considérée par Turja et Nolin.

Méthodes approchées utilisant des enveloppes

Palencia et Harbour proposent dans [PH98] d'utiliser une borne supérieure de l'interférence d'une transaction quel que soit le candidat à l'instant critique considéré pour les autres transactions que la transaction contenant la tâche analysée. Ainsi, une borne supérieure à l'interférence d'une transaction Γ_i sur une tâche analysée τ_{ua} est donnée par :

$$W_i^{Palencia}(t) = \max_{c \in hp_i(\tau_{ua})} W_{ic}(t) \text{ pour } i \neq u$$

La transaction Γ_u est traitée comme dans la méthode exacte, ce qui permet de ne considérer que $|\Gamma_u|$ instants critiques. Le pessimisme de la méthode correspond au fait qu'en fonction de la longueur de la période d'activité, le candidat entraînant la pire interférence d'une transaction n'est pas forcément le même.

Une implémentation efficace de cette technique a été proposée dans par Turja et Nolin dans [MTN04a]. Ces auteurs ont remarqué dans [MTN04b] que la dérivée de l'interférence d'une transaction ne pouvait dépasser 1. En effet, l'enveloppe utilisée par Palencia et Harbour est une enveloppe en escaliers, qui peut surestimer l'interférence dans certains cas. Ils ont donc proposé une méthode remplaçant l'enveloppe en escalier par une enveloppe en rampes (les marches de l'escalier sont transformées en rampes de dérivée 1). On peut noter que même dans la méthode classique de RTA, on considère une fonction en escalier de l'interférence, mais l'interférence réelle pourrait être représentée en rampes. Cela n'apporte pas de pessimisme dans l'analyse RTA classique, mais étant donné que l'enveloppe peut présenter des cas irréalistes en cas de changement de tâche candidate, la surestimation de l'interférence est moins grande lorsqu'on considère une fonction en rampes (voir figure 3.7). Notons que lorsque plusieurs tâches peuvent se chevaucher à l'intérieur d'une transaction, la dérivée pourrait être supérieure à 1, ce qui est inutile. Turja et Nolin [MTN04b] transforment donc une transaction en forme normale avant étude, ce qui a pour effet de regrouper plusieurs tâches qui se chevauchent, de façon à avoir une enveloppe de dérivée 1. L'enveloppe d'interférence obtenue par la méthode de Turja et Nolin donne une fonction d'interférence $W_i^{Nolin}(t) \leq W_i^{Palencia}(t)$.

Contributions

L'observation de l'étude de cas du drone nous a fait réaliser que bon nombre de transactions modélisant des tâches de lecture sur un bus de type série avaient une allure particulière, à savoir des tâches courtes et régulières (mémorisation des parties d'un message), suivies d'une tâche longue de traitement, ceci pendant la durée de réception d'un message. Nos premières études sur le modèle des transactions ont consisté à exhiber des propriétés permettant de simplifier les calculs de temps de réponse par rapport à la méthode de Turja et Nolin, dans le cas où les transactions sont des tâches de lecture sur bus série. Ensuite, nous avons élargi nos études au modèle des transactions lui-même, nous avons montré qu'il était une généralisation des modèles multiframes, et nous avons proposé une méthode mixte et mené des études de performances comparatives entre les différentes méthodes de calcul de pire temps de réponse.

Transactions série

Dans ce modèle, nous ne prenons pas la gigue en considération, les éléments unitaires composant un messages étant considérés comme réguliers.

Définition 8 *Une transaction série Γ_i est une transaction telle que :*

- *Gigue de démarrage nulle* $\forall j \in \{1..|\Gamma_i|\}, J_{ij} = 0$
- *Arrivées régulières des tâches :* $\exists p_i$ *tel que* $O_{ij} = (j-1)p_i$
- *Présence de tâches d'acquisition identiques, et d'une tâche de traitement plus longue :*
  - *Tâches d'acquisition de durée identique C_i, régulières, à échéance correspondant à l'activation de l'acquisition suivante (pas de mémorisation matérielle), définies par* $\forall j \in \{1..|\Gamma_i|-1\}, \tau_{ij} =< C_{ij} = C_i, O_{ij} = (j-1)p_i, D_{ij} = p_i, J_{ij} = 0, B_{ij} >$
  - *Tâche de traitement activée lors de l'arrivée de la fin d'un message :* $\tau_{i|\Gamma_i|} =< C_{i|\Gamma_i|} \geq C_i, O_{i|\Gamma_i|} = (|\Gamma_i|-1)p_i, D_{i|\Gamma_i|}, J_{i|\Gamma_i|} = 0, B_{i|\Gamma_i|} >$
- *L'écart entre deux messages successifs est plus grand que l'écart entre deux éléments successifs d'un même message même en prenant en compte la durée des traitements :* $T_i - |\Gamma_i|p_i - C_{i|\Gamma_i|} \geq p_i - C_i$.
- *Les tâches d'acquisition ont une priorité au moins aussi grande que la tâche de traitement.*

La figure 3.8 donne un exemple d'une telle transaction. A partir d'une transaction série, on peut construire une transaction inverse qui consiste en un modèle dans lequel on inverse l'arrivée de la tâche de traitement et des tâches d'acquisition (voir figure 3.9). Nous avons ensuite montré que les enveloppes obtenues, que ce soit en rampes, ou en escaliers, étaient identiques pour une transaction série et sa transition inverse.

Théorème 14 *[Tra07, TGC06b] Soit une transaction série Γ_i, et sa transaction Γ_i^{-1}. Les enveloppes d'interférence sont les mêmes pour la transaction série et sa transaction inverse.*

Il en découle qu'il est inutile, dans ce cas, de mettre en place la méthode des énveloppes pour valider un système de transactions séries. Une transaction inverse est telle que l'enveloppe d'interférence correspond à un instant critique existant : celui initié par la première tâche de la transaction inverse. Il suffit donc de considérer un seul instant critique.

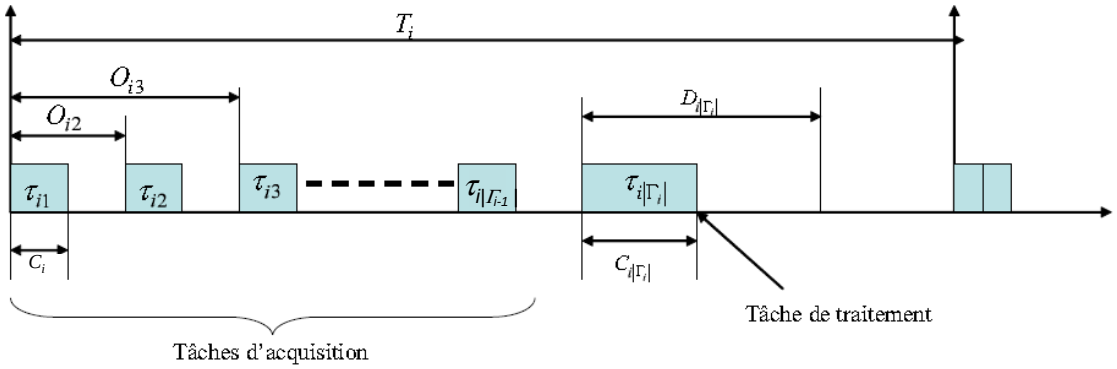

FIGURE 3.8 – Une transaction série.

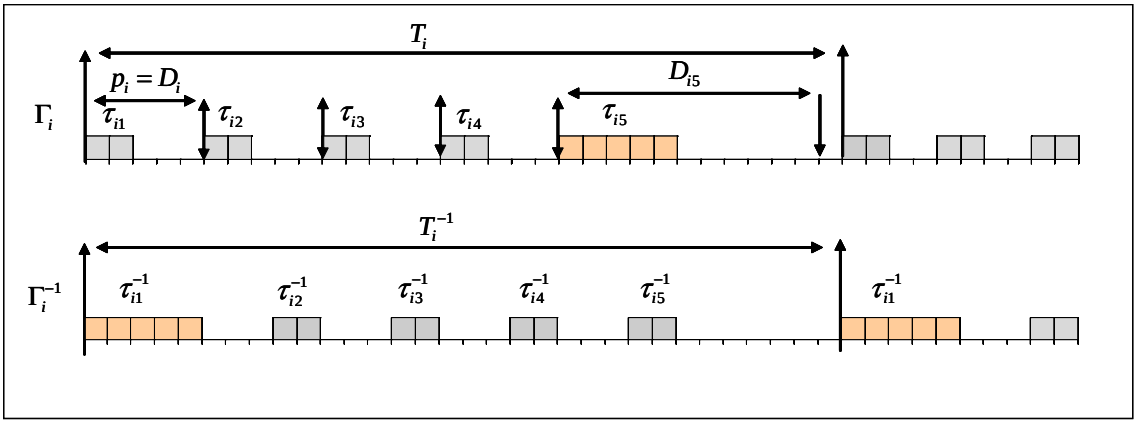

FIGURE 3.9 – Transaction série et sa transaction inverse.

Transactions monotoniques

Nous avons ensuite noté la forme particulière qu'avait l'enveloppe de la transaction inverse : dans ce cas, c'est toujours l'instant critique initié par la tâche de traitement qui correspond à l'enveloppe. L'enveloppe correspond donc à une interférence atteignable en tout point. Nous avons donc exhibé une propriété, condition suffisante (mais non nécessaire) à l'établissement de cette propriété : c'est la propriété de monotonicité [TGRR06, TGC06a]. Cette propriété exprime le fait qu'à des permutations circulaires prêt, on peut trouver une tâche, telle que, si elle est mise en tête de transaction, les WCET des tâches à partir de cette tâche de tête sont ordonnées de façon non croissante, alors que les écarts entre les tâches successives sont non décroissants.

Théorème 15 *[Tra07, TGC06b] L'enveloppe d'une transaction monotonique correspond exactement à l'interférence d'une période d'activité initiée par un instant critique correspondant au réveil de la plus longue tâche.*

A partir de ce résultat, nous avons montré dans [TGC06a] que pour calculer l'interférence d'une transaction monotonique il était nécessaire de considérer un unique cas pour la transaction, ce qui n'entraînait pas de pessimisme. Notons que la transaction série inverse est une transaction monotonique. Par conséquent, dans un système constitué de transactions série, il n'est pas nécessaire d'utiliser la technique des enveloppes, on peut, pour une précision de calcul identique, transformer chaque transaction série en transaction inverse, puis appliquer le calcul de pire temps de réponse en considérant un unique scénario pour chaque transaction inverse, puisqu'elles sont monotoniques.

Nous avons appliqué ce résultat à la validation temporelle du système embarqué sur le drone AMADO.

Transactions accumulativement monotoniques

Nous avons ensuite souhaité caractériser, dans le cadre de la thèse d'Ahmed Rahni, la condition nécessaire et suffisante entraînant le fait que l'interférence d'une transaction pour une tâche candidate particulière soit confondue avec l'enveloppe d'interférence. Cette propriété s'appelle la monotonicité accumulative, et s'exprime très simplement graphiquement (voir figure 3.10). Elle prend en compte la gigue de démarrage, contrairement à la propriété de monotonicité, ce qui la rend utilisable sur tout le modèle des transactions.

Théorème 16 *[Rah08] Si toutes les transactions sont Accumulativement Monotonique (AM) par rapport à la tâche analysée, alors le pire temps de réponse calculé par la méthode des enveloppes est exact.*

Notons que la propriété AM (de même que la propriété de monotonicité) est relative à la priorité de la tâche analysée, en effet les tâches de la transaction considérées dans l'étude peuvent différer, car on ne prend en compte que les tâches de $hp_i(\tau_{ua})$. Afin de détecter l'existence de la propriété AM d'une transaction, nous avons proposé un algorithme se basant sur la méthode de calcul rapide des enveloppes proposée dans [MTN04a]. Nous montrons ici le principe de fonctionnement de l'étude d'un système de transactions avec détection de la propriété de monotonicité accumulative sur un exemple.

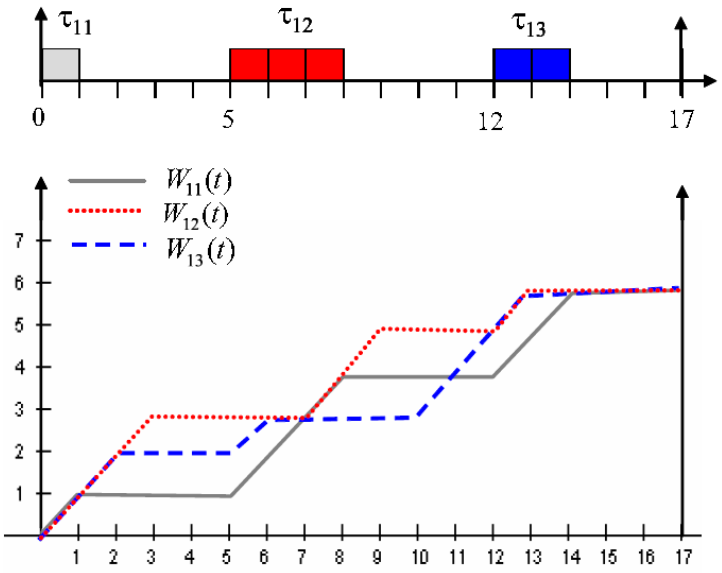

FIGURE 3.10 – Transaction accumulativement monotonique.

Mise en forme normale

Une transaction en forme normale est telle que les tâches de $hp_i(\tau_{ua})$ la composant ne se chevauchent pas (rappelons que les tâches à considérer dans une transaction, sont, pour l'étude d'une tâche analysée τ_{ua}, les tâches plus prioritaires). L'effet obtenu est que la dérivée d'une courbe d'interférence pour un instant critique considéré a une dérivée égale à 1 (voir figure 3.11). L'impact de l'interférence d'une transaction Γ_i sur la période d'activité est le même que celle de la transaction en forme normale Γ_i^* pour le calcul de l'interférence des tâches présentes dans Set_2. Etant donné que la pire interférence des tâches de Set_2 a lieu lorsque leur gigue d'activation est nulle, il n'est pas nécessaire de prendre en compte le paramètre J_{ij}. Le processus suivi pour passer une transaction Γ_i sous sa forme normale Γ_i^* est le suivant. Nous supposons que les indices des tâches sont croissants en fonction de leur *offset* (i.e. $O_{i(j+1)} \geq O_{ij}$).

1. Initialiser Γ_i^* avec les tâches et la période de Γ_i.
2. Pour toute tâche τ_{ij}, si $O_{ij}^* + C_{ij}^* \geq O_{i(j+1)}^*$, alors fusionner $\tau_i j$ et $\tau_i(j+1)$, l'*offset* est alors O_{ij}^* et la charge est $C_{ij}^* + C_{i(j+1)}^*$. Renuméroter les tâches de la transaction par *offset* croissant.
3. Si la dernière tâche peut déborder après la période sur la première, i.e. $O_{i|\Gamma_i^*|}^* + C_{i|\Gamma_i^*|}^* \geq T_i + O_{i1}^*$ alors fusionner τ_{i1} dans $\tau_{i|\Gamma_i^*|}$, c'est-à-dire augmenter $C_{i|\Gamma_i^*|}$ de C_{i1}. Renuméroter les tâches et recommencer l'étape 2.

Nous avons montré que les propriétés de dominance, monotonicité accumulative et de monotonicité était préservées par la forme normale. Il faut cependant remarquer que la première instance de la transaction mise en forme normale peut différer de sa seconde instance lorsque nous avons fusionné lors de l'étape 3 la dernière tâche et la première. Ainsi il faudra, lorsqu'on travaille avec des transactions en forme normale, considérer deux périodes.

