# Contribution à la conception et à la vérification de systèmes temps réel - Focus sur l'ordonnancement temps réel

Pierre-Emmanuel Hladik

Institut National Polytechnique de Toulouse

Numéro d'Ordre : Année : 2016

Habilitation à diriger les recherches

préparée au

Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

présentée et soutenue publiquement,

le 21/11/2016, par

Pierre-Emmanuel Hladik

Maître de conférences en informatique à

Institut National des Sciences Appliquées de Toulouse

Titre :

# Contribution à la conception et à la vérification de systèmes temps réel

—

## Focus sur l'ordonnancement temps réel

Jury :

| | | |
|---|---|---|
| Rapporteurs : | L. George | Professeur, ESIEE Noisy le Grand |
| | E. Grolleau | Professeur des universités, ENSMA Poitiers |
| | G. Lipari | Professeur des universités, Université Lille 1 |
| Examinateurs : | J.-C. Fabre | Professeur des universités, INP Toulouse |
| | N. Navet | Professeur associé, Université de Luxembourg |
| | F. Singhoff | Profresseur des universités, Université de Bretagne Occidentale |

# Table des matières

# Préambule

Ce document présente les activités de recherche que j'ai menées depuis 2004, date d'obtention de mon doctorat. Il est structuré en deux parties. La première est consacrée à mon parcours ainsi qu'à mes perspectives de recherche alors que la seconde est une compilation des travaux pubiés.

La première partie comprend deux chapitres :

— Le chapitre 1 est un curriculum vitæ détaillé qui présente ma formation, les différents postes occupés, mon implication dans la communauté scientifique ainsi que les productions résultantes.

— Le chapitre 2 présente mes activités de recherche du point de vue de l'ordonnancement temps réel. Des rappels sur ce domaine sont exposés, suivis par un résumé de mes travaux sous une forme thématique. La dernière section du chapitre est consacrée aux perspectives de travail et aux projets que je souhaite réaliser dans le futur.

La seconde partie du document comprend six chapitres et présente de manière détaillée les travaux que j'ai réalisés et encadrés. Ces présentations sont des synthèses des publications :

— Le chapitre 3 se focalise sur l'utilisation de la programmation par contraintes pour résoudre un probléme d'allocation et d'ordonnancement pour un système temps réel distribué. Deux approches sont proposées : la première s'inscrit dans le cadre de le décomposition de Benders et la seconde consiste à produire une contrainte globale dédiée à l'ordonnancement temps réel en-ligne. Une comparaison expérimentale des deux méthodes est réalisée et une étude complémentaire est menée pour aider le concepteur à revoir sa conception si aucune solution n'est faisable.

— Le chapitre 4 présente le travail réalisé par Ahmad Al Sheikh au cours de sa thèse. La problématique couverte est celle de la configuration d'une plate-forme avionique IMA. Une méthode est définie pour allouer automatiquement les partitions avioniques sur les unités de traitements et pour dimensionner les messages sur un réseau AFDX afin de minimiser la consommation de la bande passante.

— Le chapitre 5 définit un modèle de tâches ordonnancées pour des applications automobiles suivant le standard AUTOSAR. Une méthode d'analyse d'ordonnançabilité pour ce modèle est produire afin de couvrir l'ensemble des spécifications du standard. Son évaluation est menée et comparée à d'autres approches approximées.

— Le chapitre 6 décrit une méthode basée sur le model-checking pour vérifier des propriétés temporelles d'un système ordonnancé. Une première partie se consacre à l'utilisation d'un langage spécifique et model-checkable pour modéliser et vérifier des systèmes temps réels. Puis, dans une second partie, des règles de réécriture sont définies pour passer d'un modèle AADL à une représentation formelle et vérifiable. Cette étude montre les limites de l'approche et le manque de sémantique de certains éléments d'AADL.

— Le chapitre 7 expose une partie du travail réalisé par Maxime Chéramy au cours de sa thèse. Une première partie présente les motivations et les choix architecturaux qui ont conduit au développement d'un simulateur d'ordonnancement temps réel. Puis, une seconde partie est consacrée aux évaluations menées à partir de ce simulateur afin de comparer les divers algorithmes d'ordonnancement.

— Le chapitre 8 regroupe deux travaux réalisés sur la prise en considération des incertitudes dans l'ordonnancement. La première partie se consacre à une méthode d'échantillonnage pour accélérer le calcul probabliste des temps de réponse de tâches temps réel. La seconde partie est la suite du travail de thèse de Maxime Chéramy sur l'évaluation de l'impact des mémoires caches sur les performances des ordonnanceurs temps réel multiprocesseurs.

Première partie

# Synthèse des activités de recherche

CHAPITRE 1

# Curriculum Vitæ

**Pierre-Emmanuel, Jaromir Hladik**
Né le 27 décembre 1977 à Angers (Maine et Loire)

## 1.1 Formation et parcours professionnel

### Formation universitaire

| | |
|---|---|
| **2001-2004** | **Doctorat de l'Université de Nantes** |
| *Spécialité* | Automatique et Informatique Appliquée |
| **2000-2001** | **Diplôme d'Études Approfondies de l'École Centrale de Nantes** |
| *Spécialité* | Automatique et Informatique Appliquée, option informatique |
| **1997-2000** | **Ingénieur généraliste de l'École Centrale de Nantes** |
| *Spécialité* | Automatique, option robotique |

### Cursus professionnel universitaire

| | |
|---|---|
| **depuis 2007** | **Maître de conférences** |
| *Section* | Génie informatique, automatique et traitement du signal (61) |
| *Établissement* | Institut National des Sciences Appliquées de Toulouse |
| *Département* | Génie Électrique et Informatique |
| **2006-2007** | **Maître assistant associé** (Ministère de l'industrie) |
| *Établissement* | École des Mines de Nantes |
| *Département* | Informatique |
| **2004-2006** | **Attaché Temporaire d'Enseignement et de Recherche** |
| **2001-2004** | **Moniteur de l'enseignement supérieur** |
| *Établissement* | Institut Universitaire de Technologie de Nantes |
| *Département* | Qualité, Logistique Industrielle et Organisation |

## Laboratoires de rattachement

| | |
|---|---|
| **depuis 2007** | **Laboratoire d'Architecture et d'Analyse des Systèmes** |
| **LAAS-CNRS** | UPR CNRS 8001, INSIS, INS2I |
| *Équipe* | Vérification des systèmes temporisés critiques (Vertics) |
| **2006-2007** | **Laboratoire d'Informatique de Nantes-Atlantique** |
| **LINA** | UMR CNRS 6241 |
| *Équipe* | Contraintes |
| **2001-2006** | **Inst. Recherche en Communications et Cybernétique de Nantes** |
| **IRCCyN** | UMR CNRS 6597 |
| *Équipe* | Systèmes temps réel (STR) |

## 1.2 Expériences d'encadrement en recherche

### Encadrements de thèses de doctorat

| | |
|---|---|
| **2011-2014** | **Maxime Chéramy** |
| | *Thèse de doctorat de l'Université de Toulouse, École doctorale EDSYS* |
| *Titre* | Évaluation et mise en œuvre des politiques d'ordonnancement temps réel multicoeur |
| *Soutenance* | 11 décembre 2014 |
| *Jury* | Président : J.-C. Fabre (PU, INPT), Rapporteurs : P. Richard (PU, Univ. Poitiers), L. Cucu-Grosjean (CR, HDR, INRIA), Examinateurs : Y. Trinquet (PU, Univ. Nantes), Laurent George (PU, ESIEE) |
| *Encadrement* | 50% avec Anne-Marie Déplanche (MC, Univ. Nantes, IRCCyN) |
| *Financement* | Projet ANR RESPECTED |
| *Publications* | 2 revues internationales (Al Sheikh *et al.*, 2013; Chéramy *et al.*, 2015a), 2 conférences internationales (Chémary *et al.*, 2013; Chéramy *et al.*, 2014b), 3 workshops (Chéramy *et al.*, 2013, 2014a; Chéramy *et al.*, 2015b). |
| **2008-2011** | **Ahmad Al Sheikh** |
| | *Thèse de doctorat de l'Université de Toulouse, École doctorale EDSYS* |
| *Titre* | A decision support system for distributed real-time scheduling |
| *Soutenance* | 28 Septembre 2011 |
| *Jury* | Président : F. Boniol (ONERA, France), Rapporteurs : S. Baruah (Univ. of North Carolina, US), Y. Sorel (INRIA, France), Examinateur : J. Goosens (Univ. libre de Bruxelle, Belgique) |
| *Encadrement* | 50% avec Olivier Brun (CR, HDR, LAAS-CNRS) |
| *Financement* | Projet ANR SATRIMMAP |
| *Publications* | 2 revues internationales (Al Sheikh *et al.*, 2013, 2012), 3 conférences internationales Al Sheikh *et al.* (2011c,b, 2010), 1 conférence nationale (Al Sheikh *et al.*, 2011a), 1 workshop (Al Sheikh *et al.*, 2009). |

### Futur encadrement de thèses de doctorat

| | |
|---|---|
| **2017-2020** | **Thèse CIFRE à pourvoir en collaboration avec Continental** |
| | *Thèse de doctorat de l'Université de Toulouse, École doctorale EDSYS* |
| *Titre* | Génération automatique de code embarqué |
| *Encadrement* | Silvano Dal Zilio (50/3%), Pierre-Emmanuel Hladik (50/3%), Felix Ingrand (50/3%) |
| *Financement* | Agence de l'environnement et de la maîtrise de l'énergie (ADEME) |

### Encadrement de stages de niveau Master

**2015** **Raphaël Roy** (formation ingénieur INSA Toulouse)
*Sujet :* Simulink Parrot AR Drone Support improvements. *Encadrement :* 50% avec Élodie Chanthery (MC, LAAS-CNRS). *Durée :* 4 mois.

**Josué Alvarez** (formation ingénieur INSA Toulouse)
*Sujet :* Interface web pour la simulation d'ordonnancement. *Encadrement :* 50% avec Maxime Chéramy (Dr., LAAS-CNRS). *Durée :* 2 mois.

**Thomas Besson** (formation ingénieur INSA Toulouse)
*Sujet :* Application de contrôle d'un robot sur une architecture embarquée. *Encadrement :* 100%. *Durée :* 4 mois.

**2014** **Mohamed Amin Aouadhi** (formation ingénieur ENSI Tunis)
*Sujet :* Développement d'une transformation d'un modèle MCSE dans le langage FIACRE. *Encadrement :* 100%. *Durée :* 6 mois.

**Thomas Ridremont** (Master IMA, UCO Angers)
*Sujet :* Résolution des problèmes d'optimisation bi-niveau mixtes. *Encadrement :* 50% avec Nicolas Jozewofiez (MC, LAAS-CNRS). *Durée :* 3 mois.

**Oussama El Fatayri** (formation ingénieur INSA)
*Sujet :* Implémentation et expérimentation de tests d'ordonnançabilité pour des applications AUTOSAR. *Encadrement :* 100%. *Durée :* 3 mois.

**Olivier Hachette** (formation ingénieur INSA)
*Sujet :* Développement d'une bibliothèque libre d'une IMU pour AR.Drone. *Encadrement :* 100%. *Durée :* 2 mois.

**Sébastien Chazalon** (formation ingénieur INSA)
*Sujet :* Étude des performances d'un systèmes d'exploitation temps réel pour drone. *Encadrement :* 100%. *Durée :* 2 mois.

**2013** **Nicolas Kniebihli** (formation ingénieur ENSEEIHT)
*Sujet :* Développement de drivers pour capteurs et actionneurs d'un drone. *Encadrement :* 50% avec Jérôme Ermont (MC, IRIT). *Durée :* 3 mois.

**Clément Filleau** (formation ingénieur ENSEEIHT)
*Sujet :* Transformation et validation d'un modèle MCSE en FIACRE. *Encadrement :* 100%. *Durée :* 3 mois.

**2012** **Maël Zoungrana** (formation ingénieur INSA)
*Sujet :* Déploiement d'une application critique sur drone. *Encadrement :* 100%. *Durée :* 3 mois.

**2011** **Alan Diego Pontizelli** (Univ. Federal de Santa Catarina – Brésil)
*Sujet :* Verification of AADL Specifications with Formal Methods. *Encadrement :* 100%. *Durée :* 6 mois.

**Younes Maaouni** (M2R SAID, Univ. Toulouse)
*Sujet :* Simulation d'ordonnancement temps réel sur du multicoeur. *Encadrement :* 100%. *Durée :* 5 mois.

**Larissa Calsavara.** (Univ. Federal de Santa Catarina – Brésil)
*Sujet :* Modélisation et vérification stochastique en réseau de Petri. *Encadrement :* 100%. *Durée :* 6 mois.

**2010** **Houssen Alaeddine** (M2R Univ. Rouen)
*Sujet :* Méthodes de recherche locale pour l'ordonnancement avionique. *Encadrement :* 50% avec Olivier Brun (CR, LAAS-CNRS). *Durée :* 4 mois

**Vikas Shukla** (formation ingénieur INSA)
*Sujet :* Stochastic Verification of Real-Time Scheduling. *Encadrement :* 100%. *Durée :* 4 mois

**Manel Blaghji** (élève ingénieur ENSI Tunis)
*Sujet :* Résolution d'un problème d'ordonnancement périodique à l'aide de la programmation par contraintes. *Encadrement :* 50% avec Patrick Esquirol (MC, LAAS-CNRS). *Durée :* 5 mois

**Victor Boeing Ribeiro** (Univ. Federal de Santa Catarina –Brésil)
*Sujet :* Using Formal Verification to Coordinate Multi-Robot Systems. *Encadrement :* 50% avec S. Dal Zilio (CR, LAAS-CNRS). *Durée :* 6 mois

**2009** **Xiamu Shi** (Master Embedded System, ISAE)
*Sujet :* AADL to POLA Transformation using EMF. *Encadrement :* 100%. *Durée :* 5 mois

**Boussad Khelfaoui** (Master SAID, Univ. Toulouse)
*Sujet :* Evaluation des méthodes de partitionnement temps réel. *Encadrement :* 100%. *Durée :* 5 mois

**Khaled Refaat** (Univ. du Caire (Égypte))
*Sujet :* Prediction for Real-Time Stochastic Scheduling. *Encadrement :* 100%. *Durée :* 3 mois

**2007** **Dalia Aoun** (Master ASP, Univ. de Nantes)
*Sujet :* Ordonnancement temps réel multiprocesseur. *Encadrement :* 50 % avec Anne-Marie Déplanche (MC, Univ. de Nantes). *Durée :* 5 mois

**2005** **Grégory Beaumet** (Master ASP, Univ. de Nantes)
*Sujet :* Mise en place d'un environnement d'analyse de performances pour des algorithmes de vérification d'ordonnançabilité. *Encadrement :* 50% avec Anne-Marie Déplanche. *Durée :* 5 mois

**Grégoire Ballet** (Master ASP, Univ. de Nantes)
*Sujet :* Techniques analytiques de vérification de l'ordonnançabilité pour des algorithmes d'ordonnancement à priorités dynamiques. *Encadrement :* 50% avec Anne-Marie Déplanche. *Durée :* 5 mois

## 1.3 Rayonnement scientifique

### Congé pour Recherches ou Conversion Thématique (CRCT)

**2015-2016** Obtention par le CNU d'un congé pour recherche d'un an
*Établissement* École de Technologie Supérieure (ÉTS), Montréal, Québec, Canada.
*Laboratoire* Laboratoire en Architecture de Systèmes Informatiques (LASI)

• Au cours du CRCT, j'ai eu l'occasion de travailler sur des méthodes de vérification et d'aide à la décision pour la génération des architectures opérationnelles temps réel, donnant lieu à deux publications (Hladik *et al.*, 2016; Beji *et al.*, 2016). Cette expérience devrait déboucher sur une collaboration plus formelle avec le professeur Abdelouahed Gherbi de l'ÉTS en particulier par la venue de celui-ci pour un congé sabbatique en 2017 et par le dépôt d'un projet binational. Cette expérience a aussi été l'occasion de collaborer avec des chercheurs de l'École polytechnique de Montréal, en particulier David Saussié, Giovanni Beltrame et Francis Giraldeau.

### Prime d'excellence scientifique (PES)

**2013-2017** Obtention de la prime d'Excellence scientifique

### Participation à l'organisation de conférences

**2015** Organisateur de la troisième école d'été de la chaire CESEC avec pour thème les systèmes cyber-physiques

**2010-2015** Membre du comité de programme de la conférence Real-Time and Network Systems (RTNS)

**depuis 2009** Membre du comité de programme des journées formalisation des activités concurrentes (FAC)

**2009 - 2011** Membre du comité de programme et co-organisateur de la session ordonnancement temps réel à la conférence ROADEF 2009 et 2011

### Expert pour des instances d'évaluation

**2012 - 2014** Expert auprès du Ministère des Affaires Étrangères dans le cadre de la coopération franco-marocaine dans le domaine des Systèmes Embarqués (appel à projet SCAC)

**2012** Expert pour l'ANR, programme blanc international SIMI 2

**2011** Expert pour le dispositif CIFRE dans le domaine systèmes embarqués

### Rapporteur pour des revues et conférences internationnales

— Rapporteur pour les conférences nationales ROADEF, FAC

— Rapporteur pour les revues Journal of Systems and Software (JSS), Journal of Scheduling et IEEE Transaction on Parallel and Distributed Systems

— Rapporteur pour la revue francophone Journal Européen des Systèmes Automatisés (JESA)

— Rapporteur pour diverses conférences internationales : Real-Time and Network Systems (RTNS), Formal Methods (FM), Emerging Technologies and Factory Automation (ETFA), Reliable Software Technologies - AdaEurope, Modélisation, Optimisation et SIMulation (MOSIM)

### Membre de jury de thèse

| | |
|---|---|
| **mar. 2013** | Examinateur dans le jury de thèse de **Naeem Shehzad**. |
| *Établissement* | Université de Nantes, IRCCyN |
| *Titre* | Overhead control in optimal scheduling algorithms for real-time multiprocessor systems |
| *Directeurs* | Yvon Trinquet (PU, Univ. Nantes), Anne-Marie Déplanche (MC, Univ. Nantes) |

| | |
|---|---|
| **fév. 2013** | Examinateur dans le jury de thèse de **Florian Many**. |
| *Établissement* | Université de Toulouse, ONERA |
| *Titre* | Combinaison des aspects temps réel et sûreté de fonctionnement pour la conception des plate-formes avioniques |
| *Directeurs* | Frédéric Boniol (PU, ONERA), David Doose (IR, ONERA) |

| | |
|---|---|
| **avr. 2012** | Examinateur dans le jury de thèse de **Mikel Cordovilla**. |
| *Établissement* | Université de Toulouse, ONERA |
| *Titre* | Environnement de développement d'applications multipériodiques sur plateforme multicœur. La boîte à outil SchedMCore |
| *Directeurs* | Frédéric Boniol (PU, ONERA), Eric Noulard (IR, ONERA), Claire Pagetti (IR, ONERA) |

### Membre de comité de suivi de thèse

| | |
|---|---|
| **2016-2019** | Membre du comité de suivi de **Khaoula Boukir**. |
| *Établissement* | Université de Nantes, IRCCyN |
| *Titre* | Politiques d'ordonnancement temps réel multiprocesseur prouvées |
| *Directeur* | Jean-Luc Béchennec (CR, IRCCyN), Anne-Marie Déplanche (MC, IRCCyN) et Olivier H. Roux (PU, IRCCyN) |

### Soutiens financiers et valorisation - Projets et contrats

| | |
|---|---|
| **2016-2017** | Banc de test informatique pour CubeSat |
| *Rôle* | Participant |
| *Financement* | TTIL |
| *Durée* | 18 mois à partir de septembre 2016 |
| *Participants* | LAAS-CNRS (équipes TSF et Vertics) |
| **2015** | Simulink – Parrot AR Drone Support improvements |
| *Rôle* | Responsable scientifique et technique pour le LAAS-CNRS |
| *Financement* | Mathworks |
| *Participants* | LAAS-CNRS (équipes DISCO et Vertics) |
| **2010-2014** | Real-time executive support with scheduling policies for thermally-constrained multicore embedded systems (RESPECTED) |
| *Rôle* | Responsable scientifique et technique pour le LAAS-CNRS |
| *Financement* | ANR, programme ARPEGE |
| *Participants* | IRCCyN, See4Sys, LAAS-CNRS, LEAT |
| **2010-2011** | Ordonnancement dans les systèmes embarqués critiques (OSEC) – Incertitude sur la demande et robustesse |
| *Rôle* | Responsable scientifique |
| *Financement* | Ressource propre du LAAS-CNRS |
| *Participants* | LAAS-CNRS (équipes MOGISA, MRS et OLC) |
| **2007-2011** | Safety & Time Critical Middleware for future Modular Avionics Platform (SATRIMMAP) |
| *Rôle* | Responsable scientifique et technique pour le LAAS-CNRS |
| *Financement* | ANR, appel Technologies Logicielles |
| *Participants* | CEA-LIST, IRIT, LAAS-CNRS, ONERA, AIRBUS, QoS Design |
| **2006-2007** | Résolution d'un problème de placement de tâches temps réel à l'aide de la programmation par contraintes |
| *Rôle* | Participant |
| *Financement* | Projet AtlanSTIC |
| *Participants* | IRCCyN, LINA |

## Responsabilités dans les instances locales

**2012-2015** Membre du comité opérationnel de la chaire systèmes embarqués critiques (CESEC)
• Chaire financée par la fondation EADS pour mener des actions de formation universitaire au sein des écoles INSA, ISAE et INP-ENSEEIHT, dans le domaine des systèmes embarqués.

**2014** Co-responsable du pôle systèmes embarqués de l'INSA de Toulouse pour la mise en place de l'INSA Euro-Méditerranée
• Définition du programme d'enseignement dans les domaines des systèmes embarqués pour la futur INSA Euro-Méditerranée.

**depuis 2013** Correspondant des relations internationales du département de génie électrique et informatique à l'INSA de Toulouse
• Gestion des départs à l'étranger et des arrivées des étudiants d'échange (environ 70 étudiants par an).

**depuis 2011** Membre du comité de recrutement du département de génie électrique et informatique à l'INSA de Toulouse pour les sections 27, 61 et 63
• Constitution de la commission pour le recrutement (ATER, MC, PU) au département de Génie Électrique et Informatique (sections 27, 61 et 63).

**depuis 2009** Membre du jury de l'INSA de Toulouse du dispositif de validation des acquis de l'expérience (VAE) dans le domaine informatique

**2007-2010** Responsable de la quatrième année informatique au département de génie électrique et informatique de l'INSA de Toulouse
• Organisation des programmes, de l'emploi du temps et des stages pour 60 étudiants niveau M1.

## 1.4 Publications

**La liste des publications antérieures au rattachement au LAAS (avant 2007) est disponible sur `http://homepages.laas.fr/pehladik/` et depuis 2007 sur le site du LAAS (`http://www.laas.fr`).**

### Résumé

Les rapports de recherche ne figurent pas dans ce résumé.

| Année | 99 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | Tot. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Recherche** | | | | | | | | | | | | | | | | |
| Revues int. | | | | | | | 1 | | | 1 | 1 | 1 | | 1 | | 5 |
| Conf. int. | | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | | 1 | 2 | 1 | 1 | 22 |
| Conf nat. | | 1 | 1 | | | 1 | | 2 | 1 | 1 | | 1 | | | 1 | 9 |
| **Pédagogie** | | | | | | | | | | | | | | | | |
| Revue nat. | | | | | | | | | | | | 2 | | | 1 | 3 |
| Conf. nat. | | | | | | | | | | | | 1 | | | | 1 |
| Livres | 2 | | | | | | | 1 | | | | | | | 1 | 4 |

Les publications signalées par le symbole ★ signifie que les auteurs sont cités par ordre alphabétique. Le symbole ✠ signale une publication commune avec un doctorant ou un stagiaire que j'ai encadré.

### Revues internationales

1. (✠) Maxime Chéramy, Pierre-Emmanuel Hladik et Anne-Marie Déplanche : Algorithmes pour l'ordonnancement temps réel multiprocesseur. *Journal Européen des Systèmes Automatisés*, 48(7-8), 2015a **[référencé Scopus]**
2. (★,✠) Ahmad Al Sheikh, Olivier Brun, Maxime Chéramy et Pierre-Emmanuel Hladik : Optimal design of virtual links in AFDX networks. *Real-Time Systems*, 49(3), 2013 **[référencé WoS, JCR, dblp]**
3. (★,✠) Ahmad Al Sheikh, Olivier Brun, Pierre-Emmanuel Hladik et Balakrishna J. Prabhu : Strictly periodic scheduling in IMA-based architectures. *Real-Time Systems*, 48(4), 2012 **[référencé WoS, JCR, dblp]**
4. Florent Péres, Pierre-Emmanue Hladik et François Vernadat : Specification and verification of real-time systems using POLA. *International Journal of Critical Computer-Based Systems*, 2(3/4):332–351, 2011 **[référencé dblp]**
5. Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche et Narendra Jussien : Solving a real-time allocation problem with constraint programming. *Journal of Systems and Software*, 81(1), 2008 **[référencé JCR, dblp]**

### Conférences internationales

Les conférences ECRTS, FORMATS, RTNS, CP ont des taux d'acceptations inférieures à 50% (voire de moins de 25% pour ECRTS et CP).

1. Sofien Beji, Abdelouahed Gherbi, John Mullins et Pierre-Emmanuel Hladik : Model-driven approach to the optimal configuration of time-triggered flow within a ttethernet network. *In Proc. of the 9th System Analysis and Modelling Conference*, SAM, 2016
2. (✠) Maxime Chéramy, Pierre-Emmanuel Hladik, Anne-Marie Déplanche et Silvano Dal Zilio : Simulation of real-time scheduling algorithms with cache effects. *In Proc. of the 6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems*, WATERS, 2015b
3. (✠) Maxime Chéramy, Pierre-Emmanuel Hladik et Anne-Marie Déplanche : Simso: A simulation tool to evaluate real-time multiprocessor scheduling algorithms. *In Proc. of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, WATERS, 2014a
4. (✠) Maxime Chéramy, Pierre-Emmanuel Hladik, Anne-Marie Déplanche et Sébastien Dubé : A flexible simulator for real-time scheduling. *In Proc. of the Work-In-Progess of the 9th IEEE International Symposium on Industrial Embedded Systems*, SIES, 2014b
5. (★,✠) Maxime Chémary, Anne-Marie Déplanche et Pierre-Emmanuel Hladik : Simulation of real-time multiprocessor scheduling with overheads. *In Proc. of the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, SIMULTECH, 2013 [**référencé dblp**]
6. (★,✠) Ahmad Al Sheikh, Olivier Brun, Pierre-Emmanuel Hladik et Balakrishna J. Prabhu : Strictly periodic scheduling on an IMA-based avionic platform. *In Proc. of the 15th Austrian-French-German conference on Optimisation*, AFG, 2011c
7. (★,✠) Ahmad Al Sheikh, Olivier Brun, Pierre-Emmanuel Hladik et Balakrishna J. Prabhu : A best-response algorithm for periodic scheduling. *In Proc. of the 23rd Euromicro Conference on Real-Time Systems*, ECRTS, 2011b [**référencé dblp**]
8. (★,✠) Ahmad Al Sheikh, Olivier Brun et Pierre-Emmanuel Hladik : Partition scheduling on an IMA platform with strict periodicity and communication delays. *In Proc. of the 18th International Conference on Real-Time and Network Systems*, RTNS, 2010
9. (✠) Khaled S. Refaat et Pierre-Emmanuel Hladik : Efficient stochastic analysis of real-time systems via random sampling. *In Proc. of the 22nd Euromicro Conference on Real-Time Systems*, ECRTS, 2010 [**référencé dblp**]
10. (★,✠) Ahmad Al Sheikh, Olivier Brun et Pierre-Emmanuel Hladik : Decision support for task mapping on IMA architecture. *In Proc. of the 3rd Junior Researcher Workshop on Real-Time Computing*, JRWRTC, 2009
11. Florent Péres, Pierre-Emmanue Hladik et François Vernadat : Specification and verification of real-time systems using the POLA tool. *In Proc. of the 3rd International Workshop on Verification and Evaluation of Computer and Communication Systems*, VECoS, 2009b
12. Frédéric Boniol, Pierre-Emmanuel Hladik, Claire Pagetti, Frédéric Aspro et Victor Jégu : A framework for distributing real-time functions. *In Proc. of the 6th International Conference on Formal Modeling and Analysis of Timed Systems*, FORMATS, 2008 [**référencé WoS**]
13. Sébastien Faucou, Pierre-Emmanuel Hladik, Anne-Marie Déplanche et Yvon Trinquet : Overview of microkernel standards for real-time in-vehicle embedded systems. *In Proc. of the 4th Taiwanese-French Conference on Information Technology*, TFIT, 2008

14. Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Sébastien Faucou et Yvon Trinquet : Adequacy between AUTOSAR OS specification and real-time scheduling theory. *In Proc. of the 2nd International Symposium on Industrial Embedded Systems*, SIES, 2007a **[référencé dblp]**
15. Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Sébastien Faucou et Yvon Trinquet : Schedulability analysis of OSEK/VDX applications. *In Proc. of the 15th International Conference on Real-Time and Network Systems*, RTNS, 2007b
16. Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche et Narendra Jussien : Solving allocation problems of hard real-time systems with dynamic constraint programming. *In Proc. of the 14th International Conference on Real-time and Network Systems*, RTNS, 2006b
17. Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche et Narendra Jussien : Guiding architectural design process of hard real-time systems with constraint programming. *In Proc. of the 3rd Taiwanese-French Conference on Information Technology*, TFIT, 2006a
18. Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche et Narendra Jussien : How solve allocation problems with constraint programming. *In Proc. of the Work-In-Progress Session of the 17th Euromicro Conference on Real-Time Systems*, WiP ECRTS, 2005b
19. Hadrien Cambazard, Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Narendra Jussien et Yvon Trinquet : Decomposition and learning for a hard real time task allocation problem. *In Proc. of the Principles and Practice of Constraint Programming*, CP, 2004a **[référencé dblp]**
20. Hadrien Cambazard et Pierre-Emmanuel Hladik : Learning and cooperation for a hard real-time task allocating problem. *In Proc. of the 46th Annual conference of the Canadian Operational Research Society & The Institute for Operations Research and the Management Sciences International*, CORS-INFORMS, 2004
21. Pierre-Emmanuel Hladik et Anne-Marie Déplanche : Best-case response time analysis for precedence relations in hard real-time systems. *In Proc. of the Work-In-Progress Session of the 24th IEEE International Real-Time Systems Symposium*, WiP RTSS, 2003a
22. Pierre-Emmanuel Hladik et Anne-Marie Déplanche : An extension of holistic schedulability analysis for precedence relations in multiprocessor systems. *In Proc. of the Work-In-Progress Session of the 15th Euromicro Conference on Real-Time Systems*, WiP ECRTS, 2003b

### Conférences nationales

1. Pierre-Emmanuel Hladik, Silvano Dal Zilio, Olivier Pasquier, Sébastien Pillement et Bernard Berthomieu : Outillage pour la modélisation, la vérification et la génération d'applications temporisées et embarquées. *In 15èmes journées Approches Formelles dans l'Assistance au Développement de Logiciels*, AFADL, 2016
2. (★, ✠) Maxime Chéramy, Anne-Marie Déplanche et Pierre-Emmanuel Hladik : Simulation d'ordonnancement temps réel avec prise en compte de l'impact des caches. *In Proc. of the École d'été Temps Réel 2013*, ETR, 2013
3. (★, ✠) Ahmad Al Sheikh, Olivier Brun et Pierre-Emmanuel Hladik : Ordonnacement de tâches sous contrainte de périodicité stricte. *In Proc. of the 12e Conférence de la société Française de Recherche Opérationnelle et Aide à la Décision*, ROADEF, 2011a

4. (✠) Pierre-Emmanue Hladik, Florent Péres et Xiaomu Shi : Analyse d'un modële AADL à l'aide de pola. *In Proc. of the 10e journée francophone sur les Approches Formelles dans l'Assistance au Développement de Logiciels*, AFADL, 2010
5. Pierre-Emmanuel Hladik et Anne-Marie Déplanche : Ordonnancement temps réel multiprocesseur partitionné et programmation par contraintes. *In Proc. of the 10e Conférence de la société FranÁaise de Recherche Opérationnelle et Aide à la Décision*, ROADEF, 2009
6. Florent Péres, Pierre-Emmanue Hladik et François Vernadat : POLA: un langage dédié au domaine des systëmes temps réel vérifiables par model cheking. *In Proc. of the 10e Conférence de la société FranÁaise de Recherche Opérationnelle et Aide à la Décision*, ROADEF, 2009a
7. Hadrien Cambazard, Pierre-Emmanuel Hladik, Anne-Marie Déplanche et Narendra Jussien : Deux approches pour la résolution d'un problëme d'allocation de tâches en temps-réel dur. *In Proc. of the 3e Journées Francophones de Programmation par Contrainte*, JFPC, 2007
8. Hadrien Cambazard, Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Narendra Jussien et Yvon Trinquet : Décomposition et apprentissage pour un problème d'allocation de tâches temps réel. *In Proc. des Journées Nationales sur la résolution Pratique de Problèmes NP-Complets*, JNPC, 2004b
9. Pierre-Emmanuel Hladik et Anne-Marie Déplanche : Ordonnancement préemptif à priorités fixes : Comparaison de méthodes analytiques de calcul de pire temps de réponse. *In Proc. of the 11th Real-Time and Embedded Systems*, RTS, 2003c

### Revues pédagogiques nationales avec sélection

1. Élodie Chanthery, Gwendoline Le Corre et Pierre-Emmanuel Hladik : De l'illustration du guidage à l'optimisation d'un plan par un robot Lego Mindstorm NXT. *Journal sur l'enseignement des sciences et technologies de l'information et des systèmes*, 2016
2. Thierry Rocacher, Vincent Mahout et Pierre-Emmanuel Hladik : Un jeu de tir laser ou comment mélanger le traitement de signal, l'électronique et l'assembleur. *Journal sur l'enseignement des sciences et technologies de l'information et des systèmes*, 12, 2013
3. Guillaume Auriol, Vincent Mahout, Thierry Rocacher, Pascal Acco et Pierre-Emmanuel Hladik : La conception orientée objet au secours de la programmation de microcontrôleur ou inversement... *Journal sur l'enseignement des sciences et technologies de l'information et des systèmes*, 12, 2013

### Conférences pédagogiques nationales

1. Pierre-Emmanuel Hladik et Sébastien Di Mercurio : Plate-forme d'enseignement pour la conception et l'implémentation sur exécutif temps réel. *In Proc. of the Conférence sur l'Enseignement des Technologies et des Sciences de l'Information et des Systèmes*, CETSIS, 2013

### Livres pédagogiques

1. Jean Hladik et Pierre-Emmanuel Hladik : *Quaternions réels, duaux et complexes.* Ellipse, 2016. ISBN 978-2340010468
2. Jean Hladik, Michel Chrysos, Pierre-Emmanuel Hladik et Lorenzo Ugo Ancarani : *Mécanique quantique : Atomes et noyaux, applications technologiques - cours et exercices corrigés.* Dunod, 2009a. ISBN 978-2100521883
3. Jean Hladik et Pierre-Emmanuel Hladik : *Calcul tensoriel en physique, cours et exercices corrigés.* Dunod, 1999. ISBN 978-2100040711
4. Pierre-Emmanuel Hladik : *Formulaire de l'étudiant: Physique.* Ellipses Marketing, 1999. ISBN 978-2729899172

## 1.5 Enseignements dispensés

Depuis mon recrutement à l'INSA de Toulouse en 2007, mes enseignements se répartissent principalement entre les Unités de Formation (UF) présentées dans la table 1.1 (le symbole ✠ indique que je suis actuellement responsable de l'UF).

| Code | Intitulé | Niveau | Heures étudiantes |
|---|---|---|---|
| I1ANIF11E2<br>I1ANSY21E2 | Algorithmique | 1e année | TP 26h |
| I3MAIF21 | Langage d'assemblage | 3e année | TP 16,5h |
| I3MITC22 | Concepts et hardware pour la transmission d'informations | 3e année | TP 22h |
| I4AEIM11✠<br>I4IRIF31 | Informatique matérielle | 4e année | CM 5h TD 8,75h TP 41,25h |
| I4AEIL11E2<br>I4IRTR11E3✠ | Systèmes temps réel | 4e année | CM 8,75h TD 2,5h TP 13,75h |
| I5AISE11E2✠ | Ordonnancement et systèmes d'exploitation pour l'embarqué | 5e année | CM 11,25h |
| I5AISE31<br>I5AISE41✠ | Projet : conception de systèmes embarqués critiques | 5e année | CM 15h, TP 70h |
| I4AEPJ11<br>I4IRJ11 | Projet de recherche tutoré | 4e année | TD 30h |

Table 1.1 – Unités de formation dans lesquelles j'assure des enseignements à l'INSA de Toulouse

Les volumes horaires indiqués représentent les heures étudiantes. Les volumes horaires des enseignements que j'effectue dans chaque UF ne sont pas précisés car ils évoluent d'une année à l'autre en fonction des différentes charges administratives (voir partie 1.3) et des besoins à couvrir.

Depuis juin 2016, je supervise la réalisation du MOOC « Programmation en C pour l'embarqué » dans le cadre du projet Connect-IO financé par l'ANR (IDEFI numérique). Il devrait être finalisé pour décembre 2016.

CHAPITRE 2

# Présentation synthétique de mes travaux passés et futurs

Les travaux présentés dans ce manuscrit ont été réalisés après ma thèse de 2004 à 2006 à l'Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN) dans l'équipe « Système Temps Réel », puis, de 2006 à 2007 au Laboratoire d'Informatique de Nantes Atlantique (LINA) dans l'équipe « Contraintes » et enfin, depuis 2007 au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS) dans l'équipe « Vérification des systèmes temporisés critiques » (Vertics).

Au cours de l'année universitaire 2015-2016, dans le cadre d'un congé de recherche, j'ai eu l'opportunité de poursuivre mes travaux à l'École de Technologie Supérieure (ÉTS), Canada, dans le « Laboratoire en Architecture de Systèmes Informatiques » (LASI) et de nouer des relations avec des équipes de l'École Polytechnique de Montréal (voir 1.3).

Ces affiliations variées m'ont offert diverses opportunités de collaborations universitaires et industrielles et ont favorisé la maturation de mon travail dans le domaine des systèmes embarqués critiques. Ainsi, les résultats présentés dans ce document n'auraient pas été possibles sans les collaborations avec Christian Artigues (LAAS), Bernard Berthomieu (LAAS), Frédéric Boniol (ONERA), Olivier Brun (LAAS), Hadrien Cambazard (G-SCOP), Silvano Dal Zilio (LAAS), Anne-Marie Déplanche (IRCCyN), Sébastien Faucou (IRCCyN), Abdelouahed Gherbi (ÉTS), Narendra Jussien (LINA), Nadjia Kara (ÉTS) Claire Pagetti (ONERA), Olivier Pasquier (IETR), Florent Peres (LAAS), Sébastien Pillement (IETR), Yvon Trinquet (IRCCyN) et François Vernadat (LAAS).

Tout au long de ces années, j'ai eu aussi l'occasion d'encadrer de nombreux étudiants de niveau Master (voir partie 1.2 pour plus de détails) ainsi que deux doctorants à hauteur de 50% chacun. Ces thèses ont été financées par des projets de l'Agence Nationale de la Recherche (ANR). La première, menée par Ahmad Al Sheikh, a porté sur l'étude de l'ordonnancement et du routage des plate-formes avioniques et a été soutenue en septembre 2011. Le second doctorant, Maxime Chéramy, a défendu sa thèse en décembre 2014 sur l'impact de la mémoire cache au niveau de l'ordonnancement dans des systèmes multicœurs. Ces différents travaux ont donné lieu à des publications communes.

Mes travaux s'inscrivent dans le cadre de la recherche sur les systèmes informatiques temps réel. Afin d'en donner une perspective cohérente, j'ai délibérément opté pour une présentation articulée autour de l'ordonnancement. Un positionnement plus large est fait dans la section consacrée à mes perspectives de recherche.

Ce chapitre est constitué d'une première partie qui positionne les activités de recherche et d'ingénierie que j'ai menées en ordonnancement temps réel, suivie d'un bilan de mes travaux et de mon positionnement actuel dans le domaine. La dernière partie est consacrée à la présentation de différentes perspectives dans lesquelles je souhaite inscrire mes futurs travaux.

## 2.1 L'ordonnancement de systèmes temps réel

La notion de **système temps réel** est attachée à un système informatique réactif pour lequel « *the correctness of the system depends not only on the logical results of computation, but also on the time at which the results are produced* » (Stankovic, 1988). Il est bien entendu que ces contraintes temporelles ne sont pas les seules à devoir être considérées dans un système et qu'elles sont couplées à d'autres contraintes que nous n'aborderons qu'en marge dans ce document comme « l'embarquabilité » (capacité de calcul, place mémoire, énergie, etc.) ou la criticité (fiabilité, sécurité, etc.).

Concevoir de tels systèmes nécessite donc de prendre en considération ces contraintes de temps et d'en garantir leur respect. Ainsi le système doit être **prévisible** (*predictable*), c'est-à-dire que l'on doit pouvoir prédire son comportement vis-à-vis des propriétés temporelles attendues.

Dans le domaine de l'ordonnancement, un système temps réel est modélisé par un ensemble de tâches à exécuter sur un ensemble de processeurs. Une **tâche** contrôle le flot d'exécution d'un programme et peut être récurrente donnant lieu à une succession de **travaux**, appelés aussi instances de la tâche. Les travaux d'une application temps réel doivent respecter des contraintes temporelles telles que terminer leur exécution avant une certaine date, respecter des échéances de bout-en-bout, etc. À titre d'exemple, pour le modèle de tâches fondateur de Liu et Layland (1973), les tâches sont caractérisées par une durée d'exécution, une période d'activation et une contrainte d'échéance.

**Ordonnancer un système de tâches temps réel** consiste à définir une allocation spatiale et temporelle des travaux sur les processeurs de sorte à ce que les contraintes temporelles soient satisfaites. L'algorithme qui réalise ce travail est appelé **algorithme** (ou **politique**) **d'ordonnancement**. Nous parlerons dans la suite de **système ordonnancé** pour désigner un système de tâches associé à une politique d'ordonnancement.

Un système temps réel est qualifié de **dur** si les conséquences du non-respect de ses contraintes temporelles sont catastrophiques pour l'application, ou de **souple** si ce non-respect est acceptable dans certaines limites.