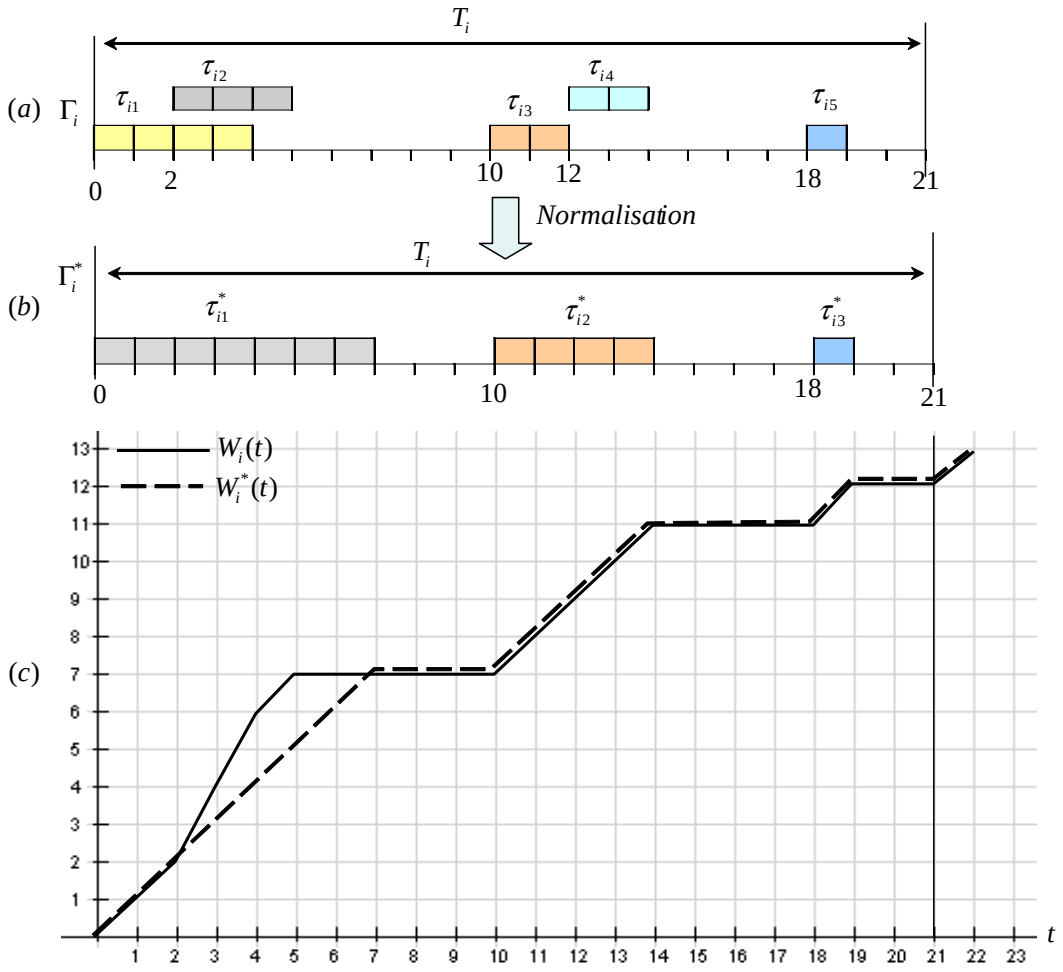

FIGURE 3.11 – (a) Une transaction quelconque. (b) La transaction mise en forme normale. (c) Interférence avant et après normalisation

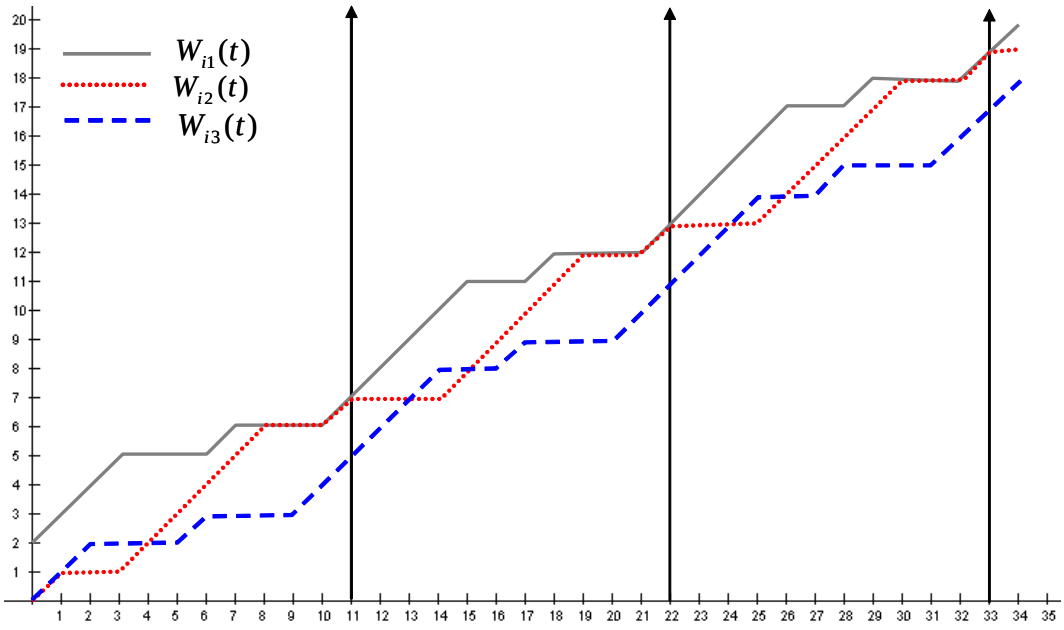

FIGURE 3.12 – L'enveloppe en rampes de l'interférence est périodique à partir de la seconde période.

Pré-calcul de l'enveloppe d'interférence d'une transaction

L'interférence d'une transaction Γ_i lorsque une tâche τ_{ic} initie l'instant critique est représentée statiquement dans deux tables (P_{ic} et P'_{ic}), l'une pour la première période et l'autre pour la seconde période de l'interférence. En effet, l'interférence d'une transaction est périodique à partir de la seconde période, ainsi soit une transaction Γ_i de période $T_i = 11$ et composée de 3 tâches indépendantes :

Tâche	O_{ij}	C_{ij}	J_{ij}	T_i
τ_{i1}	0	3	0	
τ_{i2}	6	1	1	11
τ_{i3}	10	2	2	

La mise en forme normale a fusionné la tâche τ_{i1} de la seconde période avec la tâche τ_{i3} car cette dernière déborde. La figure 3.12 montre le motif périodique à compter de la seconde période des interférences possibles des tâches de Set_2 de la transaction. L'interférence liée à l'ensemble Set_1 des tâches activées avant l'instant critique mais retardées à cause de leur gigue jusqu'à l'instant critique est quant à elle statique. L'idée de l'algorithme est de tirer parti de cette périodicité en stockant statiquement les courbes d'interférence sous forme de ses points caractéristiques (ils sont appelés points convexes et correspondent aux points situés en haut de chaque rampe) dans des tables. Ces tables serviront à calculer la période d'activité de la tâche étudiée avec la méthode des enveloppes en rampes, et à détecter les propriétés de dominance et de monotonicité accumulative.

Turja et Nolin ont proposé cette méthode, nous l'avons améliorée pour permettre la détection des propriétés que nous utilisons. Chaque point de ces tables contient un triplet $< x, y, c >$, représentant le fait que, pour une période d'activité de longueur x, une interférence totale égale à y est imposée par l'ensemble Set_2 de la transaction considérée

Γ_i lorsque la tâche τ_{ic} initie l'instant critique. Ce mémoire ne présente pas le détail des formules afin de ne pas avoir à introduire un nombre important de notations. Nous avons cependant exprimé les propriétés de dominance et de monotonicité accumulative sur ces tables, de façon à pouvoir les détecter de façon efficace. Par exemple les tables correspondant à la figure 3.12 contiennent chacune 5 points initialement pour chacune des courbes, et la courbe enveloppe, qui correspond à $W_{i1}(t)$ contient elle aussi 5 points. Le processus de vérification de propriétés s'effectue alors simplement en vérifiant les indices c des points $< x, y, c >$ de l'enveloppe obtenue.

Méthode mixte

Nous avons proposé une méthode mixte [RGR07a, RGR07b], permettant d'améliorer la technique des enveloppes. L'idée est de proposer de fixer le nombre de transactions pour lesquelles nous souhaitons mener une analyse exacte, alors que pour les autres transactions nous utilisons la technique des enveloppes. Nous avons montré que la méthode mixte était meilleure que la méthode des enveloppes. Bien entendu, plus le nombre fixé est important, plus le nombre d'instants critiques candidats exploré est important. Afin d'éviter toute exploration d'instant critique que l'on pourrait éviter d'explorer, nous avons exprimé une propriété locale aux tâches d'une transaction : la dominance.

Une tâche est dominée par une autre tâche, qu'on appelle dominante, si la courbe d'interférence générée par un instant critique initié par la dominée est toujours majorée par la courbe d'interférence générée par la dominante. Bien entendu, lorsqu'une tâche est dominée par une autre, il est inutile d'étudier la période d'activité initiée par la tâche dominée, ce qui permet d'éliminer des instants critiques candidats. La propriété AM correspond au cas particulier où une tâche domine toutes les autres, et la propriété de monotonicité est un cas particulier d'AM.

Expérimentations quantifiant les apports des propriétés et méthodes

Nous avons implémenté 3 versions de l'algorithme de la méthode mixte, nommées NM_1 (on explore tous les instants critiques possibles pour 1 transaction et on utilise la technique des enveloppes pour les autres), NM_2 (idem sauf qu'on explore tous les instants critiques pour 2 transactions) et NM_3 (idem pour 3 transactions). Bien entendu, nous prenons en compte la propriété de dominance afin de diminuer le nombre de cas explorés. Nous avons mené ces expérimentations afin de déterminer un ratio amélioration du temps de réponse calculé / coût. La figure 3.13 montre ce phénomène, aussi bien lorsqu'on augmente le nombre de tâches que lorsqu'on augmente le nombre de transactions. Nous avons constaté que le temps de calcul nécessaire à la méthode NM_1 était quasiment identique au temps de calcul nécessaire à la méthode des enveloppes en rampes. De plus, l'apport en terme de qualité du résultat fourni est sensible : ainsi par exemple, pour des systèmes de 6 transactions de 12 tâches chacune, la méthode des enveloppes en rampes présente du pessimisme pour le calcul des temps de réponse pour 21% des tâches, alors que NM_1 n'est pessimiste que pour 7,5% des tâches (voir figure 3.14). La méthode NM_2 quant à elle côute un peu plus : par exemple sur des systèmes de 10 transactions de 30 tâches chacune, elle coûte 1 seconde de plus (soit 25%) que la méthode NM_1 et la méthode des enveloppes en rampe. NM_2 améliore la qualité de NM_1 de façon sensible, ainsi, pour 6 transactions de 12 tâches chacune, NM_2 n'est pessimiste que pour 4% des tâches. NM_3 fait tomber ce pessimisme à 1% des tâches, cependant, le temps de calcul augmente très

rapidement avec la taille de l'entrée. Nous avons conclu que NM_2 fournissait le meilleur rapport qualité/coût.

Nous avons mené de nombreuses campagnes d'expérimentation afin de quantifier l'apport en qualité du test, mais aussi en terme de gain de temps, en faisant varier nombre de tâches et de transactions. Comme nous l'avions fait pour la propriété AM, nous avons proposé un algorithme efficace, inspiré du précédent, utilisé pour détecter les tâches dominantes/dominées. Nous avons montré que la propriété de dominance était très fréquente sur des configurations de transactions générées aléatoirement : en moyenne, 30% des tâches sont dominées. Ainsi le gain en terme d'instants critiques explorés a été evalué expérimentalement à 50% grâce à l'utilisation de la dominance, alors qu'il n'est que de 20% pour la propriété AM.

3.2.3 Validation exacte de transactions pour EDF

[PH03] propose une méthode exacte exponentielle de validation des transactions sous EDF : celle-ci étudie tous les instants critiques possibles comme dans le cas FPP, en utilisant la technique proposée par Spuri dans [Spu96] pour chaque scénario d'instant critique. Le même article propose alors une méthode approchée de calcul de pire temps de réponse en utilisant les enveloppes, de façon assez proche de celle que nous avons présentée pour les algorithmes FPP.

Nous avons proposé une méthode exacte pseudo-polynomiale de validation des transactions monoprocesseurs ordonnancées par EDF. Nous avons utilisé pour cela la fonction de demande processeur (DBF).

Principe de la méthode

Rappelons que d'après le théorème 6, un système monoprocesseur non concret est ordonnançable par EDF ssi la fonction $dbf(0,d) \leq d$ pour toute échéance d rencontrée sur la période d'activité synchrone du système. Nous avons étendu ce résultat au cas des tâches avec gigue d'activation en considérant la fonction de demande processeur $dbf(0,t)$ comme la charge cumulée des tâches τ_i ayant une activation dans $[-J_i, t[$ et dont l'échéance arrive avant t. Nous avons étendu cette définition au cas des instances des tâches des transactions. Nommons $df(t) = dbf(0,t)$.

Pour un scénario d'instant critique donné, correspondant à l'activation d'un candidat τ_{ic} dans chaque transaction Γ_i, notons $df_{ijc}(t)$ la charge cumulée générée par la tâche τ_{ij} étant réveillée et devant être terminée dans l'intervalle $[-J_i, t[$. Ainsi, la charge cumulée générée par la transaction dans cet intervelle pour le scénario où l'instant critique correspond à une activation de la tâche τ_{ic} sera $df_{ic}(t) = \sum_{j=1..|\Gamma_i|} df_{ijc}(t)$. Il est alors possible de calculer la dbf du système, pour un scénario donné correspondant à l'activation simultanée de τ_{1c_1}, τ_{2c_1}, ..., $\tau_{ic_{|\Gamma|}}$, la df globale $df_{c_1,c_2,...,c_{|\Gamma|}}(t) = \sum_{i=1..|\Gamma|} df_{ic}(t)$.

Nous avons montré dans [Rah08] le théorème suivant :

Théorème 17 *[Rah08, RGR08] Un système de transactions est ordonnançable ssi pour tout scénario d'instant critique possible τ_{1c_1}, τ_{2c_1}, ..., $\tau_{ic_{|\Gamma|}}$, $\forall t$,*

$$df_{c_1,c_2,...,c_{|\Gamma|}}(t) = \sum_{i=1..|\Gamma|} df_{ic}(t) \leq t$$

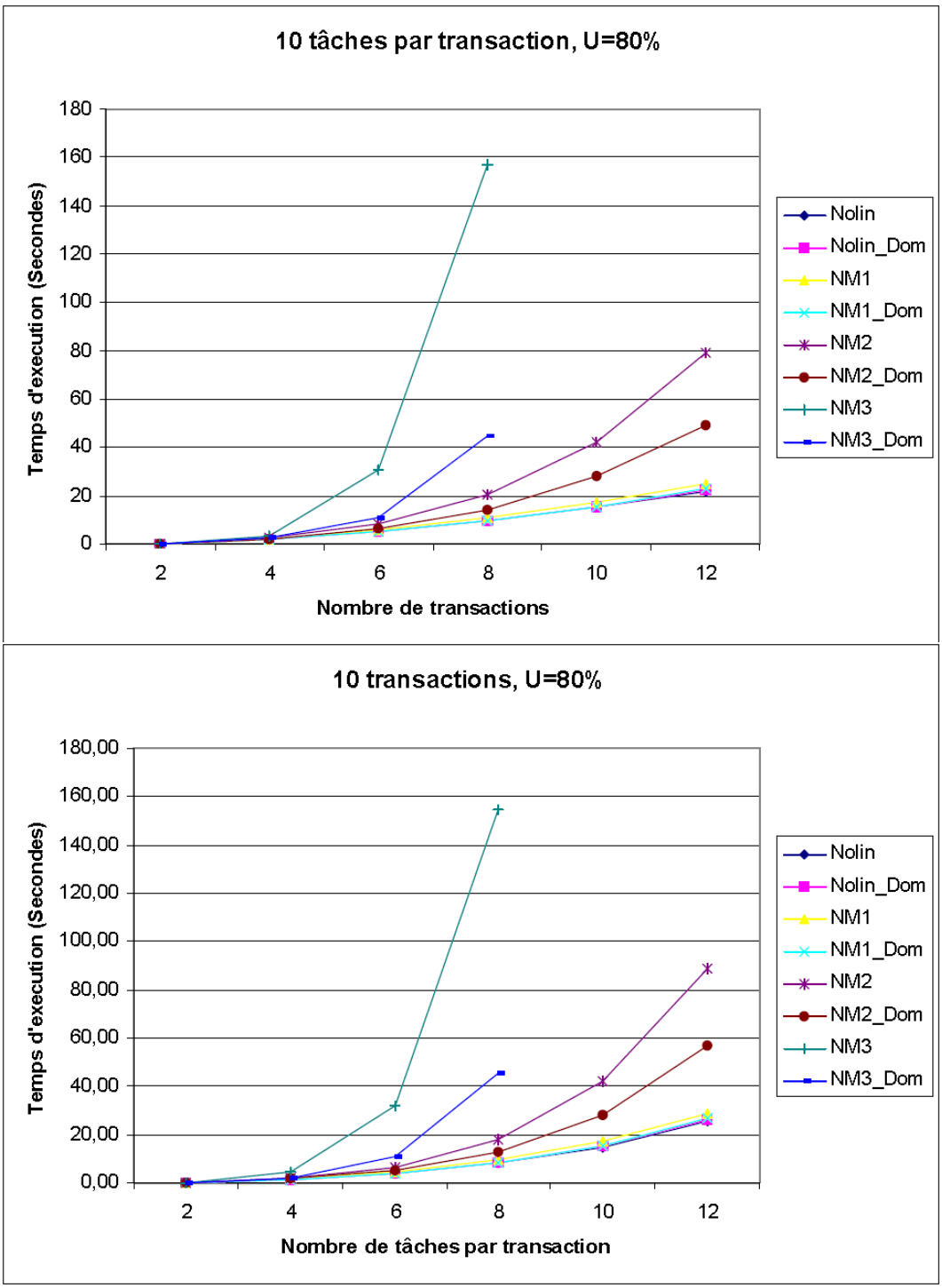

FIGURE 3.13 – Comparaison des temps d'exécutions des méthodes de calcul de temps de réponse utilisant ou non la propriété de dominance en fonction du nombre de transactions puis du nombre de tâches par transaction.

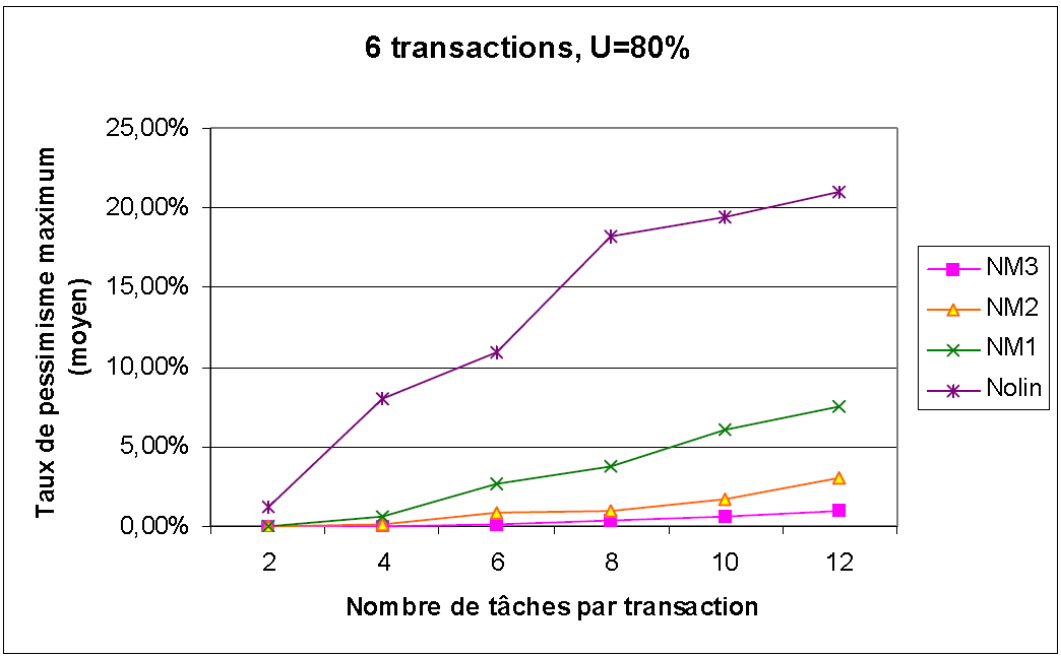

FIGURE 3.14 – Taux de pessimisme maximal (Temps de réponse obtenu par une méthode moins Temps de réponse exact divisé par le temps de réponse exact) observé, représenté en moyenne sur les systèmes étudiés.