Deux grandes approches existent pour réaliser un tel ordonnancement : l'approche **hors-ligne** consiste à construire une allocation des travaux avant le démarrage du système alors que l'approche **en-ligne** décide de l'ordonnancement pendant l'exécution du système. Ces derniers algorithmes reposent en général sur la notion de **priorité**, c'est-à-dire un ensemble d'attributs associés aux travaux servant de critère pour déterminer lequel est à exécuter à un instant donné.

Un système $S$ est dit **fiablement ordonnancé** par un algorithme d'ordonnancement $A$ si et seulement si l'ordonnancement produit par $A$ respecte les contraintes temporelles de $S$. Nous dirons alors qu'un système $S$ est **ordonnançable** s'il existe un algorithme qui l'ordonnance fiablement. Des conditions suffisantes et nécessaires permettent de déterminer, dans certains cas, si un système est fiablement ordonnancé (condition suffisante satisfaite) ou s'il ne l'est pas (condition nécessaire non satisfaite) pour un algorithme donné.

### 2.1.1 L'ordonnancement temps réel du point de vue de la recherche

Les activités de recherche liées à l'ordonnancement temps réel ont pour objectif d'étudier les fondements théoriques de l'ordonnancement et de proposer des solutions pour produire des systèmes fiablement ordonnancés. Cela se décline à travers diverses activités scientifiques que l'on peut structurer selon les cinq axes proposés ci-après et présentés sur la figure 2.1.

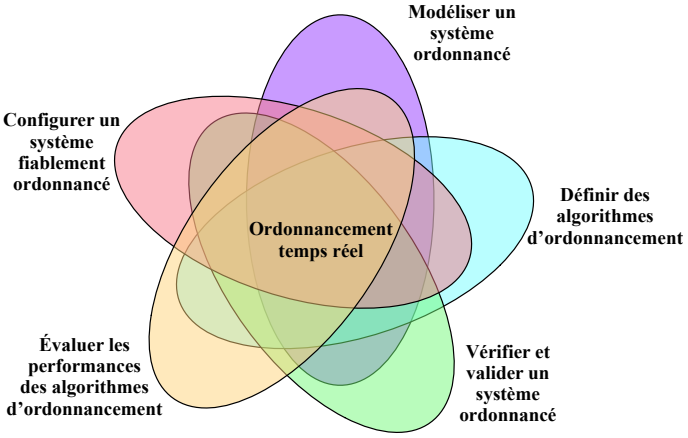



FIGURE 2.1 – Déclinaison des activités de recherche liées aux études sur l'ordonnancement temps réel (sous forme d'une généralisation du diagramme de Venn)

**Modéliser un système ordonnancé.** Bien que rarement identifiée en tant qu'activité indépendante, la modélisation d'un système ordonnancé est une problématique en soi. Deux volets peuvent être distingués : (i) la modélisation des tâches et de leur comportement et (ii) l'expression formelle des algorithmes d'ordonnancement.

Comme évoqué dans l'introduction de ce chapitre plusieurs modèles de comportement des tâches ont été proposés et formalisés. Nous pouvons citer à titre d'exemple le modèle de tâches sporadiques (Leung et Whitehead, 1982) ; le modèle *multiframe* (Mok et Chen, 1996) et sa généralisation (Baruah *et al.*, 1999) avec des durées d'exécution variables ; les transactions (Tindell, 1994) pour représenter les dépendances temporelles entre les tâches ; les branchements conditionnels dans les tâches récurrentes (Baruah, 2003) ; le modèle *digraph* (Stigge *et al.*, 2011) qui modélise les travaux par un graphe acyclique ; etc.

Cette très grande variété de modèles introduit la problématique de leur expression dans des langages adaptés afin d'être manipulés. Les travaux sur MAST (Harbour *et al.*, 2001), sur la définition du comportement des éléments de AADL (Franca *et al.*, 2007) ou sur le GQAM du standard MARTE (Gérard *et al.*, 2007) proposent ainsi des langages pour décrire des modèles de tâches dédiés à l'étude de l'ordonnancement.

Remarquons aussi les travaux de Migge (1999) pour modéliser formellement les algorithmes d'ordonnancement. Les politiques basées sur la priorité sont ainsi décrites sous la forme d'une combinaison linéaire de caractéristiques associées aux tâches (période, échéance, capacité, etc.) et par les opérateurs de tri min et max. Ce travail est cependant isolé et la majorité des politiques d'ordonnancement sont simplement définies au travers de leur algorithme.

Les langages haut-niveau de modélisaton de systèmes temps réel tels que AADL, UML/-MARTE, EAST-ADL, etc. introduisent aussi des éléments pour modéliser les composants propres à l'ordonnancement (threads, synchronisations, etc.). Par contre, les politiques d'ordonnancement y sont décrites de manière informelle, souvent par une simple référence à leur nom et par les paramètres qui leur sont propres (priorités, budget de temps, etc.). Les travaux exploitant ces modèles, comme ceux consacrés à la chaînes AADL avec Cheddar (Singhoff *et al.*, 2004), Ocarina (Lasnier *et al.*, 2009) ou Fiacre (Berthomieu *et al.*, 2009), consistent alors à projeter les éléments de haut-niveau vers des modèles de tâches pouvant être manipulés pour analyser l'ordonnancement temps réel.

**Définir des algorithmes d'ordonnancement.** Cette activité consiste à proposer des algorithmes pour construire des systèmes ordonnancés. Un grand nombre de solutions ont été proposées depuis les années 1970. Leung *et al.* (2004) ou Buttazzo (2011) en exposent de larges catalogues. Ces stratégies dépendent souvent du système (modèle de tâches, support d'exécution, contraintes temporelles, ressource, etc.) et offrent donc des solutions *ad hoc*, dans le sens où elles sont adaptées à un modèle de tâches spécifiques.

Ces dernières années, de nouvelles propositions de stratégies d'ordonnancement ont été publiées afin de répondre à de nouveaux challenges. Un effort important a notamment été consacré aux politiques pour les architectures multicœurs et multiprocesseurs (Davis et Burns, 2011; Chéramy *et al.*, 2015a) ainsi que les extensions pour les architectures *many-core*[1] (Kinsy et Devadas, 2014). La prise en compte de critères dits de criticité mixte pour des systèmes où des applications critiques et non-critiques se partagent une ressource d'exécution tout en garantissant des propriétés de sûreté et de sécurité, est aussi largement exploitée (Barhorst *et al.*, 2009; Kritikakou *et al.*, 2014). Enfin, une autre tendance récente dans l'étude des stratégies d'ordonnancement est de prendre en considération les incertitudes temporelles dues aux aspects matériels tels que les pipelines ou les mémoires caches (Altmeyer *et al.*, 2012; Chéramy *et al.*, 2015b; Phavorin *et al.*, 2015).

**Vérifier et valider un système ordonnancé.** Cette activité consiste à produire des méthodes pour vérifier ou valider le respect des contraintes temporelles en fonction d'une stratégie d'ordonnancement, nous parlons alors d'analyse d'ordonnançabilité. La majorité des travaux suivent une démarche dite analytique, c'est-à-dire qu'elle s'appuie sur des modèles avec un haut niveau d'abstraction et à partir desquels une expression analytique est produite afin de conclure sur le respect des contraintes de temps. D'autres formalismes pour conduire des analyses d'ordonnançabilité ont été utilisées comme l'algèbre max-plus (Zeng et Di Natale, 2013), la programmation linéaire (Kwak *et al.*, 1998) ou le model checking (Péres, 2010), mais cela reste anecdotique en nombre de publications.

Remarquons que les analyses d'ordonnançabilité vont de pair avec les stratégies d'ordonnancement et suivent donc les mêmes tendances : multicœur, criticité mixte, etc. ce qui conduit aussi à une littérature foisonnante sur le sujet.

**Évaluer les performances des algorithmes d'ordonnancement.** Cet aspect des recherches en ordonnancement temps réel aborde la comparaison des algorithmes d'ordonnancement en terme de performances. Un premier type de comparaison est réalisé

1. Nous utiliserons le terme *many-core* pour désigner une architecture matérielle avec un grand nombre de processeurs contrairement au multicœur qui en compte rarement plus d'une dizaine. Les architectures mémoire et de communication de ces systèmes sont aussi spécifiques afin de garantir de bonnes performances de calcul.

d'un point de vue théorique à partir des modèles de tâches et reposent alors uniquement sur des expressions analytiques. Les critères évalués sont nombreux et variés : optimalité des stratégies d'ordonnancement (Baruah, 2004), surcoûts système engendrés par les préemptions et migrations (Altmeyer *et al.*, 2012), viabilité [2] (Burns et Baruah, 2008), robustesse [3] (George, 2008), etc. Remarquons que ces méthodes proposent souvent des évaluations approchées, c'est-à-dire des évaluations de bornes supérieures et donc de conditions suffisantes d'ordonnançabilité qui peuvent conduite au rejet de systèmes fiablement ordonnancés.

L'étude des performances des stratégies d'ordonnancement se mène aussi sur de véritables architectures d'exécution. En effet, outre la complexité algorithmique de certaines méthodes, les choix d'implémentation d'un ordonnancement et la plate-forme matérielle d'exécution peuvent entraîner une importante dégradation des performances théoriques. De plus, les nouveaux supports d'exécution (many-core, architecture mixte FPGA et microprocesseur, GPU, etc.) et les architectures pour les systèmes d'exploitation (virtualisation, hiérarchisation, etc.) imposent une ré-évaluation de ces politiques d'ordonnancement. Dans ce cadre, les travaux expérimentaux, plutôt minoritaires en nombre de publications comparativement aux approches théoriques, se font par des mesures sur cible réelle. Les systèmes d'exploitation LITMUS (Calandrino *et al.*, 2006) ou Linux RT (Lelli *et al.*, 2012, 2015) ont par exemple servi de support à de telles évaluations.

Une approche intermédiaire, entre l'évaluation théorique et l'évaluation sur cible réelle, consiste à utiliser des simulateurs dédiés à l'ordonnancement (Singhoff *et al.*, 2004; Chéramy *et al.*, 2014a). Ces simulateurs permettent d'étudier facilement l'influence des paramètres d'un système (nombre de tâches, facteur d'utilisation du système, etc.) sur les performances des politiques d'ordonnancement, mais ne permettent pas d'analyser des effets précis dus à l'architecture matérielle, tels que les effets du cache dont le comportement est complexe à simuler.

**Configurer un système fiablement ordonnancé.** Cette dernière activité consiste à définir des méthodes pour configurer les paramètres d'ordonnancement d'un système (allocation des traitements sur les processeurs, niveau de priorité, réservation de la bande passante sur le réseau, etc.) de manière automatiques, ou non, en garantissant que le système soit fiablement ordonnancé.

Les problèmes spécifiques à l'ordonnancement, comme l'affectation des priorités pour les politiques à priorités fixes (Audsley, 2001) ou les échéances dans le cas d'EDF (Rivas *et al.*, 2015), sont souvent traités de manière ad-hoc avec des algorithmes dédiés.

Cependant, d'une manière plus générique, les méthodes pour la configuration se basent souvent sur une formalisation du problème d'ordonnancement spatial et temporel sous la forme d'un problème d'optimisation (par exemple la consommation d'énergie) sous contraintes (par exemple le respect des contraintes de temps). Une fois formalisés, ces problèmes peuvent se résoudre par des méthodes issues de l'aide à la décision telles que des méta-heuristiques (Kang et He, 2013), la programmation linéaire (Al Sheikh *et al.*, 2012), la programmation par contraintes (Ekelin, 2004), etc.

2. Une politique d'ordonnancement ou un test d'ordonnancement est viable (*sustainable*) si tous systèmes fiablement ordonnancés par cette politique le reste quand le système se comporte « mieux », par exemple si les durées d'exécution des tâches sont inférieures à leur pire cas d'exécution.

3. Un système est robuste s'il reste fiablement ordonnançé même si ses paramètres sont plus grands que ceux initialement prévus dans ses spécifications.

### 2.1.2 L'ordonnancement temps réel du point de vue de l'ingénierie

Les activités autour de l'ordonnancement ne sauraient être exhaustives sans considérer le point de vue de l'ingénierie dans la conception de systèmes temps réel. En effet, au cours du cycle de développement d'un système temps réel, les contraintes temporelles apparaissent à tous les niveaux. Ainsi, en prenant l'exemple classique d'un cycle en V (voir figure 2.2), ces contraintes sont exprimées dès le niveau des spécifications pour ensuite être déclinées et vérifiées a priori dans les étapes de conception et lors du codage. La branche montante d'intégration et de recette consiste principalement à tester et valider que ces contraintes sont bien respectées.

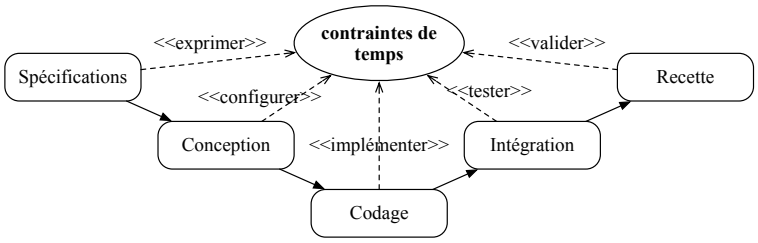



FIGURE 2.2 – Contraintes de temps et cycle de développement

La prise en considération de ces contraintes passe par les choix de conception notamment pour ordonnancer le système et a donc un impact important sur l'ensemble du cycle de développement. Pour réaliser ces choix, le concepteur va mener plusieurs activités consistant à (voir la figure 2.3) :

1. choisir une stratégie d'ordonnancement : le choix de la politique d'ordonnancement (approche en-ligne ou hors-ligne, multiprocesseur partitionné ou global, etc.) est rarement ouvert car il est généralement imposé par le système d'exploitation sur lequel est déployée l'application ou par le processus de conception mis en œuvre.
2. identifier les composants fonctionnels : au cours du développement, les fonctionnalités vont être décrites et raffinées par le concepteur conduisant à l'émergence des éléments de traitement concurrents.
3. estimer les paramètres temporels des composants fonctionnels : cette activité consiste à associer aux composants fonctionnels des paramètres temporels tels que la durée d'exécution d'un traitement, les temps de transmission des messages, etc. mais aussi les contraintes temporelles telles que les échéances. La caractérisation des durées peut se faire à l'aide d'outils dédiés ou par des mesures, mais leurs valeurs restent en général des estimations avec des marges d'incertitude.
4. associer les traitements aux composants d'ordonnancement : le concepteur doit associer les composants fonctionnels aux éléments d'ordonnancement du système (tâches, ressources, etc.). Le plus classique est de définir des tâches pour contrôler l'exécution des traitements, mais cela peut aussi passer par des interruptions ou par une implémentation matérielle (par exemple à l'aide de systèmes sur puce comme un FPGA).
5. configurer les paramètres d'ordonnancement : cela consiste à fixer les éléments propres à l'ordonnancement, ce peut être par exemple pour les tâches leur priorité, leur date de démarrage, leur allocation sur un processeur, etc.
6. vérifier et valider les contraintes temporelles : cette activité est menée à chaque étape du cycle de développement. Elle consiste à vérifier que les contraintes de temps sont bien respectées. Une validation le plus tôt possible est souhaitable afin d'éviter au maximum les retours en arrière au cours du cycle de développement.

Ces activités ne sont pas nécessairement conduites de manière séquentielle ou limitées à une étape du cycle de développement. Du point de vue du concepteur, l'ordonnancement apparait donc comme une activité transverse allant de l'étape de conception à celle d'intégration.

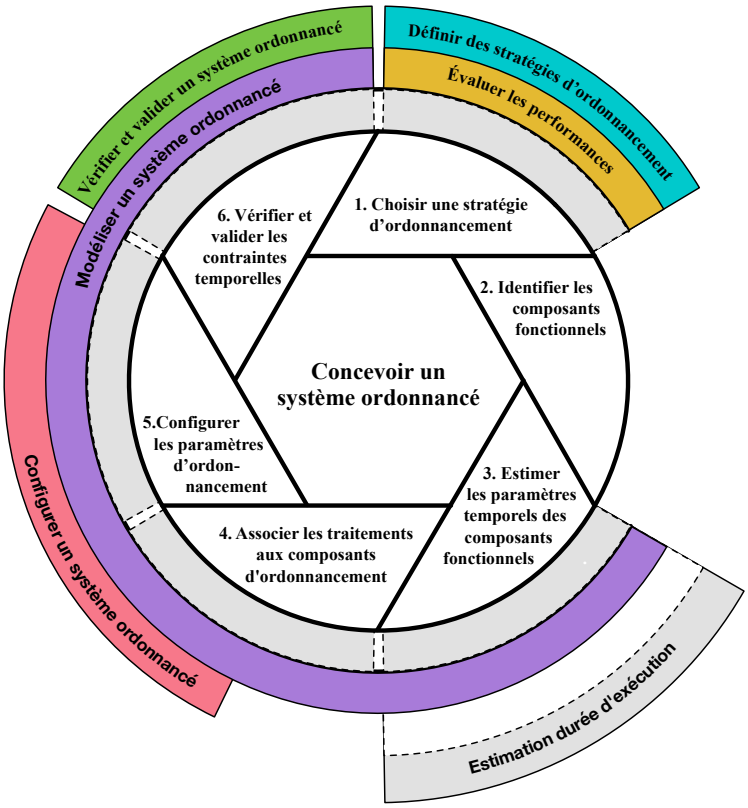



FIGURE 2.3 – Conception d'un système temps réel au travers des activités d'ingénierie liées à l'ordonnancement

Ces activités d'ingénierie peuvent être mises en correspondance avec la production scientifique dans le domaine de l'ordonnancement. La figure 2.3 donne un aperçu de ces liens avec en son centre les activités d'ingénierie décrites ci-dessus et en périphérie les activités de recherche qui peuvent s'y rattacher. Ainsi, le choix d'une stratégie d'ordonnancement (activité 1) pourra se faire en puisant dans la littérature existante et être motivé par les évaluations de leurs performances, alors que les activités 3, 4, 5 et 6 nécessitent la manipulation des modèles dédiés aux systèmes ordonnancés.

Remarquons que l'étape 4 qui associe les composants fonctionnels aux compostants d'ordonnancement est peu traitée dans la littérature scientifique, excepté dans le cadre du standard automobile AUTOSAR avec, par exemple, les travaux de Scheickl et Rudorfer (2008), Bertout *et al.* (2013) et Wozniak *et al.* (2014). Cela s'explique par la cohérence entre le processus de développement, les modèles et les architectures définis dans ce standard.

La figure 2.3 fait aussi apparaître le domaine de recherche consacré à l'estimation des durées d'exécution (Wilhelm *et al.*, 2008). Celui-ci est généralement considéré comme distinct de l'ordonnancement temps réel alors que du point de vue de l'ingénierie, ces deux domaines sont fortement liés. Quelques travaux abordent ces relations tels que ceux de Nguyen *et al.* (2015), mais sont, à ma connaissance, pour l'instant plutôt confidentiels.

## 2.2 Thématiques couvertes par mes travaux de recherche

Après avoir présenté rapidement l'ordonnancement temps réel et abordé ses différentes facettes tant du point de vue de la recherche que de l'ingénierie, les sections suivantes proposent un positionnement de mes travaux dans ce domaine.

La présentation est structurée suivant différentes thématiques de recherche et met en avant les problématiques et les méthodes proposées pour y répondre. Les thématiques sont transverses aux activités scientifique et d'ingénierie présentées dans la partie précédente. Le tableau 2.1 montre le recouvrement de ces différentes thématiques avec les projets auxquels j'ai participé et qui ont été soutenus par mes tutelles ou par l'ANR (voir partie 1.3), ainsi que les liens avec la seconde partie de ce manuscrit consacrée à l'exposé détaillé de mes travaux.

| Thématiques | AtlanSTIC | ANR SATRIMAP (Thèse A. Al Sheikh) LAAS OSEC | ANR RESPECTED (Thèse M. Chéramy) | Chapitre 3 | Chapitre 4 | Chapitre 5 | Chapitre 6 | Chapitre 7 | Chapitre 8 |
|---|---|---|---|---|---|---|---|---|---|
| | **Projets** | | | **Chapitres techniques** | | | | | |
| Application de la théorie aux standards industriels (§2.2.1 ) | | ✓ | ✓ | | ✓ | ✓ | ✓ | | |
| Études des analyses d'ordonnançabilité (§2.2.2) | | | ✓ | | | ✓ | ✓ | ✓ | |
| Aide à la décision pour la configuration (§2.2.3) | ✓ | ✓ | ✓ | ✓ | ✓ | | | | |
| Modèles de temps d'exécution incertains (§2.2.4) | | | ✓ | ✓ | | | | | ✓ |

TABLE 2.1 – Thématiques abordées dans les projets de recherche déjà réalisés et relations avec les chapitres scientifiques du manuscrit.

### 2.2.1 Application de la théorie aux standards industriels (et vice-versa)

**Problématique.** La conception de systèmes temps réel repose souvent et de plus en plus sur l'utilisation de standards industriels. Les principaux standards actuels sont RT-POSIX (IEEE std 1003.13), AUTOSAR OS pour l'automobile et ARINC 653 pour l'avionique. Les interfaces et les services des systèmes d'exploitation définis dans ces standards diffèrent en fonction des domaines industriels, mais restent bâtis autour des mêmes concepts (tâche, politique d'ordonnancement, priorité fixe, protocole de partage de ressource, etc.).