.

Cela représente un nombre exponentiel de scénarios, comme c'est le cas en FPP. Cependant, la *df* présente une propriété très intéressante par rapport à l'interférence des tâches en priorités fixes qui nous a permis de démontrer le théorème suivant.

Théorème 18 *[Rah08, RGR08] Soit* $df_i(t) = \max_{c=1..|\Gamma_i|} df_{ic}(t)$. *Un système de transactions est ordonnançable ssi* $\forall t, \sum_{i=1..|\Gamma|} df_i(t) \leq t$.

Le fait de considérer la valeur maximale de la *dbf* donne une condition nécessaire et suffisante d'ordonnançabilité. On peut donc utiliser une enveloppe représentant la valeur maximale de toutes les *dbf* correspondant aux différents candidats aux instants critiques pour chaque transaction, puis combiner les enveloppes pour obtenir une condition nécessaire et suffisante d'ordonnançabilité. Rappelons que la technique des enveloppes d'interférence donne seulement une condition suffisante en FPP.

Il nous reste à déterminer la durée d'étude nécessaire : pour chaque scénario il s'agit de la période d'activité initiée par un instant critique, mais il serait dommage de construire chaque scénario. Cependant, nous avons vu dans le cas FPP que la méthode des enveloppes permettait d'obtenir une borne supérieure de la longueur de la période d'activité. Il suffit donc, pour obtenir une borne supérieure de la longueur des périodes d'activité de chaque scénario, de considérer la borne obtenue à l'aide de la méthode des enveloppes. La borne la meilleure étant obtenue par les enveloppes en rampes, nous utilisons cette technique pour obtenir une borne supérieure L^* de la longueur des périodes d'activité pour tout scénario.

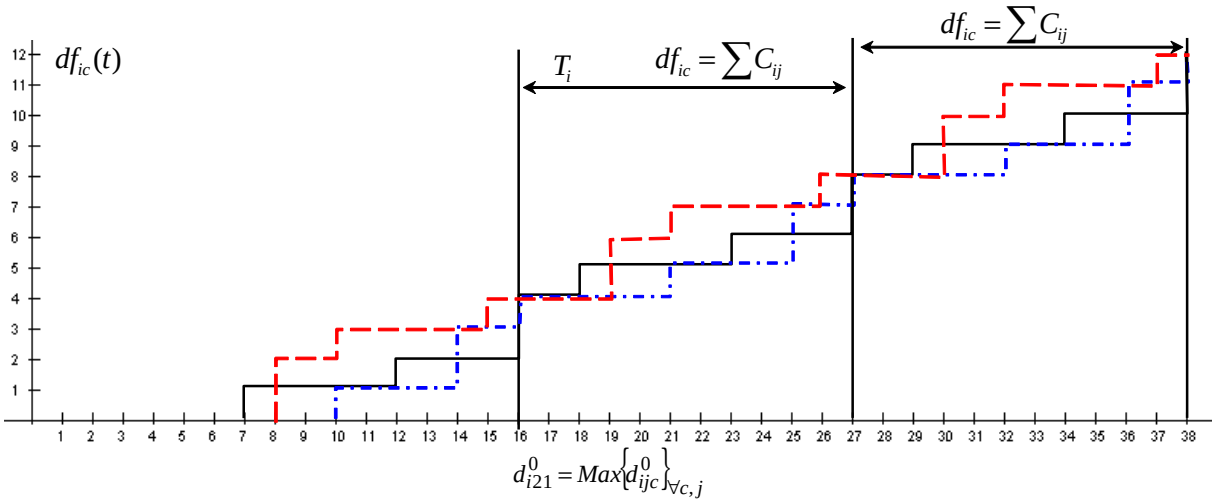

FIGURE 3.15 – Arrivée périodique de la demande processeur après la date d^0_{ic}.

Théorème 19 *[Rah08, RGR08] Soit $df_i(t) = \max_{c=1..|\Gamma_i|} df_{ic}(t)$. Un système de transactions est ordonnançable ssi $\forall t \leq L^*, \sum_{i=1..|\Gamma|} df_i(t) \leq t$.*

Méthode accélérée

Nous avons proposé une implémentation efficace du calcul de la *dbf* pour les transactions, utilisant la représentation statique des points convexes de la courbe de *dbf* exploitant la périodicité de celle-ci, en adaptant la technique proposée dans le cas FPP. Nous présentons cet algorithme ici.

Périodicité de la dbf

Dans un premier temps, nous identifions la périodicité de la fonction de la demande df_{ijc} générée par la tâche τ_{ij} lors d'un instant critique initié par une tâche candidate τ_{ic}.

Notons d^0_{ijc} la date de la première échéance d'une instance de τ_{ij} après l'instant critique initié par τ_{ic}. A partir de cette date, la tâche τ_{ij} génère une demande périodique de valeur C_{ij} toutes le T_i unités de temps. Soit la date de la première échéance la plus tardive pour les tâches de la transaction :

$$d^{max}_{ic} = Max_{\forall \tau_{ij}} d^0_{ijc}$$

A partir de d^{max}_{ic}, la demande processeur augmente de façon périodique avec une période T_i, comme cela est illustré sur la figure 3.15. Puisque nous nous intéressons à l'enveloppe des courbes $df_i(t) = \max_{c=1..|\Gamma_i|} df_{ic}(t)$, nous considérons pour tout candidat la date maximale à laquelle la demande processeur générée devient périodique de période T_i.

$$d^0_{max} = Max_{\forall c} d^{max}_{ic}$$

Utilisation d'une table de représentation statique

La table créée pour chaque candidat τ_{ic} dans chaque transaction Γ_i est P_{ic}. Cette table est une liste de couples (d, C) avec d une date d'échéance et C une charge additionnelle à traiter à cette date. Elle contiendra les charges à ajouter à la fonction df_{ic} à la date d

relativement à l'instant critique. Nous notons $P_{ic}[n].d$ et $Pic[n].C$ respectivement la $n^{\text{ième}}$ échéance (les points sont ordonnés par date) rencontrée à partir de l'instant critique initié par τ_{ic}, et la charge ajoutée à la dbf par les tâches de la transaction Γ_i en ce point. Etant donné que nous voulons profiter de la périodicité de cette fonction à partir de la date d^0_{max}, nous construisons la table jusqu'à atteindre la date $d^0_{max} + T_i$.

Ainsi, les 3 tables de représentation statique de la dbf correspondant aux 3 candidats à l'instant critique sur la figure 3.15 sont données par :

$$P_{i1} =< (7,1),(12,2),(16,4),(18,5),(23,6),(27,8) >$$

$$P_{i2} =< (8,2),(10,3),(1,4),(19,6),(21,7),(26,8) >$$

$$P_{i3} =< (10,1),(14,3),(16,4),(21,5),(25,7),(27,8) >$$

La table P_i représentant l'enveloppe des dbf est alors obtenue de la façon suivante. Initialement :

$$P_i = \bigcup_{\forall c} P_i c$$

où $\bigcup$ est l'union triée par le champ d des tables. Puis, afin de déterminer les points à conserver pour représenter l'enveloppe des dbf correspondant aux instants critiques candidats, nous définissons la relation "envelopper". Un point $P_i[a]$ enveloppe un point $P_i[b]$, noté $P_i[a] \succ P_i[b]$ si la présence de $P_i[a]$ implique que $P_i[b]$ n'est pas un point convexe de la dbf maximale. Cela s'exprime par :

$$P_i[a] \succ P_i[b] \; ssi \; (P_i[a].d \leq P_i[b].d \wedge P_i[a].C \geq P_i[b].C)$$

La relation "envelopper" utilise le fait que l'union des tables est triée suivant le champ d.

Sur l'exemple de la figure 3.15, la table P_i est donnée par :

$$P_i =< (7,1),(8,2),(10,3),(15,4),(18,5),(19,6),(21,7),(26,8) >$$

A partir de la table P_i, il est simple de calculer la demande processeur générée par les tâches de la transaction Γ_i :

$$Si \; t \leq d^0_{max} + T_i, df_i(t) = P_i[n].C \; avec \; n = max\{m : P_i[m].d \leq t\}$$

Si $t > d^0_{max} + T_i$, soit $m = \left\lfloor \frac{t - d^0_{max}}{T_i} \right\rfloor$ et $t' = t - m \times T_i$:

$$df_i(t) = df_i(t') + m \times \sum_{\forall \tau_{ij}} C_{ij}$$

Afin de valider un système, nous sommes donc amenés à construire pour chaque tâche la table P_{ic}, puis pour chaque transaction faire l'union des tables, puis utiliser la fonction "envelopper" pour ne conserver que les points convexes. Ces tables sont alors utilisées pour obtenir immédiatement la valeur de la demande processeur, qui doit toujours être $\leq t$ pendant l'étude des échéances présentes durant la période d'activité obtenue.

Méthode approchée en FPP vs. méthode exacte pour EDF

Ce résultat peut être troublant au premier abord : pourquoi en combinant les fonctions d'interférence en FPP obtient-on une condition suffisante d'ordonnançabilité, alors qu'en combinant les fonctions de demande processeur sous EDF une condition nécessaire et suffisante est obtenue ? L'explication réside dans le fait suivant : dans les deux cas, la longueur de la période d'activité en considérant différents scénarios varie. Dans le cas FPP, on utilise la longueur de la période d'activité pour établir le temps de réponse, et on a besoin de considérer chaque scénario pour connaître cette longueur. Dans le cas EDF, on utilise la longueur de la période d'activité comme durée pendant laquelle on vérifie le respect d'échéance : cette durée est une borne supérieure, mais nous n'avons cependant qu'une seule période d'activité à explorer.

Unification avec les modèles multiframe

Parallèlement aux travaux concernant les transactions, divers auteurs ont travaillé sur le modèle multiframe [MC96, MD97]. Ce modèle a été créé initialement afin de représenter des tâches dont les instances avaient des durées différentes, mais il permet indirectement de prendre en compte des décalages entre certaines tâches. Un tâche multiframe est définie comme une tâche classique dans le contexte $|0, C_i, D_i = T_i, T_i|$ excepté que chaque tâche possède une suite de durée $(C_{i1}, C_{i2}, ..., C_{in_i})$: la première instance a un WCET de C_{i1}, la second C_{i2} etc. et la $n_i + 1^{\text{ème}}$ a à nouveau un WCET C_{i1} et ainsi de suite. On peut montrer trivialement qu'une tâche multiframe peut être modélisée par une transaction contenant n_i tâches [Tra07]. Initialement, les travaux effectués sur le modèle multiframe ont proposé des conditions suffisantes d'ordonnançabilité basées sur la charge pour ce modèle, puis [BCM99] a proposé une méthode identique à celle que Palencia et Harbour avaient proposé sur le modèle plus général des transactions un an plus tôt. [BCGM99] a introduit le modèle multiframe généralisé, rapprochant un peu plus le modèle multiframe du modèle des transactions (une tâche multiframe généralisée est une transaction avec les contraintes suivantes : échéance contrainte, pas de gigue, *offset* $\leq$ à la période de la tâche). Différentes études parallèles sur la RTA, et la dbf ont été menées sur le modèle multiframe généralisé et les transactions, chaque modèle étant plus avancé dans un domaine que l'autre : ainsi, avantage est donné aux transactions pour les approches de type analyse de temps de réponse, et dbf, tandis que de plus nombreuses propriétés assez proches de la dominance ont été étudiées sur les modèles multiframe. Nous avons montré dans [Tra07] que le modèle des transactions est une généralisation du modèle multiframe généralisé. Par conséquent tous les travaux que nous avons réalisés dans le cas des transactions est applicable au modèle multiframe, et nos algorithmes sont, à ma connaissance, les plus avancés dans le domaine, aussi bien dans le cas FPP que pour l'algorithme EDF.

3.2.4 Publications

Ces études sur le modèle des transactions ont débuté pendant la thèse de Karim Traoré [Tra07], et ont été poursuivies pendant la thèse d'Ahmed Rahni [Rah08]. La principale publication est [RGR09], et un article de synthèse est en cours de modification avant soumission en revue. Les autres publications sur le sujet sont [TGC06a, TGRR06, TGC06b, RGR07b, RGR07a, RGR08].

3.2.5 Points forts de la démarche

Nous avons unifié deux modèles de façon à pouvoir utiliser les résultats des transactions aux modèles multiframes, qui ont été plus étudiés dans la littérature, mais ne possédaient pas une approche RTA aussi aboutie que celle de Turja et nolin. Nous avons proposé un test exact de complexité pseudo-polynomiale pour EDF. Ces études ont été menées sur un modèle dont nous avons montré par l'étude de cas du drone AMADO qu'il était très utile sur un grand nombre d'applications réelles.

Du côté FPP, nous avons étudié expérimentalement les apports quantitatifs de toutes les propriétés que nous avons proposées pour diminuer les scénarios étudiés. Nous avons montré expérimentalement que notre méthode mixte améliore de façon sensible la méthode de base.

3.3 Contraintes de précédence

Nombre de systèmes temps réel utilisent des communications synchrones faiblement couplées (i.e. synchronisation par sémaphore privé, boîtes aux lettres). Par exemple, supposons deux tâches communicantes telles que $\tau_1 \prec \tau_2$. Un pseudo-code possible est :

Tâche τ_1

```
Faire toujours
    contenu fonctionnel
    déclenchement de la synchronisation (par exemple vendre(sémaphore))
    attendre la prochaine activation (par exemple réveil périodique)
Fait
```

Tâche τ_2

```
Faire toujours
    attente sur synchronisation (par exemple prendre(sémaphore)
    contenu fonctionnel
Fait
```

Cet exemple montre que sur toute une famille de cas, les tâches se synchronisant ont la même période : ici τ_2 hérite du rythme de τ_1. Dans ce cas, on parle de précédences simples. Dans le cas où les précédences concernent des tâches de rythme différent mais reliés, on parle de précédences généralisées. Nos travaux concernent les deux familles de systèmes. Une première section présente donc les travaux réalisés dans le cas de précédences simples en présence de primitives de synchronisation, et la seconde section présente nos travaux en cours sur les précédences généralisées.

3.3.1 Précédences simples en présence de primitives de synchronisation

Nous avons proposé une approche de validation temporelle d'ordonnancement FPP dans le contexte de précédences simples en présence de primitives de synchronisation. Cette section présente nos travaux sur ce sujet. Les preuves des résultats n'étant pas fournies dans ce mémoire, les notations ont été simplifiées et rapprochées des notations utilisées auparavant afin de faciliter la lecture.

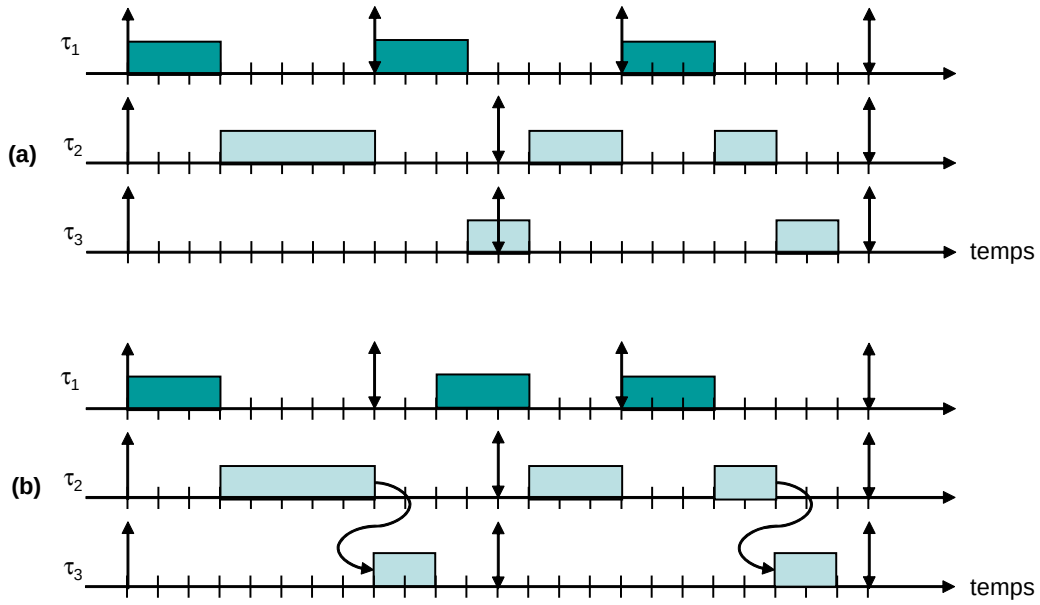

FIGURE 3.16 – (a) Ordonnancement RM non valide d'un système, (b) Ordonnancement FPP valide avec outils de synchronisation.

Puissance d'ordonnancement

Nous remarquons d'abord que la présence de synchronisations augmente la puissance d'ordonnancement des algorithmes FPP. Ainsi, considérons l'exemple suivant :

Tâche	r_i	C_i	D_i	T_i
τ_1	0	3	8	8
τ_2	0	5	12	12
τ_3	0	2	12	12

La figure 3.16, sur la partie **(a)**, montre l'ordonnancement RM, non valide, du système. L'algorithme RM étant optimal dans ce contexte, nous pouvons en déduire que le système n'est pas ordonnançable en FPP. Cependant, la figure (b) montre que si l'on a une synchronisation assurant une précédence $\tau_2 \prec \tau_3$, alors l'affectation de priorité telle que $Prio(\tau_3) > Prio(\tau_1) > Prio(\tau_2)$ permet d'ordonnancer fiablement le système. La présence de synchronisations augmente donc la puissance d'ordonnançabilité des algorithmes FPP.

Anomalies d'ordonnancement

Cependant, des anomalies d'ordonnancement peuvent se produire comme le montre la figure 3.17, tirée de [CR98]. Un système de quatre tâches, avec $\tau_3 \prec \tau_4$ assurée par synchronisation, montre une séquence d'ordonnancement valide dans le cas (a) avec l'affectation de priorités suivante : $Prio(\tau_4) > Prio(\tau_2) > Prio(\tau_1) > Prio(\tau_3)$. Cependant, pendant l'exécution du système avec un ordonnanceur hors-ligne, il est possible, comme dans le cas (b), que la charge de τ_1 soit inférieure à son WCET. Ainsi dans le cas où l'exécution de cette tâche ne dure que 3 unités de temps, la tâche τ_3 peut s'exécuter plus tôt, et ainsi déclencher la tâche τ_4 plus tôt que sur la simulation, celle-ci étant alors responsable du décalage de τ_2 fatal au respect de sa seconde échéance. Ainsi, l'étude des

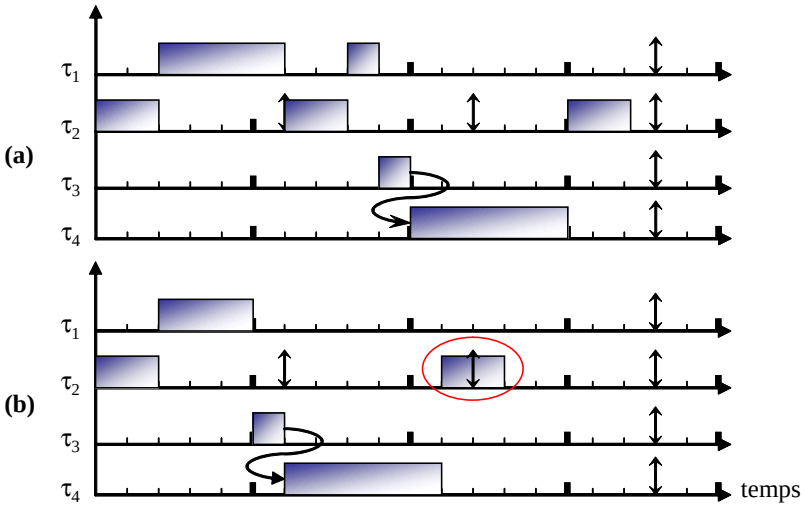

FIGURE 3.17 – (a) Simulation d'ordonnancement valide d'un système avec synchronisation, (b) Ordonnancement possible du même système à l'exécution.

pires temps de réponse des tâches dans un ordonnancement FPP de systèmes de tâches utilisant des synchronisations doivent tenir compte de ces anomalies. [HKL91, HKL94] ont proposé une méthode de calcul exacte mais exponentielle du pire temps de réponse des tâches dans ce contexte, lorsque les précédences ne concernent que des chaînes de tâches. Les mêmes auteurs ont proposé un test pseudo-polynomial approché dans [HKL94]. Nous avons adapté ce test aux graphes de précédence ré-entrants (i.e. l'instance d'une tâche en fin de chaîne de précédence peut interférer avec la ou les prochaines instances de tâches en début de chaîne).

Test général proposé

Le test d'ordonnançabilité consiste à déterminer le pire temps de réponse d'une tâche pour une affectation donnée des priorités. Nous supposons que les échéances sont arbitraires et nous plaçons dans le contexte $|C_i, D_i, T_i|prec$ avec des précédences assurées par outils de synchronisation. Si plusieurs instances d'une même tâche sont prêtes à être exécutées, elles respecteront l'ordre FIFO (non ré-entrance des instances d'une même tâche). La complexité du problème de faisabilité d'une tâche est ouverte et aucune condition nécessaire et suffisante n'est connue à ce jour. Nous détaillons rapidement les différentes étapes du test qui permettent de calculer le pire temps de réponse d'une tâche analysée :

1. *Transformation du graphe de précédence en chaînes :* nous montrons que tout graphe de précédence connexe peut être transformé en une chaîne. La configuration de tâches obtenue est équivalente à la configuration initiale. Ceci permet d'utiliser la démarche de test de [HKL91].
2. *Mise sous forme canonique des priorités de la chaîne étudiée :* cette transformation permettra de calculer précisément les dates de fin des prédécesseurs de la tâche étudiée τ_{ua}.
3. *Regroupement des tâches par rapport à la plus petite priorité ($Prio_{ua}(j)$) parmi les prédécesseurs de la tâche étudiée :* les tâches ayant une priorité supérieure à

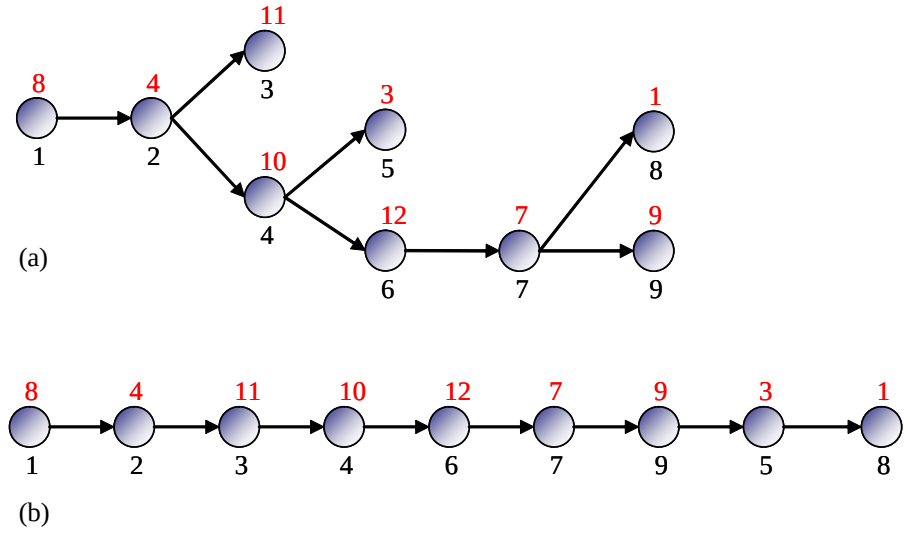

FIGURE 3.18 – (a) Un graphe connexe de précédences acycliques, (b) Sa transformation en chaîne.

$Prio_{ua}(j)$ sont sans influence sur le temps de réponse de la tâche étudiée. Les autres peuvent interférer soit en blocage en début de période d'activité, soit en préemption suivant qu'elles sont ou non précédées par des tâches de priorité supérieure à $Prio_{ua}(j)$.

4. *Calcul de la période d'activité induisant le pire temps de réponse :* la détermination de la plus grande période d'activité caractérise le pire scénario d'arrivées des tâches qui peuvent interférer avec la tâche étudiée. Ceci permet de déterminer le nombre d'instances de cette tâche dans la période d'activité.
5. *Calcul de la date de fin des instances des tâches dans l'ordre de la chaîne :* les dates de fin permettent de calculer le pire temps de réponse de la tâche étudiée.

Il est très important de noter, qu'à l'instar de l'étude dans le cas des transactions, les étapes 2 à 5 sont dépendantes de la tâche analysée τ_{ua}. Elles doivent être complètement refaites pour chaque tâche à valider.

Transformation d'un graphe en chaînes

Nous montrons ci-après que pour une affectation donnée des priorités, le graphe de précédence peut être transformé en chaînes de tâches. La règle suivante transforme chaque composante connexe du graphe de précédence en une chaîne. La règle de transformation est la suivante :
Pour chaque composante connexe du graphe de précédence une chaîne est construite en plaçant à chaque étape la tâche la plus prioritaire parmi les tâches sans prédécesseurs ou dont tous les prédécesseurs ont déjà été placés. Cette transformation est illustrée sur la figure 3.18 où chaque cercle représente une tâche dont l'identifiant est écrit en dessous, et la priorité au dessus.
Le résultat suivant, que nous avons démontré, montre que cette transformation laisse le problème d'ordonnancement inchangé.

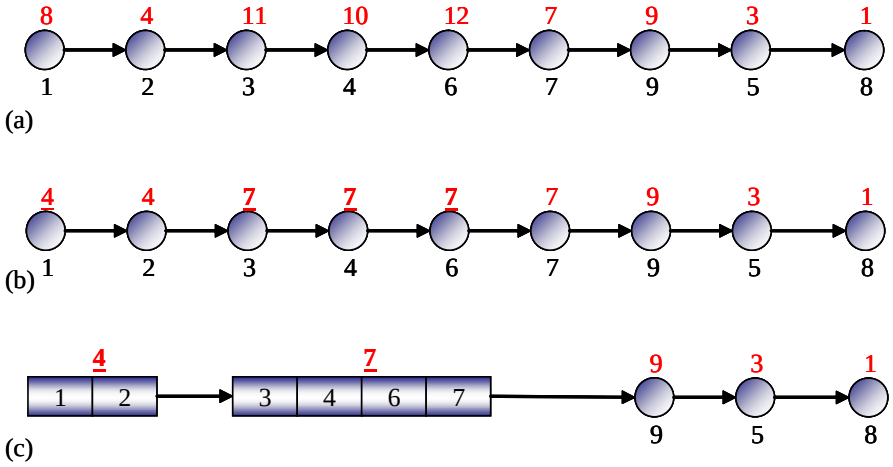

FIGURE 3.19 – (a) Une chaîne de précédences. (b) Sa mise sous forme canonique. (c) Après opération de fusion.

Lemme 1 *[RRGC02] Soit τ un ensemble de tâches, Prio l'affectation des priorités aux tâches (toutes les tâches possèdent une priorité différente), $\prec$ un ordre partiel quelconque sur τ et $\prec'$ les chaînes résultant de la transformation de $\prec$ par la règle de transformation d'un graphe en chaîne alors :*
$(\tau, Prio, \prec)$ est ordonnançable si, et seulement si, $(\tau, Prio, \prec')$ l'est.

Mise sous forme canonique des chaînes de précédence

Une chaîne de précédences mise sous forme canonique (voir figure 3.19(b)) est telle que la priorité des tâches précédant la tâche analysée est non décroissante. Elle est proposée dans [HKL91] où les auteurs montrent que le temps de réponse de la tâche analysée est le même sur la chaîne que sur la chaîne mise sous forme canonique. Cette mise sous forme canonique, puis fusion (voir figure 3.19(b)) a pour effet de simplifier les étapes d'analyse de pire scénario.

Interférence des chaînes de précédence

Le pire temps de réponse de la tâche analysée τ_{ua} survient dans une période d'activité de son niveau de priorité, qui, par définition, ne contient que des tâches de l'ensemble $\{\tau_{ua}\} \cup hp(\tau_{ua})$ (rappelons que l'ensemble $hp(\tau_{ua})$ est l'ensemble des tâches de priorité au moins aussi haute que celle de τ_{ua}). Nous allons donc distinguer deux types de segments lors de l'étude de l'interférence des chaînes de tâches sur τ_{ua} :

1. les segments H (High priority) qui ne contiennent que des tâches de plus forte priorité que $\tau_{ua,1}$, la première tâche de la chaîne contenant τ_{ua}.
2. les segments L (Low priority) qui ne contiennent que des tâches de plus faible priorité que $\tau_{ua,1}$.

Puisque nous autorisons la réentrance des chaînes de tâches, l'interférence des segments les uns sur les autres sera différente du cas non ré-entrant considéré dans [HKL91].

Interférence des chaînes de tâches différentes de celle contenant τ_{ua} sur la tâche étudiée τ_{ua} : Un segment L ne peut pas s'exécuter dans une période d'activité et n'a en conséquence aucune influence sur le pire temps de réponse de τ_{ua}. Un segment H, n'appartenant pas à la chaîne étudiée peut, quant à lui, provoquer deux effets différents sur τ_{ua} suivant qu'un segment H est précédé ou précède un segment L :

- *effet de blocage :* lorsqu'un segment H est précédé par un segment L, le segment H ne peut intervenir dans la période d'activité que si le segment L s'est terminé juste avant le début de celle-ci. Nous avons montré que parmi ces segments H, un seul pourra interférer dans la période d'activité. Ces segments H sont notés $\mathbf{H}_{sb}$ (i.e. singly blocking).
- *effet de préemption :* Un segment H non précédé par un segment L (mais qui peut en précéder un) peut s'activer plusieurs fois puisque les chaînes sont réentrantes. On parle alors de segment à préemption multiple, noté $\mathbf{H}_{mp}$ (multiply preemptive segment).

Interférence des segments de la chaîne contenant τ_{ua} sur la tâche τ_{ua} : Nous décrivons ci-dessous les interférences induites par ces différents segments sur l'exécution de la chaîne elle-même. Elle peut peut subir trois types d'effet :

- *effet de préemption :* Les tâches appartenant au premier segment (i.e. de $\tau_{ua,1}$ à τ_{ua}) ont des priorités non décroissantes puisqu'elles sont sous la forme canonique. Elles peuvent intervenir plusieurs fois dans la période d'activité étudiée. En conséquence, lors de l'estimation de cette dernière, l'enchaînement des tâches $\tau_{ua,1}, \ldots, \tau_{ua}$ est considéré comme un segment H_{mp}. De plus, s'il existe un segment H adjacent à τ_{ua}, ce dernier sera inclus dans le segment H_{mp}.
- *effet de blocage multiple :* Nous avons montré qu'un segment H suivant immédiatement τ_{ua} ne peut pas introduire d'effet de préemption sur les tâches $\tau_{ua,1} \ldots \tau_{ua}$, mais peut s'exécuter plusieurs fois dans la période d'activité. On parle alors de segment à blocage multiple, nommé ensuite $\mathbf{H}_{mb}$. Ce blocage est dû au fait que la chaîne τ_{ua} est ré-entrante. Ainsi, si k instances de τ_{ua} se sont exécutées dans la période d'activité, il peut se produire $(k-1)$ blocages induits par ce segment H_{mb}.
- *effet de blocage :* Un segment H de la chaîne précédé par un segment L peut provoquer *une fois* un effet de blocage et sera en compétition avec les segments H des autres chaînes. Nous parlons alors de segments H à blocage simple ou segment H_{sb} (singly blocking).

Période d'activité et temps de réponse

Une tâche est fiablement ordonnancée si son pire temps de réponse n'est pas plus grand que son délai critique. Lorsque les tâches sont indépendantes, nous avons vu que le pire temps de réponse d'une tâche survient lorsque la tâche est réveillée en même temps que toutes les tâches plus prioritaires (i.e., l'instant critique) [LL73, Leh90, JP86]. Cependant, les possibilités d'anomalies d'ordonnancement montrent que la notion d'instant critique ne permet pas de déterminer le pire temps de réponse d'une tâche lorsque les tâches sont soumises à des contraintes de synchronisation. Nous avons montré le résultat suivant permettant d'élaborer le pire scénario :

Théorème 20 *L'interférence maximale que peut subir τ_{ua} dans une période d'activité de niveau débutant à la date t_0, survient dans le scénario suivant :*

- *(a) sa chaîne de précédence est activée à la date t_0.*
- *(b) une instance de chaque tâche débutant par un segment H_{mp} arrive à la date t_0.*
- *(c) une instance du plus long segment H_{sb}, choisie parmi toutes les chaînes, arrive à la date t_0.*

Le calcul du pire temps de réponse de la tâche τ_{ua}, se décompose en trois étapes :

1. le calcul du nombre d'activations de la chaîne contenant τ_{ua} dans la période d'activité qui permet la prise en compte de l'interférence de l'éventuel segment H_{mb}.
2. le calcul de la date de fin de la première tâche de la chaîne (i.e. $\tau_{ua,1}$).
3. le calcul des dates de fin des tâches $\tau_{ua,2}, \ldots, \tau_{ua}$. Ce calcul prend en compte les dates de fin du prédécesseur immédiat.

Nombre d'activations de la chaîne contenant τ_{ua} dans la période d'activité : La durée d'un segment h_i de type H est notée E_i et est égale à la somme des durées d'exécution des tâches composant ce segment. La période d'un tel segment est héritée de la chaîne le contenant et est notée T_i. Dans ce calcul, deux ensembles sont à considérer :

MP (*multiple preemption*) : c'est l'ensemble des segments de type H_{mp}. Plus précisément, il regroupe les segments de type H_{mp} de toutes les chaînes différentes de celle contenant τ_{ua}, ainsi que le segment H_{mp} de la chaîne contenant τ_{ua}. Ce dernier est composé, au minimum des tâches $\tau_{ua,1}, \ldots, \tau_{ua}$ auxquelles peut se rajouter une suite de tâches $\tau_{ua,j+1}, \ldots, \tau_{ua,l}$ formant le segment H_{mb}, s'il existe.

SB (*Singly Blocking*) : Tous les segments H précédés par un segment L (i.e. H_{sb}) sont regroupés dans l'ensemble SB.

Une chaîne peut bien sûr avoir des segments simultanément dans les ensembles MP et SB. La charge des tâches de $hp(\tau_{ua}) \cup \{\tau_{ua}\}$ dans l'intervalle $[0, t)$ est donnée par le plus petit point fixe de :

$$W_i(t) = \max_{h_i \in SB}(E_i) + \sum_{h_i \in MP} \left\lceil \frac{W_i(t)}{T_i} \right\rceil E_i$$

Le premier terme désigne le plus grand segment H_{sb}, tandis que le second calcule la charge cumulée des segments H_{mp}. La période d'activité est calculée comme le plus petit point fixe L_{ua} du système suivant :

$$\begin{cases} L_{ua}^{(0)} & = & \sum_{k=1}^{\text{position de } \tau_{ua} \text{ dans la chaîne}} C_{ua,k} \\ L_{ua}^{(n+1)} & = & W\left(L_{ua}^{(n)}\right) \end{cases}$$
$$L_i = \min\left(n \geq 0,\ L_i^{(n+1)} = L_i^{(n)}\right)$$

Le nombre d'activations N_{ua} de τ_{ua} dans cette période d'activité est :

$$N_{ua} = \left\lceil \frac{L_{ua}}{T_{ua}} \right\rceil$$

Calcul du pire temps de réponse de τ_{ua}

Le nombre d'activations de τ_{ua} dans la période d'activité est maintenant connu. La dernière opération consiste à calculer toutes les dates de fin des instances de τ_{ua} dans cette période d'activité. Ceci permet de calculer le temps de réponse de chaque instance et déterminer ainsi le pire temps de réponse de τ_{ua}. Les dates de fin de τ_{ua} dépendent directement des dates de fin de son prédécesseur immédiat dans la chaîne contenant cette tâche. Nous calculons donc les dates de fin des tâches dans l'ordre topologique de la chaîne (i.e., $\tau_{ua,1} \dots \tau_{ua}$). Le calcul des dates de fin de $\tau_{ua,1}$ va se différencier des autres puisqu'il doit tenir compte (et lui seulement) du segment H_{mp} de la chaîne étudiée.