Les standards ont un rôle central pour le développement des applications car ils assurent, entre autre, leur portabilité et leur réutilisabilité. Ils permettent aussi de disposer de plusieurs implémentations de systèmes d'exploitation et ainsi d'accroître l'offre (voir Buttazzo, 2011, chap. 2). Les standards influent ainsi sur le processus de conception, voire le processus fait partie du standard comme pour AUTOSAR, et impactent donc toutes les étapes du développement et les outils à mettre en œuvre dans un cycle développement. Les prendre en considération dans les activités de recherche en ordonnancement temps réel semble donc une nécessité si l'on veut produire des solutions exploitables par le monde industriel.

Considérer les standards dans les travaux de recherche consiste à adapter les modèles et les approches en prenant en compte les spécificités des standards. Pour ce faire, l'ensemble des activités identifiées doivent être menées de manière spécialisée et peuvent ainsi s'énoncer (voir figure 2.4) :

— modéliser un système embarqué conforme à un standard : les modèles génériques de systèmes ordonnancés doivent être spécialisés pour un standard particulier, ce qui est par exemple le cas de l'annexe ARINC 653 pour AADL (`http://standards.sae.org/as5506/2/`), soit de nouveaux modèles sont introduits pour saisir les spécificités du standard, par exemple la publication (Hladik *et al.*, 2007a) propose un modèle pour les systèmes AUTOSAR.

— définir des stratégies d'ordonnancement pour un besoin industriel : le travail de définition de la stratégie d'ordonnancement consiste à fixer pour un standard une ou plusieurs stratégies. Cette définition est généralement nourrie par les apports des travaux de recherche et leur confrontation aux besoins industriels.

— vérifier et valider un système ordonnancé en intégrant un standard : l'activité de vérification et de validation pour des systèmes standardisés suit deux tendances. La première consiste à étudier l'adéquation entre les résultats des travaux de recherche et les mécanismes des standards, voire à étendre les méthodes existantes si des manques sont constatés. La seconde, au contraire, consiste à restreindre l'usage des éléments du standard pour en garantir leur vérification par de méthodes déjà établies, ce qui est le cas par exemple du profil Ravenscar (Burns *et al.*, 2004). En pratique ces deux tendances se rejoignent car il est quasiment impossible de proposer des méthodes intégrant l'ensemble d'un standard, retrouvant de facto la seconde approche.

— évaluer les performances des mécanismes du standard : la majorité des mécanismes utilisés dans les standards ont déjà été évaluée dans la littérature scientifique, cependant, certaines spécificités imposent parfois de mener des évaluations dédiées. Cela a par exemple été le cas pour les mécanismes de surveillance temporelle ou pour le protocole de partage de ressource en multiprocesseur introduits dans AUTOSAR.

— configurer un système fiablement ordonnancé sous contrainte d'un standard : l'activité de configuration en suivant un standard se traduit généralement sous la forme de contraintes spécifiques issues du standard (par ex. la notion de partition strictement périodique avec ARINC 653). Cela se concrétise par des efforts importants pour développer des outils de configuration pour les systèmes qui s'appuient sur ces standards tels que l'atelier OCARINA (Lasnier *et al.*, 2009) avec AADL et RT-POSIX ou la suite logicielle PharOS (Lemerre *et al.*, 2011) pour AUTOSAR.

Remarquons que les standards doivent aussi intégrer les avancées issues de la recherche pour faire évoluer les pratiques d'ingénierie. Cet échange réciproque est souvent fécond et se retrouve au niveau des principaux consortiums de standardisation. L'introduction récente de la politique d'ordonnancement EDF avec une variante du *Constant Bandwidth Server* (CBS) dans la branche officielle de Linux, les évolutions d'AUTOSAR pour les architectures multiprocesseurs ou encore l'introduction des méthodes formelles dans les DO 178C sont quelques exemples de cet appropriation d'avancées académiques.

**Travaux réalisés.** Plusieurs de mes travaux exposés dans ce manuscrit sont en relation avec l'étude de standards industriels (la figure 2.4 les positionne par rapport aux activités plus générales de recherche). Le premier est introduit dans le chapitre 4 et a été réalisé dans le contexte avionique en se basant sur ARINC 653 pour la partie applicative et ARINC 664 pour la partie réseau (avec l'implémentation AFDX). Ces standards ont été utilisés pour modéliser et contraindre le problème de configuration. Ainsi les méthodes développées pour l'allocation des partitions et pour le dimensionnement des communications respectent par

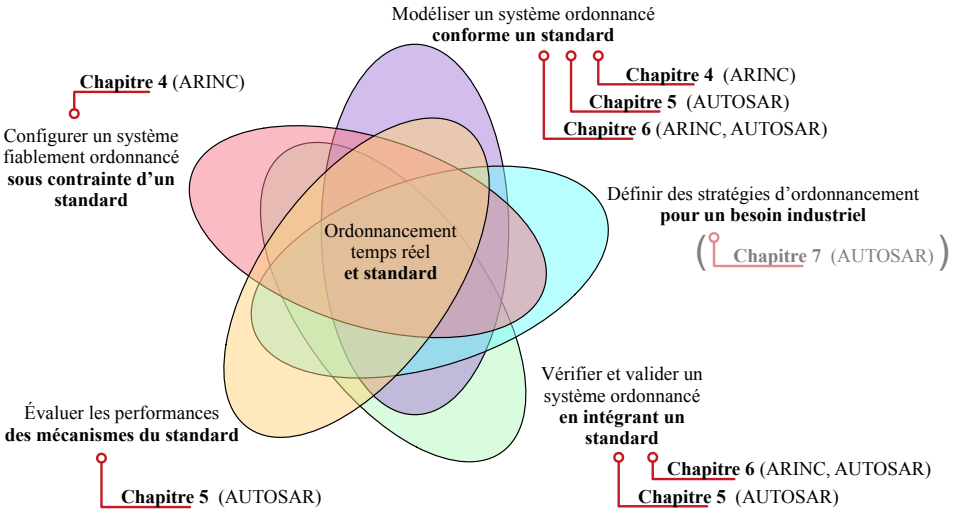



FIGURE 2.4 – Impact des standards sur les activités de recherche liées à l'ordonnancement

construction les spécifications des standards ARINC, ce qui rend générique ces approches pour les architectures avioniques.

Le chapitre 5 présente une méthode d'analyse d'ordonnançabilité pour AUTOSAR OS. L'étude a débuté par une appropriation du standard afin de modéliser formellement les différents composants qui ont une influence sur l'ordonnancement. Partant de cette formalisation, une méthode d'analyse d'ordonnançabilité a été développée. Lors de cette étude, nous avons aussi relevé certaines imprécisions dans les spécifications d'AUTOSAR OS, en particulier sur les mécanismes de surveillance des budgets de temps (la version suivante du standard corrige ces imprécisions).

Enfin, le chapitre 6 détaille une autre étude qui se base sur les standards industriels. Afin de montrer l'expressivité du langage Pola, langage de modélisation dédié à la vérification de l'ordonnançabilité, nous avons considéré des exemples tirés des domaines automobile et avionique. Pour cela, nous nous sommes appuyé sur les standards liés à ces systèmes et avons montré qu'il était possible de modéliser les mécanismes standardisés avec ce langage formel.

**Bilan.** Les différents travaux que j'ai menés m'ont permis d'acquérir une connaissance approfondie de l'ordonnancement pour les deux standards industriels que sont AUTOSAR et ARINC. Réaliser ces études m'a convaincu qu'il est indispensable de considérer les standards industriels si l'on veut exporter les résultats scientifiques de l'ordonnancement temps réel vers l'ingénierie. En cela, j'ai contribué au développement d'outils et de méthodes adaptés à cette démarche.

Ces travaux m'ont aussi convaincu de la nécessité de collaborer à la définition de ces standards — par une éventuelle participation aux comités ou, plus raisonnablement, par leur suivi et leur analyse critique —, soit pour faire remonter d'éventuelles améliorations, soit pour y intégrer des propositions issues de la recherche. Les travaux présentés dans le chapitre 7 sur l'étude des politiques d'ordonnancement multiprocesseur vont dans ce sens. En effet, ils ont été conduits dans le cadre du projet ANR RESPECTED dont l'un des buts était d'évaluer les solutions d'ordonnancement proposées en recherche au regard du domaine industriel de l'automobile et de leur éventuelle adaptation au standard AUTOSAR.

### 2.2.2 Études des analyses d'ordonnançabilité

**Problématique.** Comme exposé dans la partie précédente, être capable de vérifier qu'un système est fiablement ordonnancé est une problématique importante dans le domaine de l'ordonnancement temps réel. Les travaux fondateurs de Liu et Layland (1973) ont ouvert la porte à un domaine de recherche qui reste encore très actif quarante ans plus tard. Les études ainsi réalisées offrent un très large éventail de méthodes qui s'appuient sur des théories variées.

De manière générale, ces techniques de vérification consistent à modéliser le système avec un formalisme qui simplifie certains comportements — mais qui doit rester suffisamment riche pour être pertinent — puis à l'analyser. L'expressivité des modèles par rapport à l'implémentation réelle du système reste un problème. En effet, plus un formalisme décrit des comportements détaillés, donc proches de l'implémentation, moins les techniques d'analyse sont efficaces pour couvrir de manière exhaustive tous les comportements. Et à l'inverse, trouver un compromis entre représentativité de l'abstraction et pertinence de l'analyse est souvent délicat.

Parmi les approches employées pour analyser un modèle et statuer sur la validation du système qu'il représente, trois approches sont souvent employées (voir figure 2.5) : les approches analytiques, le model-checking et la simulation.

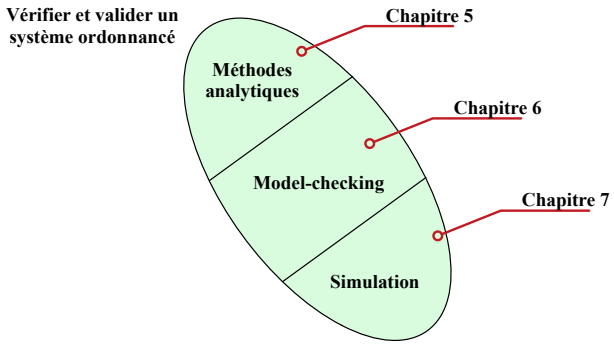



FIGURE 2.5 – Différentes approches pour l'activité de vérification et de validation de système ordonnancé

Les méthodes analytiques consistent à identifier ou à borner les pires scénarios d'exécution d'un modèle et à produire une expression analytique permettant d'évaluer l'ordonnançabilité du système. L'analyse peut être extrêmement difficile pour des systèmes ayant des comportements complexes. Ainsi, il est courant que le modèle d'un système soit transformé en un autre modèle plus simple, mais qui englobe les comportements du premier, dont l'analyse est faisable tout en s'assurant que les propriétés d'ordonnançabilité sont conservées. Dans ce cas, la démarche devient pessimiste, c'est-à-dire que le système peut être trouvé non ordonnançable à tort.

Le model-checking explore exhaustivement l'espace d'états d'un modèle. Beaucoup plus riches qu'une analyse analytique, ces méthodes permettent de vérifier de nombreuses propriétés autres que l'ordonnançabilité du système. Le formalisme utilisé est en général celui des systèmes de transitions étendus temporellement : automates, réseaux de Petri ou parfois algèbre des processus. Ces modèles offrent une description fine du comportement du système, mais l'espace d'états explose alors rapidement et limite la taille des modèles

analysables. De nombreux travaux réduisent cette complexité, sans toutefois offrir des solutions viables pour l'analyse de grands systèmes ordonnancés. En particulier, la prise en charge de la préemption introduit l'indécidabilité du modèle.

Enfin, la simulation consiste à modéliser le comportement du système à l'aide d'un formalisme adéquat, puis à exécuter le modèle à l'aide d'un moteur de simulation sur des scénarios définis par l'utilisateur. Les informations collectées peuvent être de simples traces de l'exécution du système ou des évaluations sur certains comportements du système. Les modèles simulés peuvent être particulièrement complexes et fins. Contrairement au model-checking, la taille du modèle n'est pas un facteur limitant de la simulation puisque l'espace des comportements est exploré linéairement. La principale faiblesse de la simulation est que l'analyse est optimiste, dans le sens où elle peut montrer qu'un système n'est pas ordonnançable (en exhibant un scénario incorrect), mais elle ne peut pas prouver qu'un système est fiablement ordonnancé (même si tous les scénarios étudiés sont corrects). L'analyse des traces permet cependant de détecter des erreurs temporelles et des erreurs de dimensionnement du système.

Ces méthodes de vérification ont leurs avantages et défauts et doivent être choisies en fonction de la complexité induite par le modèle et par les propriétés à vérifier. Vouloir les comparer en terme de performance n'est pas pertinent. Les méthodes analytiques sont évidemment moins complexes (au sens algorithmique) que le model-checking, mais incapables de prouver des propriétés de logique temporelle ; la comparaison entre ces méthodes en devient sans fondement.

**Travaux réalisés.** Le manuscrit présente trois travaux menés sur l'analyse d'ordonnançabilité, chacun suivant l'une des approches évoquées ci-dessus. Le chapitre 5 décrit une méthode analytique basée sur le calcul des temps de réponse pire cas pour des systèmes AUTOSAR. La difficulté principale a été d'intégrer de manière cohérente les différents résultats déjà connus en ordonnancement dans une méthode générique capable de couvrir l'ensemble des spécifications AUTOSAR.

L'approche par model-checking est abordée dans le chapitre 6 à travers le langage Pola, mis au point par Florent Péres (2010). Cette étude a pour but de valider l'expressivité du langage pour décrire des politiques d'ordonnancement et des systèmes de tâches. La validation de l'ordonnancement du système s'appuie sur les techniques classiques de model-checking appliquées sur de réseaux de Petri temporels.

Enfin, le chapitre 7 s'attache au développement d'une plate-forme de simulation d'ordonnancement pour des systèmes de tâches afin d'étudier les performances des algorithmes d'ordonnancement multiprocesseur. Ce travail a été mené dans le cadre de la thèse de Maxime Chéramy (2014).

**Bilan.** Les travaux que j'ai réalisés m'ont permis d'approfondir différentes approches pour vérifier le comportement temporel de systèmes ordonnancés et ainsi assoir mon expertise dans le domaine des politiques d'ordonnancement et des techniques de vérification. Comme dit précédemment, le point essentiel n'est pas de les comparer en terme de performance de calcul, mais plutôt d'en apprécier leurs limites et leurs possibilités en fonction des besoins de vérification.

La maîtrise de ces différentes approches m'a permis de les exploiter conjointement à d'autres problématiques, par exemple pour la configuration de systèmes en réexprimant les approches analytiques sous forme de contraintes (voir partie 2.2.3), et d'envisager de futurs travaux en établissant des passerelles entre elles (voir partie 2.4.2).

### 2.2.3 Aide à la décision au service de la configuration d'applications temps réel

**Problématique.** Comme exposé ci-avant, une des activités de recherche en ordonnancement consiste à produire des méthodes et des outils pour la configuration (allocation des tâches, priorités, etc.) automatique de systèmes ordonnancés en respectant des contraintes telles que le taux d'occupation des processeurs, les échéances des tâches, les latences sur les communications, etc.

Trouver une configuration qui respecte l'ensemble des contraintes est un problème généralement NP-complet et nécessite donc des heuristiques performantes pour y apporter une réponse. Nous pouvons identifier trois étapes qui composent ce travail (voir figure 2.6) :

1. Le problème de configuration est formalisé sous la forme d'un problème de satisfaction de contraintes ou d'optimisation sous contraintes si un critère d'optimisation est identifié. Modéliser les stratégies d'ordonnancement peut s'avérer difficile et se traduit en général par la réécriture de conditions d'ordonnançabilité.
2. L'étape suivante consiste à automatiser le calcul des paramètres de la configuration comme l'allocation des traitements, la réservation des ressources sur un réseau, l'affectation des priorités aux tâches, etc. Les travaux existants dans ce domaine sont très variés et s'appuient principalement sur des techniques d'aide à la décision.
3. Une dernière étape, rarement abordée, s'intéresse aux moyens pour guider un concepteur en cas d'infaisabilité de la configuration, c'est-à-dire lorsqu'il n'existe pas de solution au problème contraint. Le but de cette étape est alors d'identifier les causes qui rendent impossible la configuration et de les exposer rétroactivement et de manière intelligible au concepteur afin qu'il puisse revoir sa conception.

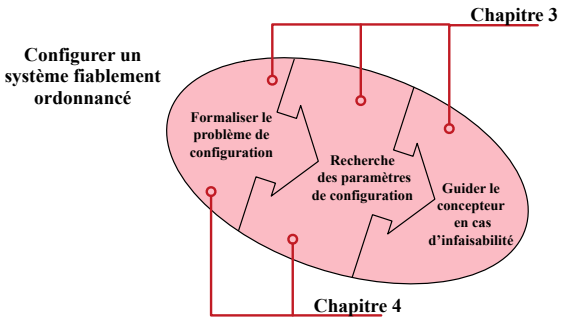



FIGURE 2.6 – Détail de l'activité de configuration de l'ordonnancement d'un système

**Travaux réalisés.** Le travail présenté dans le chapitre 3 couvre toutes les étapes évoquées ci-avant. Il traite des architectures distribuées avec un ordonnancement partitionné à priorités fixes. Les contraintes portent sur les échéances des tâches, sur l'occupation mémoire des ressources et sur le placement des traitements. Ce type de système est représentatif de ceux utilisés dans le domaine automobile. Pour résoudre ce problème, la programmation par contraintes est utilisée et deux approches sont proposées. L'une est basée sur la décomposition de Benders en isolant le problème lié à l'ordonnancement des autres contraintes. La seconde approche consiste à implémenter une contrainte dédiée à l'ordonnancement et à l'utiliser conjointement aux autres contraintes.

À la suite de ce travail, une méthode a été développée pour guider le concepteur si aucune configuration ne peut satisfaire le problème. Pour cela, les explications, apprises à l'aide de la décomposition de Benders, servent à évaluer un critère pour incriminer les tâches

ou les messages qui contraignent le plus le problème. Le concepteur peut ainsi revoir sa conception en disposant d'informations sur les causes de l'infaisabilité du problème.

Toujours dans le domaine de la configuration d'un système ordonnancée, le travail de thèse d'Ahmad Al Sheikh (2011), détaillé dans le chapitre 4, s'est focalisé sur :

— l'allocation et l'ordonnancement automatique de partitions strictement périodiques d'une application avionique. Pour cela, une modélisation en programmation linéaire mixte a été menée et un algorithme *ad hoc* inspiré de la théorie des jeux a été implémenté ; et

— le dimensionnement et le routage des messages dans un réseau avionique AFDX afin d'optimiser la consommation de la bande passante. L'implémentation d'un algorithme de *branch-and-bound* et l'utilisation de la programmation linéaire en nombres entiers a permis de résoudre ce problème.

Une démarche incluant ces deux approches a aussi été proposée. Ce qui a conduit à la mise en place d'un processus opérationnel pour configurer les partitions et le réseau, mais qui est sous-optimale puisque les problèmes sont résolus séquentiellement.

Le travail en collaboration avec l'ÉTS sur une approche pour optimiser la configuration des flots sur un réseau TTEthernet (Beji *et al.*, 2016) n'est pas présenté dans ce document, mais s'inscrit aussi dans le cadre des études sur les méthodes pour aider à la configuration d'applications temps réel.

**Bilan.** Ces deux études m'ont permis de développer des outils et des méthodes pour aider et automatiser la configuration de systèmes temps réel. Cela me semble d'autant plus nécessaire que les systèmes actuels ont une complexité qu'un expert ne peut plus appréhender. Cependant, mes différentes participations à des projet de recherche en relation avec l'industrie, m'amènent à penser que l'acceptation de tels outils dans un processus de conception de systèmes temps réel est encore à renforcer.

Ces travaux m'ont aussi convaincu de la nécessité d'établir des collaborations entre les domaines scientifiques de l'ordonnancement temps réel avec celui de la recherche opérationnelle (RO) afin d'aboutir au développement de méthodes performantes. Les études auxquelles j'ai participées en collaboration avec des équipes de RO ont été à chaque fois productives et ont permis d'alimenter les travaux de chaque communauté scientifique. Bien qu'au cours de ces trois dernières années, cette collaboration ait été moins présente dans mes activités de recherche, maintenir cette double compétence est un point qui me semble essentiel pour la suite de mes travaux.

### 2.2.4 Analyse de l'ordonnancement avec des modèles de temps d'exécution incertain

**Problématique.** La durée d'exécution d'un traitement dans un systèmes temps réel peut fortement varier pendant son exécution. Les causes sont multiples et proviennent aussi bien du traitement lui-même (par exemple dépendance aux données) que du support matériel (par exemple utilisation de mémoire cache). Or, pour garantir a priori l'ordonnançabilité du système, les méthodes de conception et de validation des systèmes ordonnancés s'appuient principalement sur des analyses pessimistes utilisant une sur-estimation des pires temps d'exécution des fonctions de traitement. Ces analyses conduisent alors à un sur-dimensionnement des ressources et imposent souvent des contraintes inutiles lors des phases de conception. Si cette démarche est appropriée dans certains domaines, comme

l'aéronautique où le besoin de certification est très fort, elle ne semble pas adaptée dans des contextes applicatifs plus souples.