Dans la suite $F_{ua}(k)$ désigne la date de fin de la $k^{ième}$ instance de τ_{ua} dans la période d'activité.

Calcul des dates de fin de $\tau_{ua}(k)$: La date de fin de $\tau_{ua}(k)$ dépend :

- du blocage lié au plus long segment de l'ensemble SB,
- de l'ensemble MP_1 contenant les segments H_{mp} (formés des tâches plus prioritaires que τ_{ua}), n'appartenant pas à la même chaîne de tâches.
- du premier segment H de la chaîne contenant τ_{ua} (segments H_{mp} et H_{mb} s'il existe). La durée de ce segment est notée $E_{ua,1}$.

La charge induite par la chaîne contenant τ_{ua} jusqu'à l'instance k de $\tau_{ua,1}$ est donnée par :

$$(k-1)E_{ua,1} + C_{ua,1}$$

La charge des tâches de priorité supérieure ou égale à $W_{ua,1}(k)$ dans l'intervalle $[0,t)$ est :

$$W_{ua,1,k}(t) = \max_{h_{ua} \in SB}(E_{ua}) + \sum_{h \in MP_1} \left\lceil \frac{W_{ua,1,k}(t)}{T_h} \right\rceil E_h + (k-1)E_{ua,1} + C_{ua,1}$$

Les dates de fin de $\tau_{ua,1}(k)$, $k = 1 \dots N_{ua}$, vont se définir comme le plus petit point fixe de l'équation suivante :

$$\begin{cases} L_{ua}^{(0)} & = & C_{ua,1} \\ L_{ua}^{(n+1)} & = & W_{ua,1,k}\left(L_{ua}^{(n)}\right) \end{cases}$$
$$F_{ua,1}(k) = \min\left(n \geq 0,\ L_{ua}^{(n+1)} = L_{ua}^{(n)}\right)$$

Calcul des dates de fin de $\tau_{ua,j}(k)$: Les calculs des dates de fin de $\tau_{ua,2}(k)$ à $\tau_{ua}(k)$ vont reposer sur les mêmes principes. La charge des tâches plus prioritaires que $\tau_{ua,j}(k)$

sur l'intervalle $[0,t)$ est donnée par la date de fin de $\tau_{ua,j-1}(k)$, plus toutes les tâches de $hp(\tau_{ua,j})$ arrivées depuis la fin de $\tau_{ua,j-1}(k)$ et enfin la charge de $\tau_{ua,j}$ elle-même :

$$W_{ua,j,k}(t) = F_{ua,j-1}(k) + \sum_{h \in MP_j} \left(\left\lceil \frac{t}{T_h} \right\rceil - \left\lceil \frac{F_{ua,j-1}(k)}{T_h} \right\rceil \right) E_h + C_{ua,j}$$

Le calcul des dates de fin de $\tau_{ua,j}(k)$ est :

$$\begin{array}{l} \left\{ \begin{array}{lcl} L_{ua}^{(0)} & = & F_{ua,j-1}(k) + C_{ua,j} \\ L_{ua}^{(n+1)} & = & W_{ua,j,k}\left(L_{ua}^{(n)}\right) \end{array} \right. \\ F_{ua,j}(k) = \min\left(n \geq 0,\ L_{ua}^{(n+1)} = L_{ua}^{(n)}\right) \end{array}$$

Le test d'ordonnançabilité de τ_{ua} découle directement des résultats précédents.

Théorème 21 *Une tâche τ_{ua} est ordonnançable si :*

$$\max\left((k-1)T + D_{i,j} - F_{ua}(k)\right) \geq 0 \qquad 1 \leq k \leq N_{ua}$$

Ce test est une condition suffisante puisque les dates de fin calculées sont des bornes supérieures du pire temps de réponse. Aucune condition nécessaire et suffisante n'est connue pour ce problème.

La vérification de toutes les échéances des tâches revient à appliquer la méthode complète sur chaque tâche. L'algorithme correspondant est pseudo-polynomial. La complexité du problème est ouverte.

3.3.2 Collaborations et publications

Ce travail a fait l'objet d'une publication [RRGC02]. Je collabore actuellement avec Claire Pagetti, Julien Forget et Frédéric Boniol, de l'ONERA de Toulouse, sur le problème de la validation temporelle de systèmes de tâches avec contraintes de précédences généralisées en l'absence de primitives de synchronisation. L'absence de primitives de communication est justifiée par le fait que pour des systèmes très critiques, on peut être amené à utiliser un exécutif minimaliste ne proposant pas de primitives de synchronisation bloquantes. Dans ce cas, les précédences sont encodées dans les paramètres temporels des tâches comme dans [CSB90] par ajustement des dates de réveil et des délais critiques. Cette méthode est classique, cependant, l'optimalité d'une telle démarche dans divers contextes n'avait jamais été démontrée. Ainsi nous avons montré l'optimalité d'une telle démarche pour les algorithmes FPP dans les contextes $|0, C_i, D_i \leq T_i, T_i|precsanssyncho$ et $|r_i, C_i, D_i \leq T_i, T_i|precsanssyncho$. Dans ce dernier contexte nous avons montré l'optimalité d'un algorithme basé sur l'algorithme d'affectation optimale d'Audsley [Aud91], respectant les précédences. Nous avons étendu les résultats à des précédences généralisées, c'est-à-dire à des précédences concernant des tâches de périodes différentes : $\tau_i \prec \tau_j$ telles que $\exists n_i, n_j, n_i T_i = n_j T_j$. Ce type de précédences est issu d'un langage de conception de systèmes temps réel développé à l'ONERA dans le cadre de la thèse de Julien Forget.

3.3.3 Points forts de la démarche et perspectives

Les facteurs pratiques tels que les précédences sont présents dans la plupart des systèmes temps réels réalistes. Il nous semblait intéressant de nous intéresser à des motifs généraux, qui étaient des graphes de précédences, afin de généraliser les résultats connus à ce cas. De plus, la prise en compte par notre méthode de la ré-entrance des chaînes de tâches obtenues rend celle-ci applicable à une gamme plus large d'applications. Les

perspectives immédiates concernent l'outillage de la méthode avec outils de synchronisation, et sa généralisation aux contraintes de précédence généralisées. Les perspectives concernant les précédences sans outils de synchronisation (i.e. par encodage) sont en cours d'étude.

3.4 Bilan et perspectives

Nous avons étudié le modèle des transactions dans le cas de FPP et d'EDF, et montré que toutes les études menées sur ce modèle étaient applicables directement aux modèles multiframe.

Dans le cas des FPP, nous avons d'abord créé un modèle applicable aux applications ayant des acquisitions sur des bus de type série, et montré qu'il n'était pas nécessaire dans ce cas de passer par la technique des enveloppes, mais que l'on pouvait utiliser la transaction inverse, qui de par sa nature monotonique, ne présente qu'un seul candidat à l'instant critique. Au passage nous avons proposé des propriétés de plus en plus générales sur les transactions : monotonicité, monotonicité accumulative, puis dominance, permettant d'une part d'optimiser le fonctionnement de l'algorithme mixte que nous avons proposé, mais aussi de quantifier le degré d'exactitude de la réponse fournie. Nous avons montré que NM_2 offrait le meilleur compromis qualité de la solution fournie/temps de calcul.

Dans le cas d'EDF, nous avons proposé le premier test d'ordonnançabilité exact de complexité pseudo-polynomiale pour les transactions.

Le modèle des transactions est aujourd'hui l'un des modèles de tâches les plus avancés, offrant une réelle amélioration de la qualité du test pour la plupart des applications temps réel dirigées par les événements. Ce modèle intègre déjà des facteurs pratiques (ressources critiques, gigue de démarrage). Nous avons commencé à travailler sur la prise en compte des suspensions de tâches [Ric03] dans les transactions. La difficulté réside dans le fait que la date de réveil d'une tâche de la transaction est conditionnée par le temps de réponse de la précédente. L'utilisation de tâches à *offset* dynamique [PGH97] pourrait vraisemblablement être utilisée. D'autres facteurs pratiques telles que les contraintes de précédence doivent encore être intégrées.

Différentes approches utilisent un modèle de transactions multiprocesseur avec échéance globale (de bout en bout), la problématique étant de répartir correctement la contrainte d'échéance sur les différents processeurs. Il me semble intéressant d'adapter nos travaux opérés dans le cas monoprocesseur au cas multiprocesseur, et dans ce cadre, d'étudier le comportement des algorithmes Pfair sur les transactions.

Le modèle prenant en compte les gigues d'activation, il semble qu'il puisse être adapté au cas réparti dans le cadre d'une analyse holistique. Cela reste à démontrer.

D'un point de vue outillage, ce modèle est mûr pour une utilisation industrielle, nous pensons donc prochainement l'intégrer dans une approche de développement orientée modèle industrielle (sujet de stage de Master 2). Afin de rendre ce modèle encore plus intéressant pour le concepteur, il faudrait proposer une approche de validation mixant tâches concrètes dirigées par le temps, et transactions.

Au niveau de l'étude de contraintes de précédences, nous avons montré un exemple typique d'utilisation de la notion d'instant critique et de la période d'activité : étant

donné un ensemble de contraintes particulières, la démarche est la suivante : déterminer un pire cas, puis déterminer la façon dont ce pire cas agit sur la longueur maximale de la période d'activité, et en déduire une borne maximale de temps de réponse. Ce type de démarche peut être généralisé à un ensemble très large de contraintes spécifiques, liées à des facteurs pratiques présents ou à venir.

Chapitre 4

Positionnement dans le cycle de développement

4.1 Atelier de conception

4.1.1 L'atelier DARTSVIEW

Les approches utilisées industriellement pour le développement d'application temps réel critiques se basent de plus en plus sur des approches orientées modèle. Des ateliers tels TopCased, ou ASSERT, permettent d'intégrer différentes méthodes, qui à partir de différentes vues et/ou différents modèles, aident au cycle de développement du logiciel. Du point de vue de la validation temporelle, des outils comme Cheddar se connectent à de tels ateliers afin d'extraire un modèle temporel et répondre au problème de la validation temporelle.

Notre démarche, commencée en 2002, et développée pendant la thèse de Ngo Khanh Hieu, que j'ai encadré à distance, a eu lieu parallèlement au développement de ces méthodologies et a proposé une méthodologie de développement orientée modèle, se basant sur le modèle de conception orienté tâches et parallélisme appelé DARTS (Design Approach for Real-Time systems) [Gom93]. Nous avons utilisé un sous-ensemble de ce modèle, permettant de concevoir une application multitâche conformément au profil Ravenscar proposé pour le développement des applications temps réel [Bur99]. Le système est vu comme un ensemble de tâches, ayant un mode particulier d'activation (activation matérielle basée sur le temps ou les événements, ou bien activation logicielle par synchronisation ou réception de message). Chaque tâche est caractérisée par des paramètres temporels classiques.

Notre approche se base sur le cycle en W, que nous proposons dans [CG05] (voir 4.1). Ce cycle est composé de 2 cycles en V classiques. Un tel cycle est bien adapté au développement de la partie logicielle des applications temps réel de contrôle/commande de taille réduite (comme le drone AMADO par exemple). Un premier cycle en V amène au contrôle/commande d'un simulateur numérique du procédé (par exemple modèle Matlab/SIMULINK) programmé sur la station de développement. Le second cycle en V consiste à adapter la conception au système embarqué, pour aboutir à l'application finale de contrôle/commande du système embarqué.

L'avantage d'une telle démarche est la réduction du temps de développement quand des développements logiciels et matériels parallèles doivent avoir lieu. Ainsi, après la

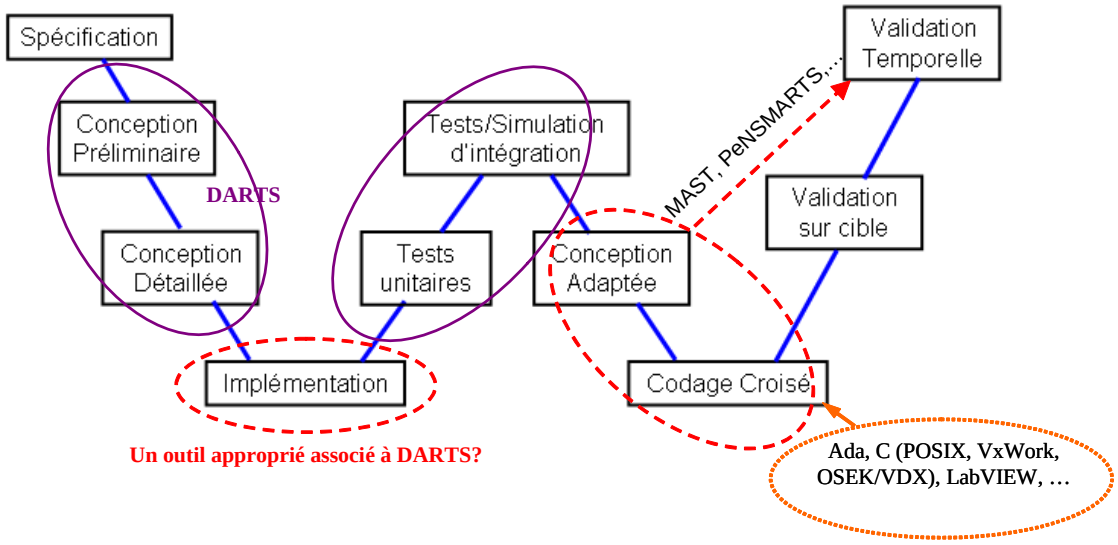

FIGURE 4.1 – DARTSVIEW et le cycle de développement en W.

phase de spécification initiale, et de conception adaptée qui aboutira à un découpage entre fonctions matérielles et logicielles, l'équipe logicielle et l'équipe matérielle peuvent commencer à développer en parallèle. L'autre avantage est d'avoir une simulation totalement numérique du système, on pourra donc réaliser des tests en boucle fermée, sans risque pour le procédé, avec la possibilité de tester simplement les cas de pannes prévues.

Nous avons mis en place un atelier, appelé DARTSVIEW (DARTS pour LabVIEW, qui est un environnement de programmation graphique flots de données très utilisé pour les applications de contrôle/commande sur PC ou PC industriel par exemple). Le rôle de DARTSVIEW est de faciliter le cycle de développement en W en se basant sur un langage simple à prendre en main et simple à utiliser pour des ingénieurs non informaticiens (au bout de 4h15 de TP, les étudiants de première année ENSMA sont capables d'autonomie avec LabVIEW). L'idée est d'utiliser l'aspect graphique flots de données du langage LabVIEW pour offrir une bibliothèque de programmation implémentant directement les concepts de DARTS. Ainsi, on peut voir le modèle DARTS de l'application de contrôle du drone AMADO sur la figure 4.2 : les parallélogramme représentent les tâches, les éclairs représentent leur mode d'activation, les rectangles surmontés de "E" et "L" représentent des tableaux noirs (communication asynchrone), les flèches représentent des accès au matériel, et les éléments comme celui reliant les tâches "Acq CI" et "Traite CI" représentent des boîtes aux lettres (communication synchrone faiblement couplée), sur la figure, ce sont des boîtes aux lettres à écrasement. La figure 4.3 représente l'implémentation de cette conception utilisant la bibliothèque DARTSVIEW.

Il suffit au concepteur, après création du modèle DARTSVIEW, d'implémenter chaque tâche par ses composants logiciels. Il est aidé pour les communications et synchronisations par des outils présents dans les modèles. Ainsi, par exemple, une tâche envoyant un message dans une boîte aux lettres est déjà munie d'une primitive d'envoi de message dans la boîte aux lettres, le programmeur n'a donc plus qu'à implémenter la tâche, en utilisant les primitives déjà placées dans la tâche pour envoyer un message. Le programmeur peut donc se focaliser uniquement sur les aspects fonctionnels, les aspects structurels liés au multitâche étant donnés par DARTSVIEW. Cela permet une implémentation rapide à

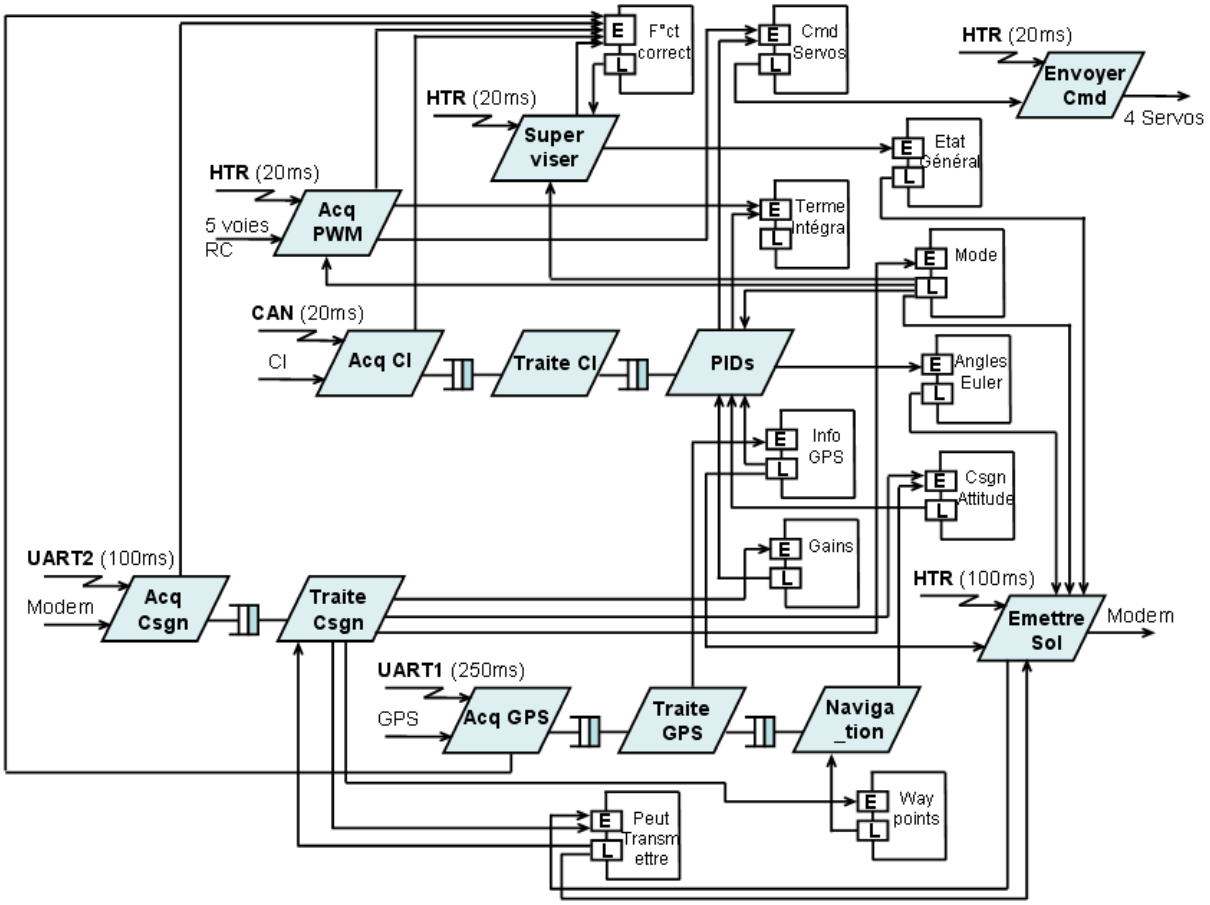

FIGURE 4.2 – conception DARTS du système de contrôle du drone AMADO.