Une difficulté pour prendre en considération la variation des durées d'exécution dans les analyses est due aux incertitudes induites par le système. En effet, la complexité du système rend quasiment impossible de connaître son état exact à un instant donné et donc d'en prédire son comportement. Il devient alors nécessaire de manipuler des modèles stochastiques et des méthodes adaptées pour analyser le système. Du point de vue des activités scientifiques liées à l'ordonnancement temps réel, la prise en compte des incertitudes impacte l'ensemble des activités classiques (voir figure 2.7).

Ainsi, de nombreux travaux sont consacrés à la prise en compte de ces incertitudes en les intégrant dans le modèle de comportement du système et dans les méthodes d'analyse, voire dans les stratégies d'ordonnancement (Kritikakou *et al.*, 2014). Dans le domaine de l'analyse d'ordonnançabilité, nous pouvons citer à titre d'exemple, les extensions à la théorie des files d'attente proposées par Lehoczky (1996, 1997a,b) dans les années 90 ou plus récemment les travaux de Kruk *et al.* (2004). D'autres approches ont été proposées en se basant sur des calculs stochastiques des temps de réponse comme ceux de Diaz *et al.* (2004) ou Cazorla *et al.* (2013). De nombreux modèles stochastiques ont aussi été proposés pour estimer au mieux le comportement d'une application réelle. Citons en exemple l'outil d'analyse RapiTime de l'université de York (Bernat *et al.*, 2003) ou l'utilisation de la théorie des valeurs extrêmes pour estimer la pire durée d'exécution d'une tâche (Cucu-Grosjean *et al.*, 2012).

L'ensemble des travaux traitant des incertitudes dans l'ordonnancement temps réel sont relativement récents et ont connu récemment un nouveau regain d'intérêt. À ma connaissance, très peu ont été menés pour intégrer les incertitudes lors de la configuration d'un système (Radojković *et al.*, 2012) et aucun n'a été appliqué pour développer des systèmes industrialisés.

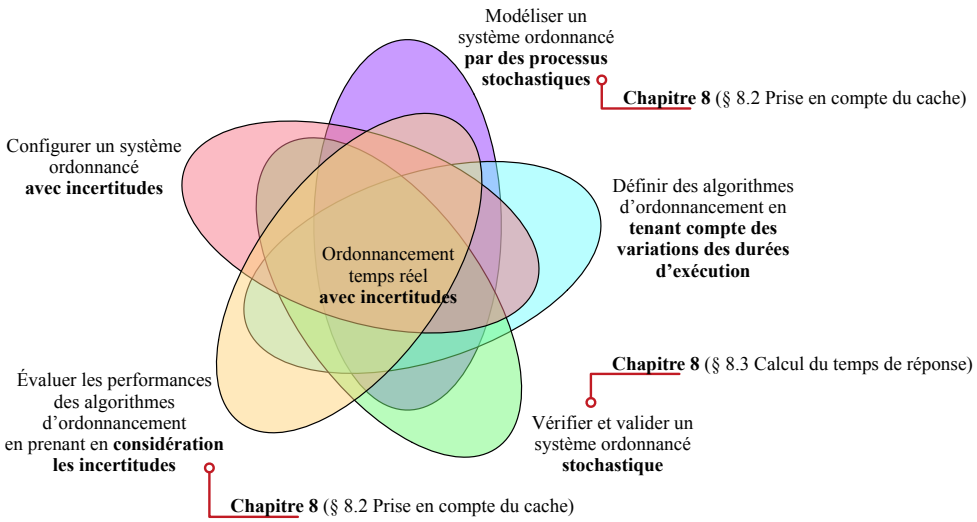



FIGURE 2.7 – Activités de recherche en ordonnancement avec prise en considération des incertitudes sur les temps d'exécution

**Travaux réalisés.** Les méthodes d'analyse stochastique d'ordonnançabilité sont complexes bien que ne prenant en considération que des modèles simples. En réduire la complexité est donc nécessaire pour les rendre utilisables. La première partie du chapitre 8

présente une étude menée sur ce sujet lors du stage de Khaled Reefat en 2009 (voir partie 1.2). Elle consiste à réduire par échantillonnage la complexité calculatoire du calcul du temps de réponse pour des tâches avec des durées d'exécution probabilistes. Ce travail a donné lieu à la publication (Refaat et Hladik, 2010) et a été poursuivi lors du stage de Vikas Shukla en 2010 (voir partie 1.2) autour de l'outil Stochan (López *et al.*, 2008) développé à l'université d'Oviedo.

En complément de cette étude, le stage de Larissa Calsavara en 2011 (voir partie 1.2) a permis de traiter des réseaux de Petri stochastiques et des méthodes de vérification associées, en particulier celles proposées par la boîte à outil Oris (Bucci *et al.*, 2004). Ce travail a consisté à évaluer l'expressivité du modèle ainsi que les performances du model-checking et ses limites. Actuellement, l'outillage et la théorie dans ce domaine ne semblent pas suffisants pour répondre aux problématiques des systèmes temps réel avec paramètres probabilistes.

Pour terminer sur ces aspects, les travaux de thèse de Maxime Chéramy ont porté sur l'évaluation par simulation des politiques d'ordonnancement multiprocesseur en prenant en considération les effets des mémoires caches partagées. Ce travail a permis d'étendre les modèles classiques de tâches avec des modèles stochastiques représentatifs des perturbations induites par les accès mémoires sur les temps d'exécution des tâches. La seconde partie du chapitre 8 présente ces travaux de manière plus détaillée.

Remarquons qu'une tentative pour prendre en considération des incertitudes sur les durées d'exécution lors de la configuration d'un système (i.e. calcul d'offset pour minimiser la probabilité de dépassement d'échéance) a été faite dans le cadre du projet OSEC mais n'a malheureusement pas produit de résultats effectifs.

**Bilan.** Les travaux que j'ai menés autour de l'ordonnancement temps réel avec des temps d'exécution incertains sont principalement de nature exploratoire. Cependant, ces études m'ont convaincu que la prise en compte des incertitudes dans la conception des systèmes temps réel est une problématique scientifique qui conduira à revoir certaines pratiques établies pour la conception de systèmes temps réel.

Mes premières études dans ce domaine ont principalement porté sur les méthodes d'analyse pour des modèles stochastiques. Alors que celles réalisées au cours de la thèse de Maxime Chéramy m'ont permis de mieux comprendre les phénomènes qui sont les causes des incertitudes, ainsi que les modèles et les méthodes pour les quantifier. Ainsi, convaincu de l'intérêt de ce domaine de recherche et partant de ces différentes expériences, je compte approfondir ces investigations au cours de mes futurs travaux comme nous le verrons dans la partie 2.4.3.

## 2.3 Point de vue sur l'ordonnancement

Les sections précédentes ont montré un échantillon assez large de problématiques liées à l'ordonnancement temps réel. Cependant, ces études me conduisent à m'interroger sur la direction à suivre dans le domaine de l'ordonnancement temps réel et comment mes futurs travaux s'y rattachent.

Mes projets (décrits en détail dans la partie 2.4) sont principalement orientés vers l'intégration des méthodes de vérification dans une démarche de conception pour les systèmes

temps réel, et vers une meilleure prise en considération des éléments physiques du système. Nous verrons dans la suite de cette section, comment j'inscris mon expérience sur l'ordonnancement temps réel dans ces perspectives.

### 2.3.1 Intégration de l'ordonnancement dans le processus de développement des systèmes temps réel

Si l'on aborde les publications sur l'ordonnancement temps réel avec l'objectif de les intégrer dans une démarche de développement de systèmes, il me semble qu'elles sont souvent difficilement exploitables. En effet, certaines études se réduisent à une simple variation de paramètres dans un modèle abstrait et éloignés des besoins concrets. La catégorisation et la définition formelle de problèmes est bien évidemment une activité importante pour analyser et comprendre une problématique de recherche, mais ne saurait être suffisante pour réaliser un systèmes temps réel. En particulier, il me semble que la production scientifique qui ne considère que le sujet de l'ordonnancement a une tendance à la prolifération en oubliant le contexte industrel.

Ainsi, au travers des projets dans lesquels j'ai été impliqué, j'ai pu constater que les méthodes par tests et mesures sur cible réelle sont majoritairement employées dans l'industrie pour valider l'ordonnançabilité d'un système. De plus, la vérification et la configuration lors de l'étape de conception sont généralement réalisées par un expert avec des outils de type simulation. Les méthodes analytiques sont rarement utilisées car les abstractions des modèles ne sont souvent pas adaptées pour modéliser le comportement d'une application (l'exemple le plus commun étant les dépendances entre les tâches) et les hypothèses d'utilisation sont souvent mal comprises par les ingénieurs et difficiles à étendre. Il en est de même pour les approches par model-checking qui souvent explosent dès que l'on tente de les mettre en œuvre sur des systèmes ordonnancés complets.

À cela s'ajoute souvent un décalage entre les activités scientifiques en ordonnancement et leur répercussion vers l'industrie. Par exemple, à ma connaissance, tous les standards industriels pour les domaines critiques sont bâtis sur des politiques d'ordonnancement à priorités fixes. Au niveau des systèmes d'exploitation industriels, seul RTEMS [4], Linux [5] et ASTERIOS [6] ont introduit EDF dans leur noyau . Les systèmes d'exploitation qui intègrent des politiques d'ordonnancement différentes relèvent du cadre universitaire et sont, par leurs hypothèses sous jacentes ou leur manque de maturité technologique, inutilisables en situation industrielle. Les approches multiprocesseurs changent légèrement la donne, mais les standards et les implémentations commerciales dans ce domaine ne considèrent, à ma connaissance, que des politiques par partitionnement à priorités fixes (voir le chapitre 7 pour plus de détails sur les politiques d'ordonnancement multiprocesseur) ou des politiques globales à priorités fixes avec migration restreinte, par exemple VxWorks SMP de Wind River.

Nous pourrions penser que les standards et les implémentations industriels intégreront les résultats des productions scientifiques d'ici quelques années. Cependant, rien n'indique que ce sera le cas, ou bien simplement à la marge, car le besoin industriel ne semble pas être sur la définition de nouvelles politiques d'ordonnancement, mais plutôt sur des méthodes pour garantir le bon comportement du système en opération. Ce qui implique

4. https ://www.rtems.org
5. L'ordonnanceur implémenté dans Linux est basé sur EDF avec un serveur CBS (Lelli *et al.*, 2015).
6. ASTERIOS de la société Krono-Safe est issu des travaux du CEA sur OASIS/PharOS et propose une solution clef-en-main pour générer un exécutif respectant des propriétés temporelles, voir le site `http://www.krono-safe.com/` [consulté en janvier 2016].

généralement des solutions simples pour pouvoir en faire la preuve ou bien un outillage complet et prouvé pour en cacher la complexité. Or les résultats scientifiques pour les politiques d'ordonnancement « évoluées » n'offrent pas de solution intégrée capable de prendre en considération la complexité des systèmes réels.

La difficulté à exploiter les résultats issus de la recherche me semble provenir principalement du manque de travaux montrant leur intégration dans un processus complet de conception. Le profil Ravenscar défini par Burns *et al.* (2004) est une réponse intéressante à ce problème car il restreint la conception à des systèmes analysables ce qui permet de leur appliquer des méthodes analytiques de vérification. De même, le projet MoSaRT (Ouhammou *et al.*, 2014) qui vise à offrir un outil pour utiliser les analyses d'ordonnançabilité dans une approche dirigée par les modèles, est une initiative qui va dans le sens d'un effort pour porter les résultats de la recherche vers l'ingénierie. Enfin, la proposition récente de Elliott *et al.* (2015) pour introduire des garanties temps réel dans le standard de traitement d'images OpenVX est aussi une approche illustrant les possibilités d'intégration d'une politique d'ordonnancement (global EDF) et les méthodes de vérification afférantes dans une démarche de conception.

D'une manière plus générale, il me semble que le domaine de l'ordonnancement temps réel a atteint un degré de maturité important depuis les travaux initiaux des années 1970 et les études poursuivies actuellement apportent une meilleure compréhension des mécanismes et des causes des problèmes liés à l'ordonnancement pour de nouvelles architectures. Par contre, un effort reste à faire pour intégrer ces résultats dans des démarches de conception afin d'apporter des solutions concrètes aux concepteurs de systèmes temps réel. Au regard de ces réflexions, l'axe de recherche présenté dans la partie 2.4.2 aborde ces problèmes et propose une solution pour intégrer l'ordonnancement et sa vérification dans un processus de développement.

### 2.3.2 L'ordonnancement, une simple brique parmi les autres

Le comportement temporel des systèmes temps réel est souvent perçu uniquement au travers du filtre de l'ordonnancement, oubliant les relations entre cette brique et les autres éléments du système. Cela se retrouve, par exemple, au travers des propriétés vérifiées par les méthodes d'analyse d'ordonnançabilité qui sont rarement en relation immédiate avec les exigences fonctionnelles du système.

Pour illustrer ce point, prenons l'exemple des contraintes d'échéance pour une tâche. Ces contraintes sont considérées comme des propriétés directement induites par les performances attendues du système. Or, dans le cas d'un système de contrôle, les exigences sont plutôt exprimées en terme de temps de réponse (dans le sens de l'automatique) et de robustesse du système. Le respect des échéances des tâches devrait alors se lire au regard de ces critères de performance et non pas simplement en terme d'ordonnançabilité. D'ailleurs, comme le montrent les travaux de Andrianiaina *et al.* (2011) et de Cervin (2012), le non respect ponctuel d'une échéance peut être absorbé par la robustesse intrinsèque du système. Ainsi, le seul critère d'ordonnançabilité n'est pas pertinent du point de vue du système et les choix d'ordonnancement devraient être jugés au regard des performances globales.

Cette constatation a été le point de départ de mon intérêt pour les problématiques dites cyber-physiques et l'axe de recherche présenté dans la partie 2.4.4 en résulte. Bien que l'ordonnancement ne soit pas présenté comme étant au cœur de cet axe, il en fait partie

en tant que brique constitutive des systèmes cyber-physiques ayant des contraintes temps réel.

Enfin, la problématique exposée dans la partie 2.2.4, sur la prise en considération des incertitudes dans l'ordonnancement, est aussi liée à l'étude des relations entre l'ordonnancement et les autres composants physiques du système (principalement la mémoire). Nous ne développerons pas plus ce point ici, mais verrons son extension dans le projet présenté dans la partie 2.4.3.

## 2.4 Pespectives de recherche

Ce chapitre se conclue sur mes perspectives de recherche. Elles s'inscrivent dans la continuité des activités que j'ai menées sur les systèmes embarqués et adressent des domaines scientifiques allant au delà de l'ordonnancement temps réel.

D'un point de vue général, mon objectif est d'approfondir deux problématiques. La première traite de l'intégration, dans un processus de développement, des méthodes de conception et de vérification formelle pour les systèmes embarqués avec des contraintes de temps. La seconde aborde les interactions entre les parties logicielles et les composants physiques, sur le comportement temporel d'un système. Par « physique », j'entends aussi bien le support d'exécution informatique que le dispositif physique (l'équivalent de la partie opérative dans un automatisme) qui est en interaction avec la partie numérique.

Dans la suite, pour éviter les confusions, l'adjectif « matériel » est utilisé pour désigner le support d'exécution d'un système informatique (processeurs, mémoires, périphériques, etc.) et l'adjectif « physique » pour désigner les parties opératives connectées au système informatique.

### 2.4.1 Déclinaison des objectifs sous forme d'axes de recherche

Mes objectifs scientifiques sont présentés dans les sections suivantes sous la forme de trois axes. Chaque axe peut se lire selon un degré de maturité et de production des résultats à plus ou moins long terme. Ainsi :

— Le premier axe s'intéresse à la conception, la vérification et la génération de code d'applications temps réel. Le travail envisagé a pour but de produire une chaîne d'outils pour assister un concepteur dans le cadre de la méthode de développement MCSE (Calvez, 1990) et en utilisant les méthodes de vérification liées à la boîte à outils Tina (Berthomieu *et al.*, 2004b). Cet axe est mature dans le sens où il prolonge un travail en cours et s'appuie sur des outils existants. La partie 2.4.2 présente cet aspect en détaillant le projet actuel et les études qui en résultent.

— Le second axe traite des moyens pour rendre prévisibles les applications temps réel sur des architectures multiprocesseurs. Cet axe s'inscrit dans le prolongement de mes études sur l'ordonnancement temps réel sous incertitude (partie 2.2.4) et permettra d'approfondir cette thématique. La partie 2.4.3 introduit cet aspect et décline sa réalisation sous la forme d'un sujet de thèse afin de développer cette activité scientifique sur plusieurs années.

— Le dernier axe s'envisage à long terme et manifeste une volonté d'inscrire mes travaux dans le contexte des activités scientifiques liées aux systèmes cyber-physiques dans le domaine des systèmes autonomes. Cela passe par une meilleure appropriation de

la thématique ainsi que la participation à la communauté existante sur ce sujet. La partie 2.4.4 présente cet aspect et ce qui motive l'introduction des aspects cyber-physiques dans mes activités de recherche.

### 2.4.2 Axe 1 : Conception, vérification et génération de code d'applications temps réel

Ce premier axe a pour cadre un travail mené à la suite du projet ANR RESPECTED en collaboration avec Sébastien Piellement et Olivier Pasquier de l'Institut d'Électronique et de Télécommunications de Rennes (IETR) sur la méthode de conception MCSE. J'ai ensuite poursuivi cette étude lors de mon CRCT à l'ÉTS. Son extension à la génération d'exécutables vérifiés et ordonnancés a permis d'obtenir un financement de thèse CIFRE pour septembre 2017 avec Continental dans le cadre des projets soutenus par l'Agence De l'Environnement et de la Maîtrise de l'Énergie (ADEME).

Ce travail s'inscrit dans la continuité de celui réalisé sur le langage Pola (voir chapitre 6), et s'appuie sur les méthodes de vérification des systèmes ordonnancés telles que celles présentées dans le chapitre 5 et sur mon expérience de conception des systèmes temps réel.

Parallèlement au futur travail de thèse, des réflexions sont aussi en cours sur le couplage entre l'approche décrite ci-après et le langage Genom (Fleury *et al.*, 1996) qui est développé au LAAS-CNRS et utilisé pour décrire des systèmes logiciels embarqués dans des robots autonomes. Le but est de proposer à terme des outils de génération garantissant les propriétés temps réel et comportementales pour des systèmes robotisés.

Un travail collaboratif avec l'ÉTS sur cet aspect est aussi en cours de définition dans le cadre d'un congé sabbatique du professeur Abdelouahed Gherbi.

**Motivations.** Cet axe a pour but de répondre d'une manière pratique à la problématique évoquée dans la partie 2.3.1 autour de la mise en œuvre des méthodes liées à l'ordonnancement lors de la conception des systèmes. Ce travail n'intègre pas les phases initiales du développement (spécification) et se focalise sur la conception et le codage. Cependant des travaux futurs pourront être étendus aux problématiques liées, par exemple, à la formalisation des exigences lors de la phase de spécification.

Dans un premier temps, cette étude vise à la production d'une chaîne d'outils couvrant la conception, la vérification et la génération de code d'applications temporellement sûres. Une des motivations est de montrer la faisabilité d'une telle chaîne à partir des outils de vérification formelle développés dans l'équipe Vertics du LAAS. Cette réalisation devrait déboucher sur un outil de développement intégré et pouvant être employé pour des projets d'enseignement ou comme démonstrateur.

Dans un second temps, ce travail sera généralisé et formalisé afin de définir un modèle de *runtime* garantissant le comportement temporel de l'exécutable et pouvant être vérifié formellement.

**Contexte.** Le contexte général de ce travail est décrit par la figure 2.8 et consiste à organiser, définir et produire les outils nécessaires pour réaliser une chaîne de modélisation et de génération d'un exécutable dont le comportement est vérifié.

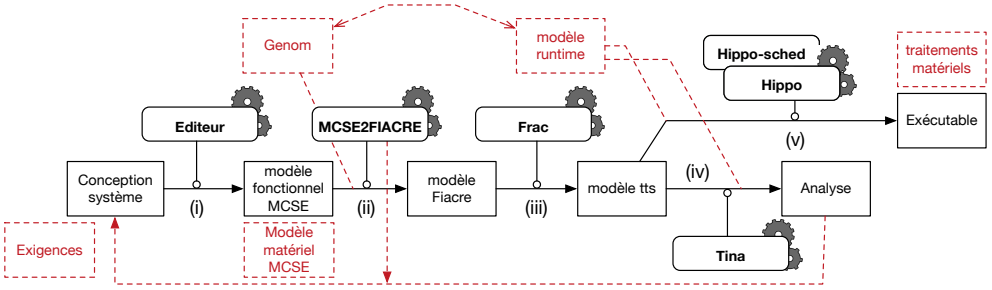



FIGURE 2.8 – Chaîne d'outils pour la conception, la vérification et l'exécution d'un système embarqué avec des contraintes temporelles. En pointillé, les éléments non traités et les extensions envisagées.

L'approche prend pour cadre une méthode baptisée « Méthodologie de Conception des Systèmes Embarqués » (MCSE) proposée par Jean-Paul Calvez (1990). C'est une méthode de conception descendante appliquée aux systèmes embarqués qui couvre les phases de spécification, de conception, d'implantation et de validation. Conjointement à la méthode MCSE, un langage spécifique de modélisation à base de composants est défini et permet de couvrir les dimensions parallèle et séquentielle d'une application. La sémantique du langage MCSE n'est pas formellement définie et la méthode ne préconise pas des techniques spécifiques de vérification ou de validation. Par contre, la méthode est particulièrement intéressante de par son caractère efficace et dédié à la conception des systèmes embarqués. Ainsi, sa syntaxe réduite permet de raisonnablement proposer des outils qui couvrent la totalité du modèle et d'en fixer la sémantique formellement.