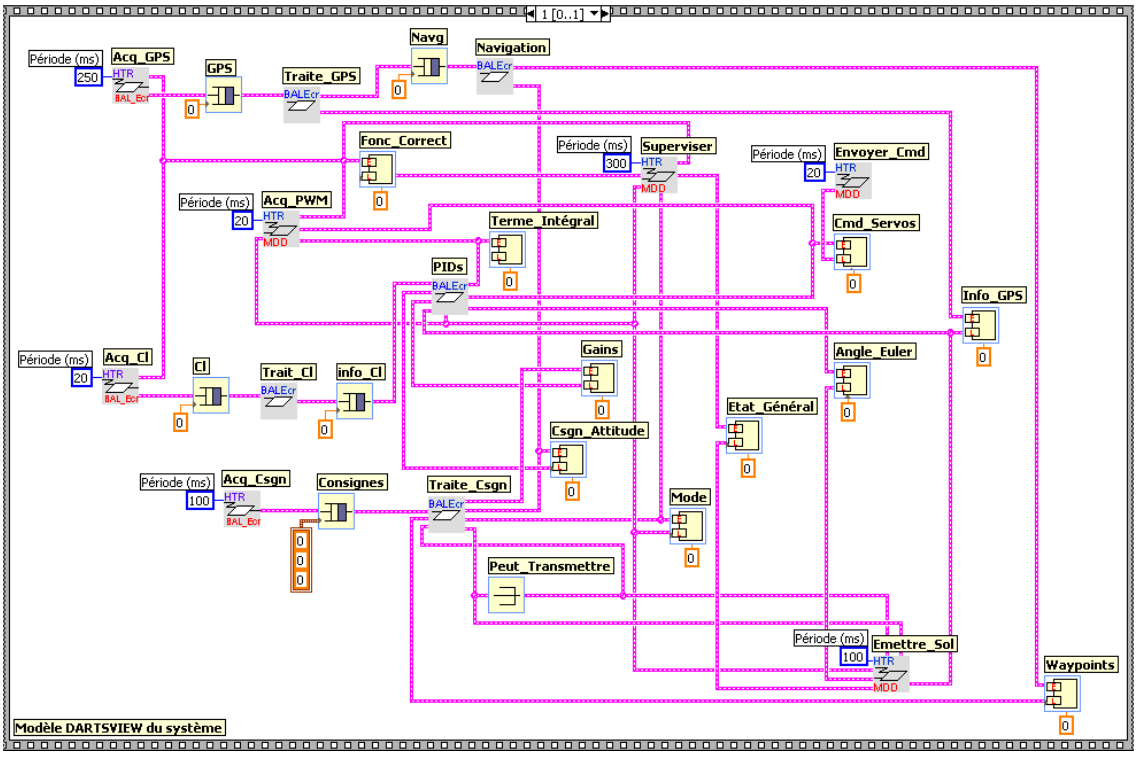

FIGURE 4.3 – Programme DARTSVIEW implémentant le système.

partir de la conception de l'application.

LabVIEW permet des tests unitaires immédiats sur tous les sous-programmes, puisque chacun peut être utilisé directement comme un programme principal. Le prototype de l'application développée sur station de travail est alors relativement rapide à mettre au point, et les fonctions peuvent être testées simplement, les tests d'intégration liés au multitâche sont faits en fermant la boucle de contrôle/commande en utilisant un simulateur (typiquement une simulation numérique avec laquelle le programme DARTSVIEW communique via TCP/IP ou UDP/IP).

La conception adaptée peut alors avoir lieu lorsque le matériel développé parallèlement au premier cycle en V logiciel est prêt à être intégré. Un programme DARTSVIEW est avant tout un modèle, par conséquent au moment de l'exécution, chaque élément participe à créer un modèle interne correspondant au modèle DARTS, enregistré au format XML. L'atelier DARTSVIEW est muni d'un générateur partiel de code prenant en entrée un modèle DARTSVIEW en XML, et un modèle XML décrivant la transformation des éléments d'un modèle DARTS vers un langage cible. Ainsi, le générateur de code génère le squelette des tâches pour différents langages cibles. A l'heure actuel il génère du code C pour les exécutifs VxWorks, la norme POSIX 1003.1d, la norme OSEK (il génère aussi pour cela un fichier de configuration OIL, ce qui a nécessité un générateur de code spécifique pour OSEK), du code pour la norme Ada 95, ainsi que des fichiers modèles d'applications pour les outils de validation temporelle PeNSMARTS et MAST [DHG+08]. DARTSVIEW aide par là la phase de développement croisé et la transformation du modèle DARTS pour la validation temporelle.

4.1.2 Publications

Cette méthode a été développée lors de la thèse de Ngo Khanh Hieu [Ngo08], et publié dans [NG03, NG07]. La méthodologie proposée est la base du livre co-écrit avec Francis Cottet [CG05].

4.1.3 Points forts de la démarche

La méthode est très simple à prendre en main, et la bibliothèque est flexible. Cependant, l'atelier ne passe pas à l'échelle des applications à haut degré de criticité (le modèle est uniquement opérationnel). De plus, des ateliers ouverts sont aujourd'hui arrivés à maturité. Nous encadrons cette année, Michaël Richard et moi, un stage de Master 2 qui va ouvrir la voie à l'intégration de nos modèles et méthodes d'ordonnancement dans l'atelier TopCased.

4.2 Aide au choix des paramètres temporels

Il est rare que les paramètres temporels et les contraintes de temps appliquées aux tâche d'un système soient immuables. Ainsi, sur l'étude de cas du drone AMADO, les contraintes de temps de bout en bout appliquées à la chaîne de régulation d'attitude ont été obtenues à partir de simulations Matlab/SIMULINK du drone. Les aérodynamiciens ont modélisé le drone, et les 3 PID appliqués à la régulation d'attitude en discrétisant le tout, et en ajoutant la prise en compte de délais entre acquisition et commande. Nous avons pu alors retranscrire ces contraintes en terme de contraintes de période et de délai critique sur les tâches concernées. Cependant, ces paramètres temporels auraient sans doute pu être plus laxes, si l'ordonnançabilité du système s'en était retrouvée impossible. L'aide au choix de paramètres temporels vise à fournir des outils utilisables au niveau conception pour, lorsqu'on a une évaluation a priori des durées des traitements qui seront implémentés, aider le concepteur à savoir quelles contraintes doivent être relaxées, ou encore quelle laxité il a à ajouter une nouvelle tâche dans un système, ou encore jusqu'à quel point il peut modifier les paramètres d'une tâche sans affecter l'ordonnançabilité du système. Il est donc intéressant de fournir des outils de type analyse de sensibilité [Ves94].

Nous avons proposé une telle approche dans [CGR02], où nous avons représenté graphiquement 3 dimensions par tâche, les dimensions représentant C_i, D_i et T_i dans le contexte $|0, C_i, D_i, T_i|$. Nous construisons ensuite l'espace d'ordonnançabilité en utilisant les conditions nécessaires, conditions suffisantes, et conditions nécessaires et suffisantes classiques afin de diminuer le nombre de points explorés. Ce type d'approche pourrait aujourd'hui être grandement amélioré avec les avancées qui ont eu lieu récemment dans l'analyse de sensibilité.

Une autre propriété temporelle importante est la gigue temporelle : elle représente la variation de temps de réponse d'une instance à l'autre. Par exemple, si une boucle d'asservissement est exécutée dans une tâche de période 1 milliseconde à échéance sur requête, il est possible que deux instances soient distantes d'une durée arbitrairement petite (une instance se termine juste avant échéance, et l'instance suivante est immédiatement réveillée et exécutée), et que deux autres soient distantes de deux périodes (exécution au réveil pour l'une, et exécution jute avant échéance pour la suivante). Cela peut être

nuisible à certains algorithmes, par exemple lorsque l'on se base sur un échantillonnage avec interpolation entre les points, ou bien lorsque l'on utilise une intégrale ou bien une dérivée de variable d'état. J'ai collaboré avec Laurent David, durant sa thèse de doctorat, à la mise en œuvre d'une plate-forme d'expérimentation (un pendule inversé contrôlé par VxWorks) permettant de mettre en lumière les effets néfastes d'une gigue temporelle sur la stabilité d'un procédé contrôlé. Cette collaboration a donné lieu à 2 publications [DGC02, DCG00]. L'une portant sur l'étude de cas, l'autre portant sur les méthodes permettant de diminuer la gigue temporelle. Ce que nous avons proposé, et qui a été proposé parallèlement par [GD97], consiste à éviter la concurrence directe entre des tâches dont on veut diminuer la gigue, en décalant leur date de réveil. Ainsi, si deux tâches τ_i et τ_j de période respective T_i et T_j doivent éviter d'être en concurrence, il suffit de décaler leurs dates de réveil respectives r_i et r_j de sorte que $|r_i - r_j| \in [max(C_i, C_j)..PGCD(T_i, T_j)[$ (où $PGCD$ est le Plus Grand Diviseur Commun et C_i est la pire durée d'exécution de τ_i).

Ce type d'approche pourrait être mise en œuvre pour aider le concepteur à choisir les dates de réveil initiales des tâches concrètes de façon à éviter l'instant critique. En effet, ce paramètre temporel, dans les applications dirigées par le temps, est typiquement utilisé dans ce but d'amélioration de l'ordonnançabilité. Cependant, nombre d'applications mélangent tâches strictement périodiques et tâches sporadiques. Nous souhaitons développer une approche mélangeant tâches non concrètes, puis transactions, à des tâches strictement périodiques dont on souhaite choisir les dates de réveil de sorte à augmenter l'ordonnançabilité du système. Le problème, sans les sporadiques, est déjà Co-NP-difficile au sens fort, mais ce problème a été adressé avec une condition suffisante d'ordonnançabilité pseudo-polynomiale dans [PL05]. Notre idée est de mettre en place une heuristique basée sur cette condition afin d'adresser le problème.

4.3 Dimensionnement de systèmes répartis

Nous travaillons à la conception d'une méthode d'ordonnancement et de placement conjoints des tâches dans un système distribué. Elle fait l'objet de la thèse de François Dorin, dont la thèse a débuté en septembre 2007. Les architectures matérielles et logicielles sont analogues à celles considérées dans la thèse de Michaël Richard, soutenue en 2002. L'objectif est ici de fournir des méthodes pour dimensionner l'architecture matérielle : nombre de processeurs, le débit des réseaux, etc. Nous souhaitons aussi analyser la robustesse d'une solution dans sa capacité à absorber des modifications mineures des paramètres des tâches comme par exemple l'allongement d'une durée de tâche ou d'une caractéristique matérielle (la fréquence d'un processeur, modification du débit d'un réseau).

L'originalité de l'approche développée au laboratoire repose sur le choix de faire l'ordonnancement et le dimensionnement/placement de façon conjointe. En effet, la plupart des approches proposées dans la littérature considèrent le problème du placement seul, les priorités des tâches étant connues. Pour cela nous utilisons une approche de type séparation & évaluation inspirée de l'algorithme de [BFR75], et permettant d'éviter par construction les symétries liées au placement de tâches sur des processeurs. Ainsi, sur la figure 4.4, les nœuds carrés correspondent au choix d'un nouveau processeur alors que

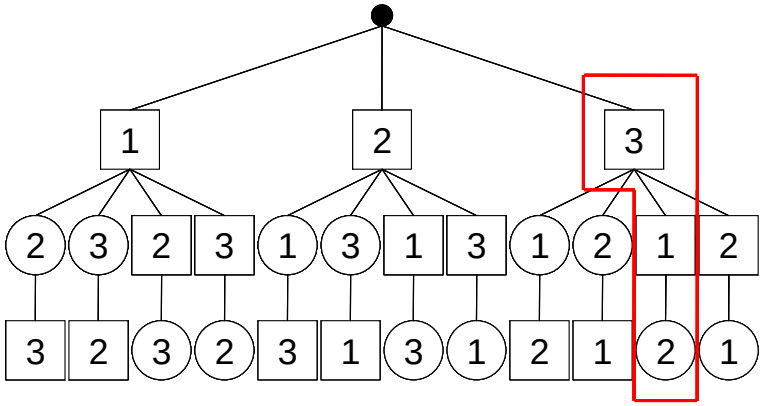

FIGURE 4.4 – Principe de l'arbre bicolore de Bratley Florian et Robillard.

les nœuds ronds correspondent au placement d'une tâche sur le processeur courant. La partie encadrée correspond à un placement de la tâche τ_3 sur un processeur, et de τ_1 et τ_2 sur un second processeur avec une priorité plus grande pour τ_1. L'idée est d'établir une borne inférieure des pires temps de réponse d'une tâche au moment où on la place dans l'arbre afin de couper les mauvaises branches au plus tôt. Nous nous plaçons dans le contexte large $dist, unrelated|C_i, D_i, T_i, J_i|prec$ dans lequel nous appliquons une analyse holistique [TC94] pour valider un placement complet, associé au test de coupe à chaque placement/ordonnancement partiel. L'approche est optimale au regard de l'analyse holistique, et consiste à placer les tâches sur un nombre minimal de processeur. Au fur et à mesure que l'on obtient des placements/ordonnancements valides, le nombre de processeur autorisés diminue afin de chercher à améliorer ce nombre par la suite. Etant donné que l'architecture répartie est centrale dans l'analyse holistique, nous nous plaçons dans le cadre de l'ordonnancement partitionné (pas de migration des tâches pendant la vie du système).

Le système est vu comme un ensemble de *pools* de processeurs identiques (voir figure 4.5) reliés par des réseaux choisis de sorte à avoir des délais de transmission bornés. Chaque tâche est affectée à un pool de processeurs. Le but de la méthode est de minimiser le nombre de processeurs par pool. Les réseaux reliant les processeurs de chaque pool sont vus comme des processeurs particuliers "ordonnançant" les messages de façon non préemptible.

Voici le principe de fonctionnement de la procédure de séparation et évaluation, pour chaque pool de processeurs :
Règles de séparation :

1. Le niveau 0 est la racine de l'arbre de recherche (aucune tâche n'a été placée).
2. Le niveau 1 est constitué de nœuds carrés correspondant à placer une tâche sur le premier processeur du système, avec la priorité maximale sur ce processeur.
3. Une tâche est placée au niveau i sur un nœud rond ou carré si
   - elle n'apparaît pas dans le préfixe de la branche (une tâche est placée une seule fois),
   - une tâche k ne peut pas être placée sur un nœud carré si le préfixe de la branche contient au moins un nœud carré contenant une tâche de numéro supérieur à k

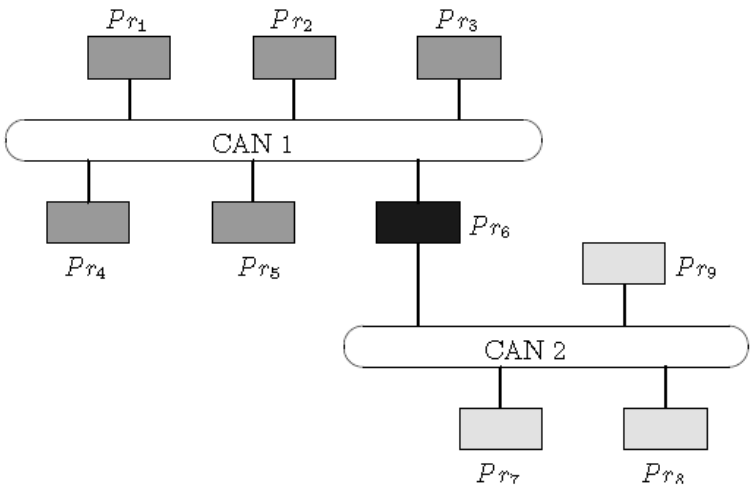

FIGURE 4.5 – Système composé de 3 pools de processeurs.

(évite les symétries),
- on ne peut pas placer de nœud carré si cela augmente le nombre de processeurs par rapport à la meilleure solution connue,

Règles de coupe : on évalue la possibilité pour la branche courante de satisfaire les contraintes, et on coupe si il n'est pas possible de diminuer le nombre de processeurs par rapport à la meilleure solution connue.

La stratégie de construction est d'utiliser une recherche en profondeur d'abord (pour éviter le problème de l'explosion combinatoire en espace), en essayant de charger les processeurs le plus possible d'abord.

La méthode se comporte d'autant mieux que le système est contraint (charge par tâche importante) [FD08]. Nous avons comparé notre méthode de réduction du nombre de processeurs à une méthode heuristique de diminution du nombre de processeurs fonctionnant dans le contexte $multi|0, C_i, D_i, T_i|$ proposée dans [FBB06]. Notre méthode permet d'économiser 2 processeurs en moyenne pour un système de 100 tâches avec une charge $U = 15$ [DRGR09].

Publications

Cette étude a fait l'objet des publications [DRGR08, DRGR09].

Points forts de la démarche

Cette démarche est la seule du genre à notre connaissance. C'est exactement le type de démarche centrale préconisée par la norme AUTOSAR dans le domaine automobile. Cependant, il nous faut adapter celle-ci à l'approche par module et pas par tâche de la norme. On peut imaginer l'apport d'une telle méthode dans un atelier de conception logicielle, permettant au concepteur de s'affranchir du placement et de l'affectation de priorité. Il nous faut considérer des contraintes de niveau conception afin de rendre la méthode utilisable industriellement, par exemple forcer le placement de certaines tâches liées à du matériel particulier, prendre en compte plus de facteurs pratiques (ressources

critiques, transactions, transactions réparties, suspension), et amener le système à prendre en compte des tâches redondées.

La méthode est optimale lorsqu'elle termine, cependant sur de grandes instances de problèmes, nous sommes amenés à arrêter la recherche et à se contenter d'une solution sous-optimale. Su ce genre de problème on pourrait adapter une méthode méta-heuristique, qui pourrait seconder l'approche optimale.

4.4 Bilan et perspectives

Le domaine de l'aide au développement d'applications temps réel est extrêmement large, et nous avons apporté quelques éléments de façon ponctuelle à ce domaine, cependant, chaque approche présentée dans les sections ordonnancement hors-ligne et en-ligne aurait pu figurer dans cette partie.

En effet, de nombreux points sur les problèmes théoriques que nous avons étudiés séparément, permettraient, groupés, d'offrir des méthodes d'aide à la conception et de validation très intéressantes pour le concepteurs d'applications temps réel critiques. Je développe ce point en dernière partie de ce mémoire.

Aujourd'hui notre équipe veut proposer des outils industriels ou pré-industriels basés sur nos avancées théoriques dans le domaine de la validation temporelle. Nous avons diverses méthodes inédites ou avancées de validation, dont la plupart sont implémentées sur des prototypes universitaires que nous avons développés (placement/ordonnancement avec minimisation du nombre de processeurs, PeNSMARTS, méthodes basées sur le modèle des transactions, DARTSVIEW, etc.).

Troisième partie

Perspectives

Chapitre 5

Perspectives

Au fil des sections de ce mémoire, j'ai eu l'occasion d'évoquer différentes perspectives de recherche liées à des études que nous menons ou avons menées. Je ne reviens donc pas sur ces perspectives détaillées ici, mais adopte une vue plus large des problèmes liés aux problèmes abordés.

5.1 Approche hors-ligne

L'ordonnancement hors-ligne et l'ordonnancement en-ligne n'ont pas les mêmes types d'application cible. D'un point de vue validation de systèmes à haut degré de criticité devant être certifiés, l'approche hors-ligne offre des avantages liés à l'absence d'ordonnanceur, car certifier un exécutif complet est un processus complexe, par rapport à la certification d'un séquenceur. Du côté méthode de développement et langage cible, les langages synchrones sont des choix adaptés à ce type d'applications très critiques, et pourraient de ce fait apparaître comme des concurrents directs des approches hors-lignes. En effet, les approches orientées modèle de type automate, permettent d'utiliser les outils du *Model checking* , et apportent donc des solutions intéressantes au problème de la validation fonctionnelle.

Cependant, la plupart de ces approches obligent le concepteur à définir un système d'horlogerie commune implicitement ou explicitement, car les automates définissant les sous-systèmes doivent être synchronisés avant implémentation. Cette structure pourrait être assouplie si on appliquait des techniques d'ordonnancement hors-ligne sur chaque sous-système directement, en prenant en compte exclusions mutuelles, et communications, puis on pourrait exprimer l'ordonnancement produit en terme de propriétés (séquentialité, délai inter-tâche) pouvant être exploitées dans le processus de validation.

D'autres types d'applications susceptibles d'utiliser des approches hors-ligne ont de fortes chances de contenir des tâches (ou simplement des ISR) non concrètes. A l'heure actuelle, ce trafic non concret, lorsqu'il est plus prioritaire que les tâches du système, est pris en compte dans le surcoût global appliqué au modèle des tâches du système. Ainsi, si nous avions mis en œuvre un ordonnancement hors-ligne sur AMADO, il nous aurait fallu prendre en compte l'interférence liée aux acquisitions de données, par définition non concrètes, dans la durée des tâches du modèle. Ceci aurait entraîné un pessimisme qui pourrait être évité si l'on avait des approches hors-ligne intégrant un trafic en-ligne, par exemple de transactions.

Nous pouvons citer d'autres problèmes qui ont été adressés dans le cadre d'ordonnanceurs hiérarchiques, que nous pourrions étudier, ainsi la norme ARINC 653 introduite en avionique partitionne, notamment temporellement, différents exécutifs qui cohabitent sur un même calculateur. Les approches hors-ligne pourraient proposer des solutions d'ordonnancement prenant en compte ce principe.

Le problème de la cyclicité est un facteur très limitant pour les approches hors-ligne de type séparation et évaluation, car bien qu'optimales pour les systèmes de tâches simultanées, ces approches ne peuvent pas pour le moment fonctionner sur les système de tâches différées. Or un concepteur peut améliorer l'ordonnançabilité de son application en décalant les dates de réveil. Que ce soit en mode mono ou multiprocesseur, le résultat de cyclicité doit donc être étendu, ou bien la méthode adaptée, de façon à être applicables dans le seul cas qu'un concepteur devrait choisir pour une application hors-ligne : une application avec des tâches différées.

5.2 Approches en-ligne

Nous avons étudié différents facteurs pratiques permettant de modéliser finement les applications, afin d'avoir une étude d'ordonnançabilité, pire cas par définition, la moins pessimiste possible. Le modèle des transactions est en ce sens l'un des modèles les plus avancés permettant de modéliser les tâches d'une application. De nombreuses pistes de recherche sont ouvertes pour ce modèle : prise en compte des suspensions, ordonnancement multiprocesseur, global et partitionné. L'ordonnancement partitionné ne devrait pas poser de difficulté théorique car le modèle intègre une gigue, et permet de ce fait une analyse holistique. Par contre, dans le cadre de l'ordonnancement global, où Pfair est très étudié, il conviendrait d'étudier ses effets face à des transactions.

Il serait intéressant d'augmenter la prise en compte de facteurs pratiques dans notre méthode de placement et ordonnancement conjoint par exemple en prenant en compte non pas des tâches mais des transactions. Dans ce cas, il nous faudra considérer des transactions réparties. Cette technique pourrait aussi être adaptée à la problématique de l'industrie automobile liée à l'utilisation d'AUTOSAR. Dans ce cas, nous devrons travailler avec des modules, quasiment (seule la partie synchronisation/communication nous intéresse, pour les fonctionnalités nous pouvons utiliser une abstraction temporelle) écrits en OSEK. Pour les modules se synchronisant, déclenchés par un événement externe (trame CAN ou FlexRay par exemple), et se trouvant implémentés dans des tâches différentes, le modèle pourrait d'ailleurs reposer sur des transactions.

De nombreuses applications réelles mélangent programmation dirigée par le temps et programmation dirigée par les événements. Comme nous l'avons évoqué pour l'approche hors-ligne, il est plus que probable que les tâches dirigées par le temps soient décalées afin d'éviter un instant critique. On ne peut pas forcer les tâches non concrètes à ne pas créer des instants critiques, mais on peut faire cela pour les tâches concrètes. Il faut pour ce cas d'une part trouver des algorithmes de validation des ordonnancements (problème co-np-difficile au sens fort rien que pour le cas des tâches concrètes différées, à l'image du problème adressé pour les messages dans un système réparti dans [GHN08]) si possible efficaces (conditions suffisantes) mais exploitant l'augmentation d'ordonnançabilité introduite par les décalages, afin de les utiliser pour proposer des algorithmes de choix

des dates de réveil des tâches concrètes (autre problème Co-NP-difficile).

5.3 La validation temporelle dans le cycle de développement

Dans le cycle en W, comme dans beaucoup de cycles de développement, la validation a une place en fin de cycle. L'explication, donnée en général dans les cours portant sur la validation temporelle, tient au fait qu'il faut connaître les pires durées d'exécution des tâches, ainsi que leur structure (communications/synchronisation) avant de mettre en place un modèle de tâche adapté à l'étude d'ordonnançabilité. Cependant, dans les systèmes complexes industriels, de nombreux modules logiciels sont ré-utilisés d'une version à une autre d'un système. Par conséquent, au niveau de la phase de conception, on a déjà un certain nombre d'informations qui permettent d'étudier partiellement un système. De plus, dans les systèmes partitionnés temporellement qui ont fait leur apparition dans l'avionique, il est probable qu'à un niveau pré-conception, on alloue à tel ou tel instrumentier un budget temps, dans lequel il devra exécuter ses tâches.

Les approches de type analyse de sensibilité permettent, pour des modèles simples, à l'aide de techniques telles que la programmation par contraintes, d'établir des domaines d'ordonnançabilité de tâches. On peut supposer que de nombreuses études vont aller dans ce sens pour prendre en compte au plus tôt des informations partielles, ou aider à établir des budgets temps tout en garantissant l'ordonnançabilité globale du système.

Notre approche d'ordonnancement/placement/dimensionnement conjoint, et les approches de type co-design en général, donnent à réfléchir sur la façon dont on pourra concevoir et dimensionner un système. La technique immédiate est de brancher des greffons de validation temporelle sur les outils de conception existants. Il existe déjà des greffons , comme Cheddar [SLNM04], extrayant des paramètres temporels d'un modèle AADL et réalisant une étude temporelle du système. Cependant, le support d'exécution a un impact sur l'ordonnancement et l'ordonnancement sur le support. Par conséquent, des techniques de co-design utilisant des modèles d'ordonnancement avancés devraient être explorées.

5.4 Outils

Des entreprises développant des systèmes temps réel complexes, susceptibles d'avoir besoin de modèles avancés de validation temporelle, se tournent aujourd'hui vers des systèmes ouverts *Open source*. Différentes raisons expliquent ce changement :

- le développement des ateliers méthodologiques est sous-traité, pour des raisons évidentes de coût, et de domaine d'expertise,
- l'avantage donné par rapport à la concurrence via l'utilisation d'un logiciel développé par une société tierce est nul car il est évident que la société d'édition de ce logiciel le proposera aussi aux concurrents directs,
- même s'il peut être à un moment le client le plus important d'un éditeur de logiciel et ainsi diriger par ses demandes les évolutions d'un logiciel, un industriel peut voir cette société soit disparaître (quid des licences ?), soit être rachetée par un

éditeur de logiciel plus imposant, sur lequel les moyens de pression seront trop faibles pour imposer les directions dans lesquelles le logiciel évoluera lors de ses prochaines versions.

C'est une chance pour les universitaires qui travaillent sur des aspects temps réel, car pour la première fois, ils peuvent développer des logiciels pérennes, utilisables assez facilement au niveau industriel, sans un investissement financier important, ou même un problème lié à l'opacité des formats d'échanges entre logiciels.

Dans les mois et les années à venir, nous allons adopter une démarche systématique d'implémentation de nos méthodes dans des outils pouvant être greffés sur des ateliers de développement ouverts orientés modèles. Nous commencerons par mettre au point, en collaboration avec l'équipe Ingénierie des Données du laboratoire, les méta-modèles permettant d'intégrer la plupart de nos modèles présents et futurs de tâches. Nous extrairons alors à partir de différents modèles utilisés dans l'industrie (profil MARTE, AADL, etc.) les informations nécessaires à la construction de nos modèles. Ce processus devrait aussi permettre de lever des problèmes théoriques qui nous auraient échappés sans la mise à disposition de nos techniques sur des cas réels. Ainsi, prochainement, nos prototypes de thèse, Master, devraient être intégrés dans des greffons et sortir du cadre du laboratoire. Ce sera aussi l'occasion de montrer notre savoir-faire, et de monter des projets en collaboration avec des industriels autour des méthodes de validation temporelle.

Quatrième partie

Bibliographie

Bibliographie

[AABD+04] Yamine Aït-Ameur, Frédéric Boniol, Rémi Delmas, Emmanuel Grolleau, Nathalie Torrecillas, and Virginie Wiels. Integration of heterogeneous formal techniques for the design of avionics systems. In *proc. Data Systems in Aerospace*, 2004.

[ABD+95] N.C. Audsley, A. Burns, R.I. Davis, K.W. Tindell, and A.J. Wellings. Fixed priority pre-emptive scheduling : an historical perspective. *Real-Time Systems*, 8 :173–198, 1995.

[AHS05] J. Anderson, P. Holman, and A. Srinivasan. Fair scheduling of real-time tasks on multiprocessors. *Handbook of Scheduling*, 2005.

[AS97] Tarek F. Abdelzaher and Kang G. Shin. Comment on a pre-run-time scheduling algorithm for hard real-time systems. *IEEE TRANS on Software Engineering*, 23(9) :699–700, 1997.

[AS00] J. Anderson and A. Srinivasan. Early release fair scheduling. *Proceedings of ECRTS*, 2000.

[Aud91] N.C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, University of York, 1991.

[Bak91] T.P. Baker. Stack-based scheduling of real-time processes. *Journal of Real-Time Systems*, 3 :67–99, 1991.

[BBB03] E. Bini, G.C. Buttazzo, and G.M. Buttazzo. Rate monotonic analysis : the hyperbolic bound. *IEEE Transactions on Computers*, 57(2) :933–942, 2003.

[BBNR07] Sanjoy Baruah, Enrico Bini, Chau Huyen Nguyen, and Pascal Richard. Continuity and approximability of response time bounds. In *Proc. Euromicro Conf. on Real-Time Systems, WIP*, Pisa, Italy, 2007.

[BCGM99] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *The International Journal of Time-Critical Computing Systems*, (17) :5–22, 1999.

[BCM99] S. Baruah, D. Chen, and A. Mok. Static-priority scheduling of multiframe tasks. In *11th Euromicro Conference on Real-Time Systems*, 1999.

[BCPV96] S. Baruah, N. Cohen, G. Plaxton, and D. Varvel. Proportionate progress : A notion of fairness in resource allocation. *Algorithmica*, 15 :600–625, 1996.

[BFR75] P. Bratley, M. Florian, and P. Robillard. Scheduling with earliest start and due date constraints on multiple machines. *Naval Research Logistic Quaterly*, 22(1) :165–173, 1975.

[BGP95] S. Baruah, J. Gehrke, and C.G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proc. of the 9th International Parallel Processing Symposium*, pages 280–288, 1995.

[BHR90] Sanjoy K. Baruah, Rodney R. Howell, and Louis Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2 :301–324, 1990.

[BHS99] B. Bullnheimer, R. F. Hartl, and C. Strauss. A new rank based version of the ant system :a computational study. *Central European Journal for Operations Research and Economics*, 7(1) :25–38, 1999.

[Bin07] Enrico Bini. Low power. In *Ecole d'Ete Temps Réel*, Nantes, 2007.

[BMR90] S. Baruah, A. Mok, and L. Rosier. The preemptive scheduling of sporadic real-time tasks on one processor. In *Proc. of the 11th Real-Time Systems Symposium*, pages 182–190, 1990.

[Bur99] A. Burns. The ravenscar profile. *ACM SIGAda Ada Letters*, 19(4) :49–52, 1999.

[BW97] A. Burns and A.J. Wellings. Synchronous sessions and fixed priority scheduling. *Journal of systems architecture*, 44(2) :107 – 118, 1997.

[Cav97] Sergio Vanderlei Cavalcante. *A hardware-software co-design system for embedded real-time applications*. PhD thesis, University of Newcastle upon Tyne, 1997.

[CG05] F. Cottet and E. Grolleau. *Systèmes Temps Réel de Contrôle-Commande*. Dunod, 2005.

[CG06] L. Cucu and J. Goossens. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. *Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 397–405, 2006.

[CG07] L. Cucu and J. Goossens. Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems. In *Design, Automation and Test in Europe*, pages 1635–1640, 2007.

[CGG04] Annie Choquet-Geniet and Emmanuel Grolleau. Minimal schedulability interval for real time systems of periodic tasks with offsets. *Theoretical of Computer Sciences*, 310 :117–134, 2004.

[CGGC00] Annie Choquet-Geniet, Emmanuel Grolleau, and Francis Cottet. Étude hors ligne d'une application temps réel à contraintes strictes. *Technique et Science informatique*, 19(10) :1373–1397, 2000.

[CGR02] Francis Cottet, Emmanuel Grolleau, and Pascal Richard. Une approche graphique pour l'aide à la conception d'applications temps réel ordonnançables. *Technique et Science Informatiques*, 21(3) :315–343, 2002.

[CR98] F. Cassez and O. Roux. Automates hybrides à files. In *Séminaire du GdR ARP-STS*, 11 Dec. 1998.

[CS89] H. Chetto and M. Silly. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10) :1261–1269, 1989.

[CSB90] Houssine Chetto, Marilyne Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2, 1990.

[DCG98] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. Technical Report IRIDIA/98-10, Université Libre de Bruxelles, 1998.

[DCG99] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2) :137–172, 1999.

[DCG00] Laurent David, Francis Cottet, and Emmanuel Grolleau. Gigue temporelle et ordonnancement par échéance dans les applications temps réel. In *proc. IEEE Conf. Inter. Francophone d'Automatique, CIFA'00*, pages 681–686, Lille, France, 2000.

[Der74] M.L. Dertouzos. Control robotics : the procedural control of physical processes. In *proc. IFIP Congress*, pages 807–813, 1974.