Afin de mettre à disposition un outillage complet pour la vérification et la génération de code dans une démarche MCSE, nous nous appuyons sur la boîte à outils Tina (Berthomieu *et al.*, 2004b) qui est dédiée à la vérification formelle de systèmes décrits par des réseaux de Petri temporels. Outre les fonctionnalités classiques d'édition graphique, l'outil Tina propose (entre autres) un certain nombre de constructions de graphes de comportements pour les réseaux de Petri et les réseaux temporels ainsi que des model-checkers pour les logiques temporelles LTL et CTL.

Tina peut aussi traiter des entrées depuis une description de systèmes réalisée en Fiacre (Berthomieu *et al.*, 2008), un langage formel de plus haut niveau que les réseaux de Petri, qui permet de modéliser des systèmes d'un point de vue comportemental et temporel. Un ensemble de compilateurs est proposé pour accompagner ce langage afin de le traduire vers différents outils d'analyse. Dans le cas de Tina, un modèle Fiacre est compilé sous la forme d'un *Time Transition Systems* (TTS) pour pouvoir ensuite en faire la vérification.

Dans ce contexte, la chaîne décrite par la figure 2.8 est une instanciation de la phase de conception du cycle de développement de MCSE et comprend les étapes suivantes : (i) le résultat de la conception du système est modélisé selon le modèle fonctionnel MCSE, (ii) le modèle fonctionnel est traduit en Fiacre, (iii) le modèle est ensuite compilé à l'aide de l'outil `frac` sous forme d'un TTS qui servira d'entrée à Tina ; (iv) le modèle est utilisé par Tina afin de vérifier les propriétés temporelles et comportementales du système ; (v) un générateur, baptisé `hippo`, produit un exécutable depuis le TTS garantissant la même sémantique que le modèle vérifié. Au cours de la génération, le système doit aussi être vérifié du point de vue de l'ordonnancement (outil `hippo-sched`).

Cette chaîne est construite suivant des principes classiques pour vérifier le comportement de systèmes temporels. L'intérêt de passer par Fiacre est double (Berthomieu *et al.*, 2008) :

d'une part simplifier la traduction du modèle fonctionnel MCSE en utilisant les notions de processus, de synchronisation et de composants offertes par Fiacre (que l'on ne retrouve pas dans les formalismes bas-niveau comme les réseaux de Petri ou les automates), et d'autre part permettre l'utilisation d'outils de vérification autres que Tina sans avoir besoin de redéfinir la traduction. Nous verrons dans la suite que le choix a été de réduire au maximum l'impact de l'ordonnancement sur le comportement des éléments du système afin d'en facilité la vérification. Ainsi le langage Pola, décrit dans le chapitre 6, n'a pas été retenu bien que la démarche employée ici reste similaire à celle utilisée pour vérifier l'ordonnancement d'un modèle AADL.

Actuellement, les outils de vérification sont disponibles et matures, c'est-à-dire que la partie (iii)-(iv) de la chaîne existe et est outillée [7]. Les étapes (i) et (ii) suivent une démarche basée sur l'ingénierie des modèles avec la définition d'un méta-modèle du langage MCSE afin de produire un éditeur sous la forme d'un plugin Eclipse à l'aide du projet Sirius [8]. Ce méta-modèle est aussi employé lors de l'étape (ii) afin d'écrire une transformation entre les modèles MCSE et Fiacre. L'éditeur et la transformation sont implémentés sous forme de prototypes, mais nécessitent encore quelques modifications avant d'être pleinement opérationnels.

**Réalisations attendues.** Le travail qui sera réalisé se focalise sur les méthodes et outils de génération d'un exécutable temporellement vérifié en évitant d'introduire un écart de sémantique entre le modèle produit par le concepteur, le modèle vérifié et l'exécutable. Dans notre cas, nous commenceront par générer un code à partir du modèle le plus proche de celui vérifié, à savoir le TTS. Le support d'exécution n'est alors qu'une machine virtuelle pouvant exécuter le format TTS et garantissant ainsi le respect de la sémantique. Cette approche est similaire à ce qu'offre, par exemple, BIP (Sifakis, 2005) pour des systèmes non temporisés.

Comme exposé dans les parties antérieures de ce chapitre, nous avons vu qu'un point spécifique à l'exécution d'un modèle temporisé est la prise en charge du temps, aussi bien au niveau des modèles vérifiés que de l'exécutable. Pour traiter ce point, deux approches, la démarche synchrone et la démarche asynchrone, sont classiquement distinguées en fonction de la manière dont l'écoulement du temps est considéré et du moment de la prise en considération des événements. Il existe cependant une troisième approche, dite *asynchrone conduite par le temps* (*Time-Triggered*) et qui sépare les aspects logiques (fonctionnalité et chronométrage) des aspects liés au placement et à l'ordonnancement des tâches. Ainsi, la production des événements et des données se fait sous l'hypothèse synchrone (temps logiquement nul) et est conduite par le temps, alors que l'exécution des traitements est asynchrones. Kirsch et Sengupta (2007) décrivent ce principe à l'aide du *Logical Execution Time* (LET), qui est la durée séparant l'instant où un traitement peut commencer à s'exécuter et la date où il doit produire son résultat. Ainsi la date de production des données est parfaitement connue et ne dépend pas de la durée effective du traitement. Il est par contre nécessaire lors de la génération du système exécutable de s'assurer que l'exécution d'un traitement respecte bien son LET. Le *framework* pour exécutifs temps réel OASIS (Louise *et al.*, 2011) est un exemple mettant en œuvre cette démarche tout comme Giotto (Henzinger *et al.*, 2003). Le travail exposé dans cette partie est ainsi un complément aux travaux que j'ai pu mené sur les aspects asynchrones.

7. Les outils liés à Tina sont disponibles sur `http://projects.laas.fr/tina/` et ceux permettant de compiler du Fiacre sur `http://projects.laas.fr/fiacre/`.

8. Sirius est un projet soutenu par la fondation Eclipse et dédié à la conception d'interfaces graphiques. `https://projects.eclipse.org/projects/modeling.sirius` [consulté en février 2016]

La génération de code de l'étape (v) de la figure 2.8 suit une approche asynchrone conduite par le temps. Cette approche a été retenue pour deux raisons : (i) l'abstraction du temps physique facilite la vérification du comportement d'un système, en particulier en ignorant la préemption ; (ii) la notion de LET offre plus de souplesse que l'approche purement synchrone. Par contre, l'ordonnancement du système doit être vérifié lors de la génération afin de garantir que les traitements respectent bien leur LET et ceci en prenant en considération l'ordonnanceur du système qui accueille la machine d'exécution du modèle LET [9].

Les travaux à réaliser dans le cadre de la thèse avec Continentale sont donc multiples :

— réalisation de la machine d'exécution `hippo` : un prototype de la machine a été développé sous Xenomai [10], mais son comportement doit encore être prouvé. À cela s'ajoute une étude sur les moyens pour prendre en considération les événements asynchrones externes au système. Il est aussi envisagé de porter cette machine sur OSEK pour montrer la faisabilité de cette solution pour des architectures embarquées.

— compilation du modèle : la compilation du modèle Fiacre vers du code C embarqué a été réalisée afin d'en faire une preuve de faisabilité, mais elle doit être entièrement revue pour en optimiser les resources (taille du code, structures de données, etc.) et l'intégrer de manière entièrement automatisée dans la chaîne d'outils.

— validation de l'ordonnancement : une première ébauche d'analyse d'ordonnançabilité a été réalisée et procède par une méthode *brute force* énumérant tous les scénarios d'ordonnancement possibles afin d'en faire la vérification en s'inscrivant dans le contexte défini par Goossens *et al.* (2016) et en le couplant avec le simulateur SimSo (voir chapitre 7). Bien que les premiers résultats soient encourageants, de nombreuses optimisations sont encore à faire pour rendre efficace cette approche.

— transformation du modèle MCSE vers Fiacre : bien que l'hypothèse LET soit particulièrement adaptée pour gérer et vérifier le modèle MCSE, de légères adaptations du comportement de son modèle fonctionnel sont à réalisées ainsi que la formalisation de la transformation. Une étude pour garantir l'équivalence sémantique entre les deux modèles doit aussi être réalisée de manière approfondie.

**Ouvertures.** Ce travail offre de nombreuses perspectives et permet d'envisager différents prolongements suivant les résultats obtenus (voir éléments en pointillés rouges sur la figure 2.8). Les plus immédiats et découlant directement de la chaîne sont : mettre en place de nouveaux outils pour remonter les résultats de l'analyse au niveau du modèle MCSE, exprimer et traduire automatiquement les exigences en propriétés vérifiables, ou prendre en considération les composantes matérielles du système dans l'analyse et la génération.

D'une manière plus générale, si la démarche se révèle pertinente et permet d'obtenir une chaîne d'outils effective dans le cadre de MCSE, il est envisageable de la généraliser à d'autres langages en entrée. Ce point sera abordé à travers l'étude autour de Genom, un langage pour les robots qui est développé au LAAS. De la même manière que le langage spécifique de MCSE, Genom présente l'avantage d'être simple et pouvant être pris en charge exhaustivement. Partant de ces deux expériences, il sera intéressant de définir un cadre général pour exploiter Fiacre comme langage pivot en combinaison avec la définition formelle de *runtime* adapté à chaque langage.

Se pose alors la question de la modélisation du *runtime* afin de le rendre paramétrable, ainsi que les moyens pour garantir la cohérence entre le modèle vérifié et l'exécutable. En effet,

9. Une première étude sur la vérification de l'ordonnancement du système a été conduite en validant l'ensemble des traces produites par les activations des *opérations* du modèle MCSE.

10. https ://xenomai.org

l'activité de vérification est souvent réalisée sur des modèles dont le niveau d'abstraction ne prend pas en considération les mécanismes de la plateforme d'exécution et dont les outils de génération de code ne proposent pas de formalisation du système servant de support d'exécution, ici le *runtime* (Brun, 2010; Lelionnais *et al.*, 2014). Cela induit donc un écart entre le comportement vérifié et le comportement du système exécuté. Le travail consistera donc à définir formellement le *runtime* afin de l'intégrer dans le modèle formel vérifié.

### 2.4.3 Axe 2 : Conception de systèmes temps réel prévisibles sur des architectures multiprocesseurs

Ce deuxième axe est de nature plus prospective que le précédent et se décline sous la forme d'un sujet de thèse (demande de financement courant 2017 auprès de l'école doctorale). Ce travail s'inscrit dans la continuité de la thèse de Maxime Chéramy et poursuit mes activités sur l'ordonnancement temps réel en portant une attention particulière aux aspects matériels des systèmes.

**Motivations.** Les architectures multiprocesseurs actuelles introduisent un grand nombre d'incertitudes dans les modèles de comportement des systèmes temps réel. En effet, elles sont généralement conçues pour augmenter en moyenne les performances et non la prévisibilité temporelle. L'exemple communément cité est l'utilisation des mémoires caches partagées par les différents processeurs et tâches qui induit des contentions sur les bus d'accès ainsi que des incertitudes sur l'état des mémoires. Concevoir un système temps réel prévisible en utilisant de telles architectures reste encore un challenge.

L'objectif de ce deuxième axe de recherche est d'étudier les méthodes et outils pour prendre en considération les incertitudes engendrées par le cache dans les système multiprocesseurs temps réel. Pour cela, nous procéderons à la mise en place d'une classification des techniques existantes, puis à l'évaluation et la comparaison d'un sous-semble représentatif de ces méthodes.

**Contexte.** La prise en considération des perturbations temporelles induites par la partie matérielle dans la conception des systèmes embarqués temps réel n'est pas une nouveauté. Ainsi, depuis de nombreuses années, la communauté scientifique travaille sur des méthodes pour intégrer les éléments matériels des architectures informatiques (pipeline, cache, bus, etc.) aussi bien dans l'analyse des durée d'exécution des systèmes ordonnancés que pour la validation de leurs comportements temporels.

Il n'en reste pas moins de nombreux problèmes à résoudre dus à l'utilisation croissante des architectures multiprocesseurs pour les systèmes embarqués. Problèmes d'autant plus présents que l'offre actuelle des fondeurs privilégie la performance sur le déterminisme.

Concevoir des systèmes critiques temps réel pour des architectures multiprocesseurs nécessite de garantir leur prévisibilité, c'est-à-dire que l'on puisse prédire leur comportement futur pour garantir certaines propriétés. Cette notion est ancienne et a été introduite par Stankovic et Ramamritham (1990) et a été redéfinie et étendue plusieurs fois. Récemment, Axer *et al.* (2014) ont formalisés la notion de prévisibilité (*predictability*) pour un système comme étant une propriété qui peut être prédite par une analyse ad-hoc, la précision de l'analyse étant limitée par les incertitudes (« *The notion of predictability should capture if, and to what level of precision, a specific property of a system can be predicted by a*

*system-specific optimal analysis. It is the source of uncertainty that limits the precision of an analysis* », Thesis 2.1, p. 82:5).

Proposer des méthodes pour prédire temporellement le comportement d'un systèmes informatique est rendu difficile par les nombreux facteurs impliqués et qui entrent en jeu dans la conception du système : processeur, jeu d'instructions, langage, compilateur, conception logiciel, systèmes d'exploitation, ordonnancement, communication, etc. De plus, les architectures de processeurs modernes utilisent des éléments comme les pipelines, le multhreading ou les mémoires partagées qui introduisent de nombreuses incertitudes [11] sur l'exécution, rendant difficile la prévisibilité. Cette constatation est d'autant plus vraie pour les architectures multiprocesseurs pour lesquelles de nombreux éléments sont partagés entre les différents flux d'exécution du système.

Afin de contourner ces problèmes, la démarche de conception usuellement utilisée consiste à sur-dimensionner les marges d'utilisation des ressources physiques. L'inconvénient d'une telle approche est, d'une part, que le respect des exigences n'est pas pour autant complètement garanti, et d'autre part, que les ressources sont gaspillées et mal exploitées. Proposer de nouvelles méthodes pour mieux prendre en considération la prévisibilité dans un processus de conception sans sur-dimensionner le système semble nécessaire.

Pour cela deux tendances dans les travaux de recherche cohabitent. La première consiste à approfondir les méthodes d'analyse pour « augmenter » la prévisibilité du système et en réduire les incertitudes. La seconde consiste à proposer des architectures matérielles et logicielles dont la prévisibilité est plus facile à garantir. Ces deux approches sont bien sûr complémentaires et doivent être couplées afin de produire des solutions matures pour la conception des systèmes.

L'article de Axer *et al.* (2014) propose une synthèse des différents travaux existants pour concevoir des systèmes temps réel prévisibles. En ce qui concerne les architectures multicœurs, les auteurs distinguent les méthodes pour isoler les interférences temporelles entre les tâches afin d'en prédire les temps d'exécution (par exemple le partitionnement du cache), des méthodes au niveau du systèmes d'exploitation, en particulier l'ordonnancement, pour réduire les surcoûts et les variations temporels. Cette problématique rejoint l'étude réalisée dans le cadre de la thèse de Maxime Chéramy (voir partie 8.3).

En dehors de l'aspect multiprocesseur, de nombreuses approches abordent aussi la prévisibilité des systèmes. On peut citer, sans être exhaustifs, des compilateurs intégrant des analyses de pire temps d'exécution ou optimisant l'allocation mémoire pour en réduire les latences d'accès, des jeux d'instructions facilitant la prévisibilité temporelle du matériel, ou des micro-architectures dédiées. Dans cette dernière catégorie, citons à titre d'exemple, deux approches distinctes. La première, menée dans le cadre du projet « Precision Timed » [12] (PRET) à l'Université de Berkeley, propose une architecture optimisée pour garantir de bonnes performances de calcul tout en préservant la prévisibilité temporelle et étend le jeu d'instructions de la machine par des mécanismes pour contrôler le temps d'exécution des traitements. La seconde approche adoptée dans les projets européens PROARTIS [13] puis PROXIMA [14] consiste à exploiter au maximum l'aléatoire des architectures, voire à introduire volontairement de l'aléatoire pour garantir des hypothèses mathématiques propre aux analyses stochastiques (Kosmidis *et al.*, 2013).

11. Les mécanismes sont déterministes, mais connaître l'état du système à un instant donné est quasiment impossible ou trop complexe à modéliser pour mener une analyse efficace. Il est alors nécessaire de travailler sur des modèles probabilistes et prévisibles.
12. `https://chess.eecs.berkeley.edu/pret/` [consulté juillet 2016]
13. `https://www.proartis-project.eu` [consulté juillet 2016]
14. `http://www.proxima-project.eu` [consulté juillet 2016]

**Réalisations attendues.** La prise en considération des incertitudes sur les durées d'exécution dans la validation et la conception des systèmes temps réel est un sujet actif et offrant des perspectives intéressantes. Afin d'aborder cette problématique et de l'explorer, nous nous intéresserons à l'évaluation des méthodes et techniques pour prendre en considération les effets dus au cache dans des architectures multiprocesseurs. La question sous-jacente étant de mieux cerner l'intérêt d'utiliser ces méthodes pour un concepteur de systèmes critiques. Cette étude sera menée dans le cadre d'une thèse pour pouvoir l'inscrire dans la durée.

Une première partie du travail sera consacrée à classer les méthodes servant à maîtriser les effets induits par le cache sur les systèmes temps réel multiprocesseurs. Cette classification devra permettre de décrire les méthodes suivant différentes perspectives, comme par exemple les composants du système impliqués (code compilé, protocole d'accès, etc.) ou les types de gains obtenus (réduction des incertitudes, etc.). Il s'agira donc dans un premier temps de s'approprier la littérature sur le sujet, puis de concevoir un réseau sémantique pour en classer les diverses approches. Si la variété des méthodes se révèle trop vaste pour espérer approcher l'exhaustivité, le champ sera réduit en ne considérant que certains types d'approches, par exemple en ignorant celles reposant sur la génération d'architecture matérielle dédiée.

La suite du travail consistera à sélectionner un sous-ensemble de ces méthodes, tout en essayant d'être représentatif de la diversité des approches, puis de les évaluer sur des *benchmarks* afin de les comparer. Les méthodes retenues pour l'étude dépendront donc de la disponibilité et de la maturité des outils disponibles pour les mettre en œuvre. Le cadre méthodologique sera aussi à définir et nécessitera le développement d'outils *ad hoc*.

Cette comparaison commencera par la définition de critères pour mettre en perspective ces techniques par rapport aux besoins d'un concepteur de système, par exemple, un critère pour mesurer le degré de sur-dimensionnement du système, ou un critère tel que le taux de défaillances estimé. Il faudra ensuite procéder aux évaluations et à la comparaison des méthodes.

Enfin, la dernière partie du travail consistera à ébaucher une démarche de conception utilisant ces techniques dans le développement de systèmes temps réel critiques. Cela passera par une mise en œuvre de cette démarche en s'attachant à en montrer sa faisabilité et son applicabilité.

### 2.4.4 Axe 3 : Méthodes de conception des systèmes cyber-physiques

Ce dernier axe décrit une volonté pour inscrire à plus long terme mes travaux dans le contexte des activités scientifiques liées aux systèmes cyber-physiques. Cela passe par une implication au sein de la communauté scientifique traitant de ce sujet et par une appropriation des problématiques propres aux systèmes cyber-physiques.

**Motivations et perspectives.** Les systèmes cyber-physiques sont définis comme étant des « *engineered systems of synergistically interacting physical and computational components. The key property of these systems is that functionality and salient system properties are emerging from an intensive interaction of physical and computational components. As the computational components are aware of their physical context, they are intrinsically distributed, (time)-synchronizing, have to cope with uncertainty of sensoric-input and need to produce real-time reactions.* » (Giese *et al.*, 2012)

Autrement dit, les systèmes cyber-physiques sont des systèmes vus dans leur globalité et dont les composants informatiques peuvent être des systèmes embarqués temps réel. Ces systèmes ne sont pas nouveaux en soi. Un avion ou une ferme solaire sont des systèmes cyber-physiques. La problématique scientifique dites cyber-physiques ne concerne donc pas de nouveaux systèmes, mais émerge de l'absence, ou de l'inadéquation, de méthodes pour traiter conjointement des aspects physiques et informatiques lors de leur conception.

En effet, actuellement, ces systèmes sont conçus à l'aide de formalismes et d'outils variés où chaque modèle et méthode se focalise sur des propriétés spécifiques en ignorant les autres afin d'en permettre l'analyse. Bien que cette séparation permette de concevoir de manière effective des systèmes cyber-physiques, elle pose des problèmes pour valider la conception d'un système dans sa globalité et pour étudier les impacts que peuvent avoir les choix de conception qui sont à l'intersection de ces deux mondes. Ainsi, les travaux dans le domaine du cyber-physique visent à réduire cet écart et à proposer des méthodes intégrant les aspects physiques et informatiques (Lee, 2015).

En ce qui concernes mes travaux, souhaitant approfondir l'étude des interactions entre les éléments informatiques et les composantes physiques dans les systèmes, les étendre à la problématique des systèmes cyber-physiques semble naturel. Cependant l'une des difficultés pour traiter ce sujet est de couvrir un ensemble de compétences suffisamment variées et complémentaires pour appréhender les systèmes cyber-physiques dans leur globalité et non pas uniquement sur leurs seules composantes physiques ou « cybers ». Ainsi, il me semble essentiel de collaborer sur cette problématique avec un ensemble de chercheurs aux spécialités de recherche différentes, mais ciblées, et de mettre en place des activités de recherche transverses. En cela, le LAAS-CNRS avec le projet Adream [15], ainsi que l'environnement toulousain (laboratoires et industriels), offre un cadre approprié pour débuter cette démarche et ensuite développer des collaborations extérieures [16].

Afin d'initier cela, j'envisage de me baser sur les travaux présentés dans la partie 2.4.2 pour les intégrer dans un processus de conception où le système n'est pas simplement réduit à sa partie informatique, mais où les parties physiques sont aussi intégrées. Les systèmes envisagés pour mener ces expérimentations et étudier l'impact des choix d'implémentation sur les propriétés physiques sont ceux disponibles sur la plateforme Adream, en particulier des robots et des systèmes mobiles autonomes.

15. `https://www.laas.fr/public/fr/adream` [consulté juillet 2016]

16. Depuis février 2016, l'INSA de Toulouse a mis en place un programme d'aide à la « mobilité seule », c'est-à-dire non couplé à un CRCT, afin de soutenir des projets courts (minimum 1 mois) de mobilité sortante. J'envisage d'utiliser cette possibilité pour réaliser une mobilité dans une université reconnue dans le domaine des systèmes cyber-physiques dès juillet 2017.

Deuxième partie

# Exposés scientifiques détaillés

CHAPITRE 3

# La PPC au service de l'ordonnancement temps réel

Cette partie présente une méthode basée sur la programmation par contraintes pour allouer des tâches sur des ressources d'exécution distribuées en respectant des exigences fonctionnelles et non-fonctionnelles (temporelle, ressources, énergie, etc.). En cas d'échec, des explications sont produites (expliquant pourquoi l'allocation n'est pas faisable) pour guider le concepteur dans sa nouvelle conception. Ces travaux ont été publiés dans (Cambazard *et al.*, 2004a; Cambazard et Hladik, 2004; Cambazard *et al.*, 2004b; Hladik *et al.*, 2005b,a; Hladik et Déplanche, 2005; Hladik *et al.*, 2006b,a; Cambazard *et al.*, 2007; Hladik *et al.*, 2008). L'étude a été réalisée en deux temps. En premier lieu, au cours de ma thèse, une approche par décomposition a été proposée (voir partie 3.4), puis, dans les années suivantes, une contrainte globale dédiée à l'ordonnancement à priorité fixe a été implémentée (voir partie 3.3). Originellement réalisée pour les systèmes distribués, cette étude a naturellement été étendue au cas des architectures multiprocesseurs avec un ordonnancement partitionné (Hladik et Déplanche, 2009).

Le problème d'allocation pour les systèmes temps réel a été très largement étudié et une classification en est difficile à cause de la variété des modèles des architectures logicielles (périodiques, échéances, précédences, etc.) et matérielles (multiprocesseur, distribué, hétérogène, etc.), des contraintes considérées (mémoire, redondance, etc.), des critères d'optimisation (minimisation des ressources, répartition des charges, etc.), et des stratégies de résolution (programmation linéaire, méta-heuristiques, etc.). De complexité NP-difficile (Lawler, 1983), ce problème a été abordé à l'aide de nombreuses méthodes énumératives et heuristiques : théorie des graphes, séparation et évaluation, algorithme génétiques, regroupement, etc. Cependant, aucune de ces méthodes ne semble se démarquer plus qu'une autre. De plus, la majorité de ces techniques souffre de deux défauts : i) l'implémentation est fortement couplée au modèle considéré ou à l'objectif, ce qui rend difficile, voire impossible d'étendre la méthode à de nouvelles contraintes, ii) leurs performances sont très sensibles aux paramètres initiaux et les algorithmes nécessitent d'être expérimentalement réglés.

Le travail exposé ci-après se base sur la programmation par contraintes (ppc) pour résoudre le problème d'allocation. Les avantages principaux de la ppc sont : sa *déclarativité* : les variables, domaines et contraintes sont explicitement décrits ; sa *généricité* : les techniques de résolutions ne sont pas dépendantes du problème, des mécanismes généraux sont utilisés pendant la recherche ; son *adaptabilité* : chaque contrainte peut être considérée comme

indépendante et le modèle peut être simplement étendu en agrégeant les contraintes ; son *indépendance aux paramètres* du problèmes. Cette technique a été largement utilisée pour résoudre des problèmes combinatoires et a prouvé son efficacité pour de nombreuses applications comme la planification ou l'ordonnancement dans des domaines aussi variés que la finance ou la biologie. Bien que les travaux de Szymanek *et al.* (2000), Schild et Würtz (2000) et Ekelin (2004) utilisent la ppc pour produire une séquence d'ordonnancement pour des systèmes distribués, ce travail a été le premier à proposer une résolution du problème d'allocation d'un système temps réel dur asyncrhone avec la ppc.

Deux approches ont été considérées. La première introduit une contrainte globale et un algorithme *ad hoc* de filtrage pour modéliser l'ordonnancement. Cet algorithme est optimisé pour prendre en considération les spécificités et les structures propres à l'ordonnançabilité temps réel à priorités fixes. La seconde approche utilise la complémentarité de la ppc et des méthodes d'optimisation pour la recherche opérationnelle avec des approches hybrides (Thorsteinsson, 2001; Jain et Grossmann, 2001; Benoist *et al.*, 2002). Cette méthode repose sur la décomposition de Benders (Hooker et Ottoson, 2003) qui sépare le problème d'allocation de celui de l'ordonnancement : le problème d'allocation est résolu à l'aide des outils de ppc dynamique, alors que le problème d'ordonnancement est traité de manière spécifique par une analyse d'ordonnançabilité. L'idée principale est d'apprendre de l'analyse d'ordonnançabilité pour introduire dynamiquement des contraintes dans le problème d'allocation afin de réduire l'espace de recherche. Nous pouvons ainsi comparer cette approche à une démarche d'apprentissage par l'échec.

Les expérimentations menées montrent que ces deux méthodes permettent de résoudre le problème d'allocation de manière efficace. De plus, une propriété fondamentale de ces méthodes est leur complétude : quand un problème n'a pas de solution, il est possible de le prouver et d'en produire une explication. Cette explication pouvant être utiliser pour guider le concepteur pour revoir son architecture.

Aucune preuve n'est fournie dans ce document, le lecteur intéressé peut se référer aux articles publiés pour en avoir les détails. De même, pour des raisons de concision nous ne prenons pas en compte les communications dans ce document. Le problème ainsi étudié est équivalent au problème de partitionnement en ordonnancement multiprocesseur. Les communications ont été prises en considération dans le problème d'origine (Hladik *et al.*, 2008) et ont fait l'objet d'un rapport distinct en ce qui concerne le réseau CAN (Hladik et Déplanche, 2005).

## 3.1 Modélisation du problème

L'architecture logicielle est modélisée par un ensemble des tâches $\mathcal{T} = \{1, ..., N\}$. Une tâche $i$ est définie par ses caractéristiques temporelles et ses besoins en ressource : sa période, $p_i$ ; son pire temps d'exécution sans préemption, $e_i$ ; et son besoin mémoire, $m_i$. Une priorité, $\pi_i$ est attachée à chaque tâche $i$ et la tâche $j$ est dite plus prioritaire que $i$ si et seulement si $\pi_j > \pi_i$.

L'architecture matérielle est constituée d'un ensemble $\mathcal{P} = \{1, ..., M\}$ de $M$ processeurs ayant des capacités mémoire $\mu_k$. Tous les processeurs de $\mathcal{P}$ ont la même vitesse d'exécution et peuvent communiquer entre eux sans surcoût. Chaque processeur est associé à un ordonnanceur à priorité fixe préemptif. Les tâches ne peuvent pas migrer.

Un placement est une application $A : \mathcal{T} \to \mathcal{P}$ qui place une tâche $i$ sur un processeur $k : A(i) = k$. Le problème d'allocation consiste à trouver une application $A$ qui respecte toutes les contraintes décrites ci-après :

— **Capacité mémoire** : la mémoire utilisée sur un processeur $p_k$ ne peut pas excéder sa capacité ($\mu_k$), soit $\forall k = 1..m, \sum_{A(i)=k} m_i \leq \mu_k$

— **Facteur d'utilisation** : le facteur d'utilisation d'un processeur ne peut pas excéder sa propre capacité de traitement soit $\forall k = 1..m, \sum_{A(i)=k} \frac{e_i}{p_i} \leq 1$

— **Résidence** : une tâche peut avoir besoin d'une ressource logicielle ou matérielle spécifique qui n'est disponible que sur certains processeurs (par exemple, une tâche qui traite l'acquisition d'un capteur doit être présente sur le processeur connecté à l'entrée du périphérique). Une telle contrainte est définie par un couple $(i, \alpha)$ où $i \in \mathcal{T}$ est une tâche et $\alpha \subseteq \mathcal{P}$ est un ensemble de processeurs sur lesquels sont disponibles la ressource. Un placement $A$ doit respecter $A(i) \in \alpha$

— **Co-résidence** : cette contrainte impose que plusieurs tâches soient placées sur un même processeur (par exemple elles partagent une ressource commune). Cette contrainte est définie par un ensemble de tâches $\beta \subseteq \mathcal{T}$ et doit respecter pour un placement $A$ $\forall (i, j) \in \beta^2, A(i) = A(j)$

— **Exclusion** : certaines tâches doivent être répliquées pour des raisons de tolérance aux fautes et ne doivent pas être assignées sur un même processeur. Cela correspond à un ensemble $\gamma \subseteq \mathcal{T}$ de tâches qui ne doivent pas être placées ensemble. Un placement $A$ doit satisfaire $\forall (i, j) \in \gamma^2, A(i) \neq A(j)$

— **Échéance** : le temps de réponse d'une tâche $i$, c'est-à-dire la durée entre son activation et sa terminaison, doit être borné par sa période $p_i$. Pour cela, nous utilisons la condition nécessaire et suffisante sur le pire temps de réponse d'une tâche proposée par Lehoczky (1990) : $R_i = e_i + \sum_{j \in hp(i,A)} \left\lceil \frac{R_i}{p_j} \right\rceil e_j$ avec $hp(i, A)$ l'ensemble des tâches plus prioritaires que la tâche $i$ et placées sur le processeur $A(i)$. La résolution de cette équation est discutée dans la partie suivante.

## 3.2 Le modèle de contraintes

Le modèle de contraintes utilisé est basé sur une formulation redondante de deux ensembles de variables : $x$ et $y$. Commençons par considérer les $n$ variables $x$ (nos variables de décision) qui correspondent chacune à une tâche et représentent le processeur choisi pour cette tâche : $\forall i \in \mathcal{T},\ x_i \in [1..M]$. Les variables booléennes $y$ indiquent la présence ou non d'une tâche sur un processeur : $\forall i \in \mathcal{T}, \forall p \in \mathcal{P},\ y_{ip} \in \{0, 1\}$. Des contraintes d'intégrité (*channeling constraints*) sont utilisées pour assurer la consistance du modèle. Le tableau 3.1 donne le modèle de contraintes résultant sans considérer la contrainte sur les échéances.

**Variables**
$x_i = [1..m], \forall 0 < i \leq n$
$y_{ip} = \{0, 1\}, \forall 0 < i \leq n, 0 < p \leq m$

**Contraintes**

| | |
|---|---|
| Résidence | $\forall (i, \alpha),\ x_i \neq \alpha$ |
| Co-Résidence | $\forall (i, j) \in \beta^2, x_i = x_j$ |
| Exclusion | $alldifferent(x_i \mid i \in \gamma)$ |
| Capacité mémoire | $\forall p \in \{1..M\},\ \sum_{i \in \{1..N\}} y_{ip} \times m_i \leq \mu_p$ |
| Facteur d'utilisation | $\forall p \in \{1..M\}, \quad \sum_{i \in \{1..N\}} \frac{ppcm(p) \times e_i \times y_{ip}}{p_i} \leq ppcm(p)$ |
| Intégrité du modèle 1 | $\forall i, x_i = j \Leftrightarrow y_{ij} = 1$ |

TABLE 3.1 – Modèle de contraintes (sans celles traitant de l'ordonnançabilité)

Remarquons que les facteurs d'utilisation sont reformulés à l'aide du *ppcm* (plus petit commun multiple) des périodes des tâches car le solveur de contraintes sur lequel nous avons travaillé ne peut pas prendre en considération des contraintes avec des coefficients réels et des variables entières.

Ce modèle ne tient pas compte des contraintes temporelles. Les équations de calcul du pire temps de réponse on pour forme

$$r = \sum_{i=1}^{k} \left\lceil \frac{r}{a_i} \right\rceil b_i + c \tag{3.1}$$

où $r$ est un entier positif. En notant $f(r) = \sum_{i=1}^{k} \left\lceil \frac{r}{a_i} \right\rceil b_i + c$, ce problème consiste à trouver le plus petit point fixe positif $r^*$ de $f$, soit $\min\{r^* > 0, f(r^*) = r^*\}$. À cause de la fonction partie entière la résolution directe n'est pas immédiate. Il est facile de montrer que le point fixe peut-être atteint en un nombre fini de pas, mais avec une complexité pseudo-polynomiale. La contrainte temporelle est donc difficile à exprimer par une contrainte primitive du solveur. Les parties suivantes décrivent deux moyens pour intégrer les contraintes temporelles, la première sous la forme d'une contrainte globale et la seconde basée sur la décomposition du problème.

## 3.3 Contrainte globale d'ordonnançabilité

La contrainte globale `schedulable` prend en charge la résolution de l'équation (3.1). Elle est définie par trois paramètres : un ensemble $X = \{x_1, \ldots, x_N\}$ de $N$ variables, une architecture matérielle définie par $\mathcal{P}$ et une architecture logicielle $\mathcal{T}$. La sémantique résultante de la contrainte est :

**Définition 3.1.** *`schedulable`$(X, \mathcal{P}, \mathcal{T})$ est vérifiée, si et seulement si, $X$ définit un placement des tâches de $\mathcal{T}$ sur $\mathcal{P}$ tel que $\forall i \in \mathcal{T} \quad R_i \leq p_i$ avec $R_i$ est le pire temps de réponse de la tâche $i$.*

En d'autres termes, la contrainte est vérifiée, si et seulement si, le placement $X$ est ordonnançable. Un premier résultat important peut-être obtenu pour cette contrainte :

**Propriété 3.3.1.** *Assurer l'arc-consistance pour la contrainte `schedulable` est NP-dur.*

Notre algorithme de filtrage est basé sur une forme plus faible de consistance obtenue à l'aide de certains raisonnements hypothétiques. Le filtrage est essentiellement incrémental. Partant d'un placement partiel qui est ordonnançable, nous prenons en considération le fait que placer une tâche sur un processeur (instanciation de variable) permet d'interdire le placement de certaines tâches sur des processeurs (retrait de valeur).

Plus formellement, un placement partiel est une application $a : \mathcal{U} \subset \mathcal{T} \rightarrow \mathcal{P} \cup \{-1\}$ telle que

$$\begin{array}{ll} i \in \mathcal{U} & \mapsto a(i) = p_k \\ i \in \mathcal{T} - \mathcal{U} & \mapsto a(i) = -1 \end{array} \tag{3.2}$$

Une extension $a'$ d'un placement partiel $a$ par l'affectation d'une tâche $i$ sur un processeur $p$ $(x_i = p)$ telle que $a(i) = -1$ est définie par $\forall t_k \in \mathcal{U}$, $a'(k) = a(k)$ et $a'(i) = p$. Elle est notée $a' \leftarrow a + (x_i = p)$.

L'algorithme 1 présente le pseudo-code de la contrainte pour intégrer l'ordonnançabilité. Il commence par vérifier l'ordonnançabilité de l'extension du placement partiel (ligne 2). Cela est facilement réalisable en calculant le pire temps de réponse des seules tâches déjà allouées. Si l'allocation est ordonnançable, l'algortihme de filtrage supprime les valeurs inconsistantes (ligne 3 à 14) en testant l'ordonnançabilité de chaque valeur de toutes les variables restantes (ligne 8). L'élagage est ainsi propagé à travers l'ensemble des contraintes jusqu'à atteindre un point fixe (boucle while).

**Algorithme 1** Algorithme de filtrage de la contrainte globale pour le placement $(x_i = p)$

```
1:  procédure GLOBAL CONSTRAINT(a' ← a + (x_i = p)))
2:      flag := CHECKSCHEDULABILITY(a')      ▷ flag devient vraie si a' est ordonnançable
3:      tant que flag faire
4:          flag := false
5:          pour each unallocated task j faire
6:              pour each q in x_j's domain faire
7:                  a'' := a' + (x_j = q)
8:                  si not CHECKSCHEDULABILITY(a'') alors
9:                      Remove q from domain of x_j   ▷ Propagation aux autres contraintes
10:                     flag := true
11:                 fin si
12:             fin pour
13:         fin pour
14:     fin tant que
15: fin procédure
```

Pour réduire le nombre d'appels à CHECKSCHEDULABILITY et ainsi réduire le nombre de calculs de pire temps de réponse, il est possible de pré-calculer des règles de dominance entre les tâches :

**Propriété 3.3.2.** *Si $x_i = p_k$ rend la tâche $i$ non-ordonnançable, alors $k$ peut être supprimé du domaine des variables $x_j$ telles que $\pi_j < \pi_i \wedge e_j \geq e_i \wedge p_j \leq p_i$.*

**Propriété 3.3.3.** *Si $x_i = p_k$ rend la tâche $b$ non-ordonnançable, alors $k$ peut être supprimé du domaine des variables $x_j$ telles que $\pi_j > \pi_b \wedge e_j \geq e_i \wedge p_j \leq p_i$.*

De plus, le fonctionnement de notre algorithme de filtrage — il est important de garder en mémoire que pour un placement partiel ordonnançable $a$ et son extension $a' \leftarrow a + (x_i = p)$ — seule l'ordonnançabilité des tâches placées sur $p$ qui ont une priorité inférieure ou égale à celle de la tâche $i$ doit être testée. Cela réduit grandement la complexité de CHECKSCHEDULABILITY et accroit l'efficacité de la contrainte.

## 3.4 Modélisation du problème par décomposition de Benders

La méthode proposée dans cette partie s'inscrit dans le cadre de la théorie *Logic-Based Benders Decomposition* développée par Hooker et Ottoson (2003) dans laquelle un problème linéaire est décomposé suivant ses variables en un problème maître et un problème esclave. Les deux sous-problèmes sont ainsi simplifiés et découplés en problèmes plus petits et donc potentiellement plus facile à traiter. Un mécanisme d'apprentissage permet ensuite de faire coopérer ces deux sous-problèmes. Dans le cadre de la décomposition de

Benders, l'apprentissage tire profit de la linéarité du problème esclave par l'intermédiaire de la résolution de son dual. À partie de cette résolution, des *coupes* sont produites sur le domaine de recherche du problème maître.

Les travaux de Hooker se placent dans un cadre plus général que la décomposition de Benders. Pour cela, il élargit la notion de dual à la logique en introduisant des *inferences duales* pour tout type de problème. La dualité se réfère alors à la capacité de produire la preuve logique de l'optimalité du problème esclave et de la validité de la coupe fournie.

Dans le contexte d'un sous-problème discret de satisfaction de contraintes, qui est le nôtre, la preuve est celle de l'infaisabilité du sous-problème. Notre approche se situe dans ce cadre avec le problème de l'ordonnançabilité considéré comme un sous-problème logique et permettant ainsi d'intégrer la programmation par contraintes dans un schéma de décomposition de type Benders (Thorsteinsson, 2001; Benoist *et al.*, 2002). La formalisation de la résolution passe ensuite par la définition des coupes que nous pouvons fournir au problème maître.

La décomposition de notre problème est faite selon ses contraintes (placement et ressources d'une part, temporelles d'autre part) sans partitionnement explicite des variables. La figure 3.1 illustre la stratégie que nous adoptons. Elle consiste à rechercher un placement admissible pour le problème maître à l'aide de la ppc puis à soumettre la solution trouvée à un test d'ordonnançabilité. Le problème esclave vérifie l'ordonnançabilité du placement et en cas de preuve de son infaisabilité retourne un ensemble de contraintes symboliques et d'inégalités arithmétiques (les coupes) qui identifient pourquoi cette allocation n'est pas ordonnançable et qui servent à éliminer toutes les solutions du problème maître qui partagent ces caractéristiques. Ensuite, une nouvelle solution du problème maître est recherchée et soumise au test d'ordonnançabilité, et ainsi de suite.