[DG97] M. Dorigo and L.M. Gambardella. Ant colony system : a cooperative learning approach to the tsp. *IEEE Trans on Evolutionary Computation*, 1(1), 1997.

[DG00] R. Devillers and J. Goossens. Liu and layland's schedulability test revisited. *Information Processing Letters*, 73(5) :157–161, 2000.

[DGC02] Laurent David, Emmanuel Grolleau, and Sébastien Constantin. Plate-forme d'expérimentation pour l'ordonnancement des applications temps réel à contraintes strictes. In *proc. Real-Time Systems (RTS'02)*, pages 33–46, Paris, 2002.

[DHG+08] José María Drake, Michael González Harbour, José Javier Gutiérrez, José Carlos Palencia, and Julio Luis Medina. Modeling and analysis suite for real time applications. Technical report, Universidad de Cantabria, 2008.

[DLS97] Z. Deng, J.W.S. Liu, and J. Sun. A scheme for scheduling hard real-time applications in open systems environment. In *Proc. 9th Euromicro Workshop on Real-Time systems*, pages 191–199, 1997.

[Dor91] M. Dorigo. *Optimization, learning and natural algorithms (in Italian)*. PhD thesis, Dip. di elettronica, Politecnico de Milano, Italy, 1991.

[DRGR08] François Dorin, Michaël Richard, Emmanuel Grolleau, and Pascal Richard. Minimizing the number of processors for real-time distributed systems. In *proc. 16th International Conference on Real-Time and Network Systems, RTNS2008*, Rennes, France, Oct. 2008.

[DRGR09] François Dorin, Michael Richard, Emmanuel Grolleau, and Pascal Richard. Minimisation du nombre de processeurs pour les systèmes temps réel distribués. In *Proc. 10eme conférence de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'09)*, 2009.

[DS95] M. DiNatale and J.A. Stankovic. Applicability of simulated annealing methods to real-time scheduling and jitter control. In *Proc. IEEE Real-Time Systems Symposium*, 1995.

[ER08] Friedrich Eisenbrand and Thomas Rothvoß. Static-priority real-time scheduling : Response time computation is np-hard. In *Proc. 29th Real-Time Systems Symposium*, pages 397–406, Barcelona, Spain, 2008.

[ER09] Friedrich Eisenbrand and Thomas Rothvos. New hardness results for diophantine approximation. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *APPROX-RANDOM*, volume 5687 of *Lecture Notes in Computer Science*, pages 98–110. Springer, 2009.

[FBB06] N. Fisher, S. Baruah, and T. Baker. The partitioned scheduling of sporadic tasks according to static-priorities. In *Proc. 18th Euromicro Conference on Real-Time Systems*, 2006.

[FD08] Emmanuel Grolleau et Pascal Richard François Dorin, Michaël Richard. Minimizing the number of processors for real-time distributed systems. In *proc. 16th International Conference on Real-Time and Network Systems*, Rennes, France, Oct. 2008.

[GB95] T.M. Ghazalie and T.P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9(1), 1995.

[GCG00a] Emmanuel Grolleau and Annie Choquet-Geniet. Cyclicité des ordonnancements de systèmes de tâches périodiques différées. In *proc. Real-Time Systems*, 2000.

[GCG00b] Emmanuel Grolleau and Annie Choquet-Geniet. Off-line computation of real-time schedules by means of petri nets. In *proc. Workshop On Discrete Event Systems*, pages 309–316, Ghent, Belgium, 2000.

[GCG00c] Emmanuel Grolleau and Annie Choquet-Geniet. Utilisation des réseaux de petri pour l'ordonnancement hors-ligne optimal des systèmes temps réel. In *proc. IEEE Conférence Internationale Francophone d'Automatique*, Lille, France, 2000.

[GCG01] Emmanuel Grolleau and Annie Choquet-Geniet. Ordonnancement de tâches temps réel en environnement multiprocesseur à l'aide de réseaux de petri. In *proc. Real-Time Systems*, Paris, France, 2001.

[GCG02] Emmanuel Grolleau and Annie Choquet-Geniet. Off-line computation of real-time schedules using petri nets. *Discrete Event Dynamic Systems*, 12(3) :311–333, 2002.

[GCGC98a] Emmanuel Grolleau, Annie Choquet-Geniet, and Francis Cottet. Cyclicité des ordonnancements au plus tôt des systèmes de tâches temps réel. In *proc. Rencontres Francophones du Parallélisme, RenPar'10*, pages 39–42, Strasbourg, 1998.

[GCGC98b] Emmanuel Grolleau, Annie Choquet-Geniet, and Francis Cottet. Validation de systèmes temps réel à l'aide de réseaux de petri. In *proc. Approches Formelles dans l'Assistance au Développement de Logiciels*, pages 137–148, Poitiers, France, 1998.

[GCGC99] Emmanuel Grolleau, Annie Choquet-Geniet, and Francis Cottet. Modélisation de systèmes temps réel par réseaux de petri autonomes en vue de leur analyse hors-ligne. In *proc. Modélisation des Systèmes Réactifs*, pages 17–26, Cachan, France, 1999.

[GD97] J. Goossens and R. Devillers. The non-optimality of the monotonic assignments for hard real-time offset free systems. *Real-Time Systems : The International Journal of Time-Critical Computing*, 13(2) :107–126, 1997.

[GHN08] M. Grenier, L. Havet, and N. Navet. Pushing the limits of can - scheduling frames with offsets provides a major performance boost. In *Proc. 4th European Congress on Embedded Real Time Software (ERTS 2008)*, Toulouse, France, 2008.

[Gom93] Hassan Gomaa. *Software Design Methods for Concurrent and Real-Time Systems*. Addison Wesley, 1993.

[Gro99] Emmanuel Grolleau. *Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de Petri en environnement monoprocesseur et multiprocesseur*. PhD thesis, ENSMA - Université de Poitiers, 1999.

[HKL91] M.G. Harbour, M.H. Klein, and J.P. Lehoczky. Fixed priority scheduling of periodic tasks with varying execution priority. In *Proceedings of Real-Time Systems Symposium*, pages 116–128. San Antonio (Texas), 1991.

[HKL94] M.G. Harbour, M.H. Klein, and J.P. Lehoczky. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Transaction on Software Engineering*, 20(1) :13–28, 1994.

[HKO+93] Michael Gonzalez Harbour, Mark H. Klein, Ray Obenza, Bill Pollak, and Tom Ralya. *A Practitioner's Handbook for Real-Time Analysis : Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, 1993.

[Hor74] W. Horn. Some simple scheduling algorithms. *Naval Research Logistic Quarterly*, 21, 1974.

[Jac55] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Management science research project 43, University of California, Los Angeles, 1955.

[JP86] M. Joseph and P. Pandya. Finding response times in real-time system. *The Computer Journal*, 29(5) :390–395, 1986.

[JS93] K. Jeffay and D.L. Stone. Accounting for interrupt handling costs in dynamic priority task systems. In *Proc. IEEE Real-Time Systems Symposium*, Raleigh-Durham, NC, USA, 1993.

[Kai82] Claude Kaiser. Exclusion mutuelle et ordonnancement par priorité. *Technique et Science Informatiques*, 1 :59–68, 1982.

[Lab74] J. Labetoulle. Un algorithme optimal pour la gestion des processus en temps réel. *Revue Française d'Automatique*, Informatique et Recherche Opérationnelle :11–17, 1974.

[Lar04] Gaëlle Largeteau. *Quantification du taux d'invalidité d'applications temps réel à contraintes strictes*. PhD thesis, ENSMA, Université de Poitiers, 2004.

[Leh90] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–213, 1990.

[LGD05] Yacine Laalaoui, Emmanuel Grolleau, and Habiba Drias. Ant colony system for real-time scheduling with timing, precedence and exclusion constraints. In *Proc. International Arab Conference on Information Technology*, 2005.

[Liu00] J.W.S. Liu. *Real-time systems*. 2000.

[LL73] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1) :46–61, 1973.

[LM80] J. Leung and M. L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 1980.

[LRT92] J.P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks infixed-priority preemptive systems. In *Proc. Real-Time Systems Symposiu*, pages 110–123, Phoenix, AZ, 2-4 Dec 1992.

[LSS87] J.P. Lehoczky, L. Sha, and J.K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proc. IEEE Real-Time systems Symposium*, pages 261–270, 1987.

[LW82] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation(2)*, pages 237–250, 1982.

[MC96] A. Mok and D. Chen. A multiframe model for real-time tasks. In *Proc. of the 17th Real-Time Systems Symposium*, pages 22–29, Washignton, December 1996.

[MD97] A. Mok and D.Chen. A multiframe model for real-time tasks. *IEEE Transactions on software Engineering*, pages 635–645, 1997.

[Mok83] A.K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment.* PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983.

[MTN04a] J. Mäki-Turja and M. Nolin. Faster response time analysis of tasks with offsets. In *Proc 10th IEEE Real-Time Technology and Applications Symposium*, 2004.

[MTN04b] J. Mäki-Turja and M. Nolin. Tighter response time analysis of tasks with offsets. In *Proc 10th International Conference on Real-Time Computing and Applications*, 2004.

[NG03] K.H. Ngo and E. Grolleau. La méthode darts et la programmation multitâche en labview. In *Proc. FurturVIEW*, pages 69–74, Poitiers, 12-13th June 2003.

[NG07] K.H. Ngo and E. Grolleau. Dartsview, a toolkit for darts in labview. In *proc. Junior Researcher Workshop on Real-Time Computing*, Nancy, 2007.

[Ngo08] K.H. Ngo. *Aide au développement de systèmes temps réel à l'aide d'un langage graphique flots de données.* PhD thesis, ENSMA - Université de Poitiers, 2008.

[NJ09] Jifu Nong and Long Jin. A convergence proof for ant colony algorithm. In *Proc. 2009 International Joint Conference on Computational Sciences and Optimization*, pages 974–977, 2009.

[NRB09] Thi Huyen Chau NGuyen, Pascal Richard, and Enrico Bini. Approximation techniques for response-time analysis of static-priority tasks. *Real-Time Systems*, 43 :147–176, 2009.

[PGH97] J. Palencia, J. G. Garcia, and M. G. Harbour. On the schedulability analysis for distributed hardtime systems. In *Proc. of the 9th Euromicro Workshop on Real-Time Systems*, pages 136–143, Toledo, Spain, Jan. 1997.

[PH98] J. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proc 19th IEEE Real-Time System Symposium*, 1998.

[PH03] J. Palencia and M. G. Harbour. Offset-based response time analysis of distributed systems scheduled under edf. In *Euromicro conference on real-time systems*, Porto, Portugal, June 2003.

[PL05] R. Pellizzoni and G. Lipari. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems*, 30(1-2) :105–128, 2005.

[Rah08] Ahmed Rahni. *Contribution à la validation d'ordonnancement temps réel en présence de transactions sous priorités fixes et EDF*. PhD thesis, ENSMA - Université de Poitiers, 2008.

[RGR07a] A. Rahni, E. Grolleau, and M. Richard. Méthode d'évaluation du pire temps de réponse de tâches à offset. In *Proc. 5 eme Ecole d'été temps réel*, 2007.

[RGR07b] A. Rahni, E. Grolleau, and M. Richard. New worst-case response time analysis technique for realtime transactions. In *Proc. ISoLa Workshop On Leveraging Applications of Formal Methods, Verification and Validation*, Poitiers, France, December 12-14 2007.

[RGR08] Ahmed Rahni, Emmanuel Grolleau, and Michael Richard. Feasibility analysis of non-concrete real-time transactions with edf. In *proc. 16th International Conference on Real-Time and Network Systems*, Rennes, France, October 2008.

[RGR09] Ahmed Rahni, Emmanuel Grolleau, and Michael Richard. An efficient response time analysis for real-time transactions with fixed priority assignment. *Innovations in Systems et Software Engineering Journal (ISSE)*, 5(3) :197–209, 2009.

[Ric03] Pascal Richard. On the complexity of scheduling real-time tasks with self-suspension on one processor. In *Proc. 15th IEEE Int. Euromicro Conferecnce on Real-Time Systems*, 2003.

[RRGC02] M. Richard, P. Richard, E. Grolleau, and F. Cottet. Contraintes de précédence et ordonnancement mono-processeur. In *Proc. Real-Time Systems*, pages 121–138, 2002.

[SB96] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10(2) :179–210, 1996.

[Ser72] O. Serlin. Scheduling of time critical processes. In *proc. Spring Joint Computers Conference*, pages 925–932, 1972.

[SG91] T. Shepard and M. Gagne. A pre-run-time scheduling algorithm for hard real-time systems. *IEEE Trans on Software Engineering*, 17(7) :669–677, 1991.

[SH98] M. Sjodin and H. Hansson. Improved response-time analysis calculations. In *Proc. Real-Time Systems Symposium*, pages 399–408, 1998.

[SHAB03] A. Srinivasan, P. Holman, J. Anderson, and S. Baruah. The case for fair multiprocessor scheduling. In *Proc. of the 11th International Workshop on Parallel and Distributed Real-Time Systems*, 2003.

[SLNM04] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar : a flexible real time scheduling framework. *ACM SIGAda Ada Letters*, 24(4) :1–8, 2004.

[Spu96] M. Spuri. Analysis of deadline scheduled real-time systems. Technical Report RR-2772, INRIA, 1996.

[SRL90] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols : an approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9) :1175–1185, 1990.

[SS02] Kwang Mong Sim and Weng Hong Sum. Multiple ant colony optimization for network routing. In *Proc. 1st International Symposium on Cyber Worlds*, 2002.

[SSL89] B. Sprunt, L. Sha, and J.P. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1(1) :27–60, 1989.

[Sta90] P. Starke. Some properties of timed nets under the earliest firing rule. *Lecture Notes in Computer Science*, 424 :418–432, 1990.

[Str88] J.K. Strosnider. *Highly Responsive Real-Time Token Rings*. PhD thesis, Carnegie Mellon Univ., 1988.

[TC94] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess, Microprogram*, 40(2-3) :117–134, 1994.

[TGC06a] K. Traoré, E. Grolleau, and F. Cottet. Characterization and analysis of tasks with offsets : monotonic transactions. In *Proc 12th International Conference on Embedded and Real-Time Computing Systems and Applications*, Sydney, Australia, August 16-18th 2006.

[TGC06b] K. Traoré, E. Grolleau, and F. Cottet. Shedulability analysis of serial transactions. In *Real-Time and Network Systems.*, pages 141–149, Poitiers, France, May 30-31 2006.

[TGRR06] K. Traoré, E. Grolleau, A. Rahni, and M. Richard. Response-time analysis of tasks with offsets. In *12th IEEE International Conference on Emerging Technologies and Factory Automation*, 2006.

[Tin92] Ken Tindell. Using offset information to analyse static priority pre-emptively scheduled task sets. Technical Report YCS-182, Dept. of Computer Science, University of York, England, 1992.

[Tin94a] K. Tindell. *Fixed priority scheduling of hard real-time systems*. PhD thesis, University of York, England, 1994.

[Tin94b] Ken Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS-221, Dept. of Computer Science, University of York, England, 1994.

[Tra07] Karim Traoré. *Analyse et validation des applications temps Réel en présence de transactions : application au pilotage d'un drone miniature*. PhD thesis, ENSMA - Université de Poitiers, 2007.

[Ves94] S. Vestal. Fixed-priority sensitivity analysis for linear compute time models. *IEEE Transactions on Software Engineering*, 20(4) :308–317, 1994.

[XP93] J. Xu and D. Parnas. On satisfying timing constraints in hard real-time systems. *IEEE Transactions on Computers*, 19(1) :70–84, 1993.

[Xu93] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *Transactions on Software Engineering*, 19(2) :139–153, 1993.

Acronymes

ACO Ant Colony Optimization

ACS Ant Colony System

ACSRTS Ant Colony System for Real-Time Scheduling

AM Accumulativement Monotonique

AMADO Aéronef Miniature Autonome de Détection et d'Observation

ARINC 653 Avionics Application Standard Software Interface

AS Ant system

ASIC Application-Specific Integrated Circuit

B_i Pire durée de blocage, interférence maximum pouvant être appliquée à une tâche τ_i par des tâches moins prioritaires

C_i Pire durée d'exécution ou WCET d'une tâche τ_i

CN Condition nécessaire

CNS Condition nécessaire et suffisante

CS Condition suffisante

D_i Délai critique ou échéance relative d'une tâche τ_i

$d_{i,k}$ Date d'échéance de l'instance k de τ_i

DBF Demand Bound Function

DGA Délégation Générale pour l'Armement

DM Deadline Monotonic

DSP Digital Signal Processing

EDF Earliest Deadline First

EDL Earliest Deadline Last

ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechnique

FIFO First In First Out

FPGA Field-Programmable Gate Array

FPP Fixed Priority Policy

$hp(i)$ Tâches au moins aussi prioritaires que τ_i

INRIA Institut National de Recherche en Informatique et en Automatique

ISR routine de traitement d'interruption

J_i Gigue de démarrage

LAII Laboratoire d'Automatique et d'Informatique Industrielle

LEA Laboratoire d'Etudes Aérodynamiques

LISI Laboratoire d'Informatique Scientifique et Industrielle

LMS Laboratoire de Mécanique des Solides

MACO Multiple Ant Colony Optimization

ML Minimal Laxity

MMAS Max-Min Ant System

OSEK Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen

POSIX Portable Operating System Interface

r_i Date de réveil d'une tâche concrète τ_i

$r_{i,k}$ Date de réveil de l'instance k de τ_i

LL Least Laxity

NMEA National Marine Electronics Association

ONERA Office National d'Études et de Recherches Aérospatiales

RdP Réseau de Petri

RM Rate Monotonic

RMA Rate Monotonic Analysis

RTA Response Time Anayslis

RTEMS Real-Time Executive for Multiprocessor Systems

SIC Signal, Image, Communication

SoC System-on-Chip

T_i Période d'activation d'une tâche τ_i

TR_i Temps de réponse maximal de τ_i

$TR_{k,i}$ Temps de réponse de l'instance k de τ_i

UAV Unmanned Aerial Vehicle

ULB Université Libre de Bruxelles

U_i Charge de la tâche τ_i

WCET Worst-Case Execution Time