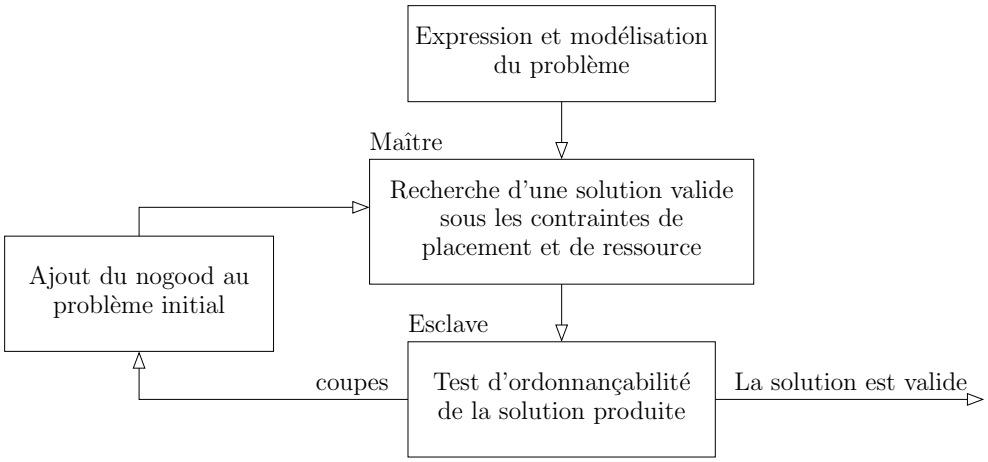



FIGURE 3.1 – Représentation schématique de la résolution du problème de placement

L'efficacité de la résolution du problème global (recherche d'une solution valide) dépend fortement de la manière dont les problèmes coopèrent. Pour cela, nous devons retourner des coupes pertinentes pour le problème maître, mais aussi mettre en œuvre d'autres mécanismes pour améliorer la recherche : re-modélisation du problème, heuristiques de but, etc.

L'ajout de nouvelles contraintes au problème maître suite à un test d'ordonnançabilité est mise en œuvre par l'utilisation d'*explications* (Jussien, 2001). Une explication est un ensemble de contraintes $C'$ (sous-ensemble des contraintes originelles du problème initial) et d'un ensemble de décisions prises pendant la recherche (la décision $i$ sera notée $dc_i$),

permettant ainsi d'en garder la trace. Par exemple, l'explication de pourquoi la valeur $a$ de la variable $v$ a été retirée suite à des décisions $dc_i$, s'écrit :

$$C' \wedge dc_1 \wedge dc_2 \cdots \wedge dc_k \Rightarrow v \neq a$$

Une contradiction survient lorsque le domaine d'une variable $x$ se vide, ce qui est équivalent à ne trouver aucune solution au problème. L'explication de cette contradiction est calculée par l'union de toutes les explications de chaque valeur de $x$. Dans le cas d'une contradiction, les algorithmes de *backtrack* classiques remettent en question la décision prise sur la variable précédente de $x$. Avec des explications, des algorithmes intelligents de *backtrack* peuvent être envisagés, permettant ainsi de revenir sur le choix d'une décision antérieure qui est la cause de la contradiction. Ainsi, en conservant les explications, il est possible de mettre en œuvre un mécanisme d'apprentissage par l'échec.

Dans le cadre de notre problème, si à la suite de la production d'une coupe par le problème esclave et à la propagation de cette contrainte, nous obtenons une contradiction. L'explication de cette contradiction permet alors de savoir quelles décisions sont la cause de la contradiction et ainsi de revenir directement dessus. Si plusieurs décisions en sont la cause, pour conserver la complétude de la méthode, il faut revenir sur la dernière décision prise.

Lorsqu'un placement partiel est produit par le problème maître, la condition sur le pire temps de réponse est vérifiée pour l'ensemble des tâches. Si une tâche est trouvée non ordonnançable, nous identifions un sous-ensemble de tâches pour lesquelles nous sommes certain que la tâche non ordonnançable le restera quel que soit le placement des autres tâches. Pour une tâche $i$ non ordonnançable pour un placement $a$, ce sous-ensemble est constitué de la tâche $i$ et des tâches appartenant à $hp(i, a)$. En effet, si pour un placement $a'$, nous avons $hp(i, a) \subseteq hp(i, a')$ alors la tâche $i$ n'est pas ordonnançable pour le placement $a'$. Nous pouvons donc interdire toutes les solutions du problème maître produisant ce sous-ensemble.

La propriété « il faut interdire qu'une tâche non-ordonnançable et les tâches plus prioritaires qu'elle sur le même processeur soient de nouveau placées sur un même processeur » se traduit par une contrainte de type *NotAllEqual* sur les variables $x_j$ (il ne faut pas que les $x_j$ soient tous égaux) :

$$NotAllEqual\{x_j | t_j \in hp(i, a) \cup \{t_i\}\} \tag{3.3}$$

Pour éviter d'avoir une accumulation de coupes inutiles qui entraînerait des surcoûts temporels pendant la résolution du problème maître, nous vérifions que des ensembles de contraintes ne sont pas inclus dans d'autres ensembles. Il est alors possible de rechercher une contrainte plus forte que celle de l'équation 3.3. Pour cela, nous recherchons un sous-ensemble minimal de $hp(i, a)$ tel que la tâche $i$ soit toujours non-ordonnançable. Pour ce faire nous nous sommes inspiré de l'algorithme QuickXplain (Junker, 2001) qui recherche un ensemble minimal de contraintes pour des problèmes linéaires. Pour cela, nous définissons pour une tâche $i$ et un ensemble $X$ de tâches plus prioritaires qu'elle, $R_i(X)$ le pire temps de réponse de la tâche $i$ en présence des tâches de $X$ :

$$R_i(X) = e_i + \sum_{j \in X} \left\lceil \frac{R_i}{p_j} \right\rceil e_j \tag{3.4}$$

et nous recherchons l'ensemble de tâches minimal $X$ tel que $X \subset hp(i, a)$ et $R_i(X) > p_i$. L'algorithme 2 décrit le principe exposé ci-avant. La recherche de tous les sous-ensembles

minimaux n'a pas de solution algorithmique simple et est coûteuse en temps de calcul, ce qui peut pénaliser la recherche d'un placement valide. De plus, surcharger le problème d'admissibilité par de nombreuses contraintes est aussi coûteux en temps de calcul. En effet, suite à une nouvelle prise de décision, l'ensemble des contraintes doit être propagé. Il est donc préférable de limiter le nombre de contraintes et de guider efficacement la recherche de solution, plutôt que d'apprendre beaucoup de nouvelles contraintes qui apportent peu d'information pour la résolution du problème.

**Algorithme 2** Algorithme de type QuickXplain dédié à la recherche d'un ensemble minimal de tâches non-ordonnançable

```
1: procédure QuickXPlainTask(i, a)
2:     X := ∅                                          ▷ phase d'initialisation
3:     σ_1, ..., σ_{#hp(i,a)} une énumération de hp(i, a)
4:     tant que R_i(X) ⩽ p_i faire
5:         k := 0
6:         Y := X
7:         tant que R_i(Y) ⩽ T_i et k < #hp(i, a) faire
8:             k := k + 1
9:             Y := Y ∪ {σ_k}
10:        fin tant que
11:        X := X ∪ {σ_k}
12:    fin tant que
13:    retourne X := X ∪ {σ_k}
14: fin procédure
```

## 3.5 Comparaisons des méthodes

Pour comparer l'approche par décomposition de Benders avec la contrainte globale, nous avons générer des configurations aléatoires. Pour cela, nous avons suivi les principes suivants :

— le nombre de tâches $n$ est fixé à 40. Les périodes sont générées de manière à limiter la valeur de $ppcm(p)$ en conservant un maximum de valeurs (Goossens et Macq, 2001). Les priorités sont affectées aléatoirement. Un paramètre, $\%_{global}$, est utilisé pour régler le facteur d'utilisation global tel que $\sum_{i=1}^{n} e_i/p_i = m\%_{global}$. La difficulté du problème d'ordonnançabilité dépend de $\%_{global}$ pour lequel les valeurs varient de 40 à 90.

— le nombre de processeurs, $m$, est fixé à 7. La capacité mémoire d'un processeur est générée telle que $\sum_{k=1}^{m} m_k = (1 + \%_{mem}) \sum_{i=1}^{n} \mu_i$ où $\%_{mem}$ représente la capacité mémoire supplémentaire disponible sur l'architecture matérielle.

— le pourcentage de tâches participant à une contrainte de placement est donné par le paramètre $\%_{res}$ pour les contraintes de résidence, $\%_{co}$ pour celles de co-résidence, et $\%_{exc}$ pour celles d'exclusion. Les différentes contraintes sont dimensionnées de manière à ce que $\%_{res}$, $\%_{co}$ et $\%_{exc}$ soient respectés. La difficulté d'un problème est donnée en fonction de ces différents pourcentages.

Plusieurs classes de problèmes ont été définies en fonction des contraintes. Le tableau 3.2 décrit les paramètres utilisés pour chaque classe. En combinant ces paramètres, plusieurs catégories ont été définies. Une catégorie W-X-Y correspond à un problème avec une difficulté de classe W pour la mémoire, une classe X pour les contraintes de placement et de ressource et une classe Y pour l'ordonnançabilité.

| Mémoire | | Placement | | | | Ordonnançabilité | |
|---|---|---|---|---|---|---|---|
| | $\%_{mem}$ | | $\%_{res}$ | $\%_{co}$ | $\%_{exc}$ | | $\%_{global}$ |
| 1 | 60 | 1 | 0 | 0 | 0 | 1 | 40 |
| 2 | 30 | 2 | 15 | 15 | 15 | 2 | 60 |
| 3 | 10 | 3 | 33 | 33 | 33 | 3 | 90 |

TABLE 3.2 – Les classes de difficulté

| | **Benders** | | | | | | | **CP** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | %Suc | NbC | Time | Nodes | Iter | Noe | Cut | %Suc | NbC | Time(s) | Nodes |
| **1-1-3** | 99 | 99 | 5,6 | 3105,9 | 107,5 | 482,5 | NA | 99 | 99 | **3,8** | **1685,3** |
| **2-2-2** | 100 | 56 | **0,4** | **370,6** | 19,9 | 79,7 | NA | 100 | 56 | 5,3 | 1985,9 |
| **2-2-3** | 90 | 30 | **8,3** | 3267,8 | 30,1 | 154,2 | NA | 90 | 30 | 10,4 | **4059,8** |
| **2-3-2** | 100 | 19 | 0,4 | 172,6 | 6,8 | 37,3 | NA | 100 | 19 | **0,3** | **9,2** |
| **3-2-2** | 99 | 57 | 1,4 | 960,2 | 39,6 | 143,2 | NA | 99 | 57 | 1,4 | **458,1** |

TABLE 3.3 – Moyennes des résultats pour chaque approche.

Le tableau 3.3 présente les résultats obtenus avec Benders et avec la contrainte globale : %Suc donne le pourcentage d'instances résolues ; NbC le nombre d'instances consistantes ; Time le temps en seconde de résolution ; Nodes le nombre de nœuds explorés ; Iter le nombre d'itérations de la méthode de Benders ; Noe le nombre de contraintes de type *NotAllequal* et Comb le nombre de combinaisons linéaires correspondant aux coupes extraites de la méthode de Benders. Les valeurs moyennes de ces données sont présentées ici pour les instances résolues (une solution est trouvée ou l'inconsistance est prouvée) sur 100 instances par catégorie. Une limite de temps a été fixée à 10 minutes par instance. La stratégie de recherche utilisée est basée sur une approche par impacts Refalo (2004).

Le tableau 3.4 détaille le temps de résolution et donne les valeurs moyennes, médianes, minimales et maximales pour les instances résolues.

Les résultats montrent que l'approche basée sur la contrainte globale (97.8% d'instances résolues) est très compétitive par rapport à celle de Benders (97.3% d'instances résolues). Les valeurs minimales et maximales des temps de résolution montrent que Benders peut-être plus efficace pour les problèmes où l'ordonnancement est facile (classe 2-2-2 où le temps moyen avec Benders est nettement inférieur à celui avec la contrainte globale). Cela s'explique par le fait que la contrainte globale doit résoudre beaucoup d'équations d'ordonnançabilité qui ont peu d'impact sur l'élagage alors que Benders ne considère que les contraintes de placement et de ressource.

Pour les catégories de problèmes difficiles en ordonnancement (par exemple la classe 1-1-3), Benders est clairement moins rapide que la contrainte globale. En effet, pour Bendres les contraintes liées à l'ordonnançabilité sont entièrement apprises au cours de la résolution et donc peu efficace pour guider la recherche au début. La contrainte globale permet d'améliorer ce comportement.

Le résultat le plus significatif est qu'en pratique l'aspect pseudo-polynomial des équations

| | **Benders** | | | | | **CP** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | %Suc | Time(s) | | | | %Suc | Time(s) | | | |
| | | Av | Med | Min | Max | | Av | Med | Min | Max |
| **1-1-3** | 99 | 5,6 | 2,8 | **0,27** | **60,6** | 99 | **3,8** | **0,45** | 0,3 | 78,6 |
| **2-2-2** | 100 | **0,4** | **0,27** | **0,17** | **4,2** | 100 | 5,3 | 0,34 | 0,23 | 314,9 |
| **2-2-3** | 90 | **8,3** | **0,25** | 0,17 | **344** | 90 | 10,4 | 0,36 | **0** | 375 |
| **2-3-2** | 100 | 0,4 | **0,26** | 0,17 | 6,5 | 100 | **0,3** | 0,37 | **0** | **1,9** |
| **3-2-2** | 99 | 1,4 | **0,32** | **0,17** | **15** | 99 | 1,4 | 0,33 | 0,23 | 91,2 |

TABLE 3.4 – Valeurs Moyennes, médianes, minimums et maximums des temps de résolution.

d'ordonnançabilité n'est pas un facteur bloquant pour l'élagage. Ainsi, la contrainte globale est efficace pour ce genre de problème.

## 3.6 Exploitation des explications pour aider le concepteurs

En comparaison avec les autres méthodes de recherche, utiliser un solveur de contraintes permet d'expliquer les raisons d'un échec. En effet, quand le domaine d'une variable devient vide (aucune valeur n'existe pour cette variable qui respecte toutes les contraintes), le solveur de ppc notifie à l'utilisateur qu'il n'y a pas de solution. Cependant, en utilisant l'approche basée sur les explications pour les contraintes, il est possible d'expliquer les causes d'un échec (Jussien, 2003).

Dans le cas du problème d'allocation pour lequel aucune solution ne peut être trouvée, nous analysons l'ensemble des contraintes qui retournent une explication sur l'inconsistance du problème. Au niveau de la conception, il est nécessaire d'incriminer à un haut niveau les paramètres du systèmes qui sont la cause de l'inconsistence : contraintes d'allocation, ordonnancement, capacité des processeurs, etc. Dans cette étude nous ne nous focalisons que sur les aspects logiciels en fournissant au concepteur un indicateur sur l'implication de chaque tâche sur les raisons de l'échec. Pour cela un critère prenant en compte chacune des contraintes est évalué. Nous nous plaçons ici dans le contexte de l'approche par décomposition et donc utilisons les coupes apprises par le problème esclave. L'évaluation du critère est basé sur les remarques suivantes :

— plus une tâche apparaît dans les coupes dues à l'ordonnancement, plus la tâche a un impact sur l'ordonnançabilité,

— le niveau de propagation d'une coupe de type *NotAllEqual*($x_i$) est fortement lié à sa taille, c'est-à-dire que plus la contrainte est petite en terme de variables impliquées, plus son impact est grand.

Dans sa forme la plus restreinte (sans prendre en considération le réseau), le critère devient :

$$\mathcal{C}_i = \sum_{\substack{c \in \texttt{NAE} \\ x_i \in c}} \frac{1}{\#c}$$

avec NAE l'ensemble des contraintes de types *NotAllEqual*.

Ce critère représente la présence d'une tâche dans chacune des contraintes et son impact. Plus $\mathcal{C}_i$ est élevé, plus l'impact de la tâche $i$ est important pour expliquer l'inconsistance du problème. En étudiant les tâches ayant les valeurs les plus grandes, il est possible d'identifier rapidement les sources du problème et le concepteur peut changer en conséquence les contraintes du problème. La manière dont cette décision est prise dépend évidemment du processus de conception et des latitudes dont dispose le concepteur.

## 3.7 Bilan et analyse retrospective

Ce travail montre l'intérêt et l'efficacité d'utiliser la programmation par contraintes pour traiter des problèmes de configuration de systèmes temps réel. Que ce soit par la définition de contraintes globales dédiées pour l'ordonnancement ou par l'utilisation des mécanismes propres à la ppc, cette méthode apporte des solutions efficaces et flexibles. L'extension pour guider un concepteur en cas d'échec offre aussi des possibilités intéressantes.

Cette étude sur la modélisation des contraintes d'ordonnançabilité m'a permis de me familiariser avec la ppc et d'en comprendre l'efficacité ainsi que les limites. Plusieurs études ont été menées à travers des stages de master pour poursuivre dans cette voix, mais aucun résultats probants n'en a résulté.

Nous ne détaillerons pas dans ce document l'application directe de cette méthode au contexte multiprocesseur partitionnée car l'extension est immédiate et a été publiée dans (Hladik et Déplanche, 2009). La poursuite de ces travaux en prenant en considération des contraintes de type de bout-en-bout ou des coûts supplémentaires en cas de partage de données (voir l'étude réalisée par Mehiaoui *et al.* (2013)) a été envisagée et malheureusement abandonnée faute de ressources. Son couplage avec des méthodes d'analyse d'ordonnançabilité pour AUTOSAR (voir chapitre 5) serait aussi un travail intéressant.

Le point le plus prometteur reste l'exploitation des explications en cas d'échec pour guider le concepteur. Ce problème a aussi été rencontré lors de l'étude présentée dans le chapitre 4. Une étude plus approfondies pour intégrer ces explications dans un processus de développement serait une piste intéressante à approfondir. À ma connaissance, très peu de travaux s'occupent des moyens pour aider un concepteur à revoir ses choix en cas d'échec de la configuration.

CHAPITRE 4

# Optimisation d'une plate-forme IMA

Dans le domaine avionique, les architectures embarquées ont connu une mutation importante avec l'apparition des architectures modulaires intégrées (IMA). En offrant aux applications un support d'exécution et de communication standard et mutualisé, ces architectures ont permis une réduction du poids et de la complexité de l'architecture physique. Cette réduction de la complexité du bas niveau s'est cependant traduite par une difficulté accrue de conception et d'intégration des applications car il faut gérer le partage des ressources.

Les travaux présentés dans ce chapitre ont abordé cette problématique sous deux angles différents. Le premier concerne l'ordonnancement des fonctions avioniques en suivant une politique strictement périodique sans recouvrement des exécution. Le second aborde le dimensionnement du réseau avionique AFDX et plus particulièrement des liens virtuels. Ces deux problématiques ont été couvertes par la thèse d'Ahmad Al Sheikh dans le contexte du projet ANR SATRIMMAP et publiées dans (Al Sheikh *et al.*, 2009, 2010, 2011c,b, 2012, 2013). Les preuves des théorèmes et tous les résultats expérimentaux ne sont pas exposés dans ce manuscript. Le lecteur intéressé peut se référer principalement à (Al Sheikh *et al.*, 2012, 2013) pour de plus amples détails.

Les approches que nous avons explorées suivent le schéma de conception pour une plate-forme avionique présenté dans la figure 4.1. Cette approche est sous-optimale car les différentes étapes sont disjointes et ne considèrent pas la plate-forme dans son ensemble. Les retours suite à un échec sur l'une des étapes ne sont pas automatisés et seules les parties grisées ont été étudiées.

La partie 4.1 décrit l'étape d'allocation et d'ordonnancement des partitions sur les modules, c'est-à-dire l'ordonnancement de fenêtres temporelles d'exécution sur les machines d'exécution, en proposant diverses solutions pour optimiser l'utilisation de la plate-forme. La partie suivante s'attache au dimenssionnement des communications et à leur routage. Pour ces deux études des solutions diverses ont été proposées afin de trouver un compromis entre complexité du problème et performance des algorithmes.

Un travail complémentaire sur l'allocation a été réalisé par Pira et Artigues (2013) dans le cadre du projet OSEC. Une formalisation plus compacte et efficace du problème a ainsi été proposée. N'ayant pas participé à cette étude, les résultats ne seront pas présentés

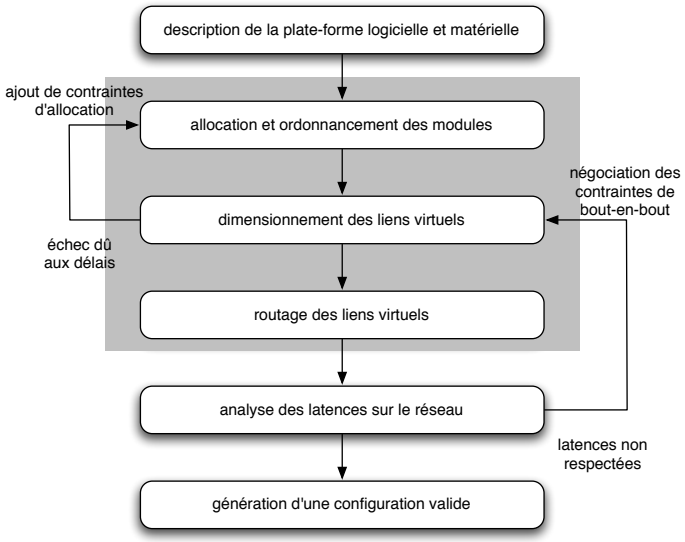



FIGURE 4.1 – Conception d'une plate-forme IMA par décomposition

dans ce manuscrit. Il peut-être cependant intéressant pour un lecteur voulant approfondir le sujet de consulter ce travail.

## 4.1 Ordonnancement d'une plate-forme IMA

La particularité de l'ordonnancement des fonctions avioniques sur une plate-forme IMA provient de son aspect strictement périodique. Ainsi, des fenêtres de temps, appelées partitions, doivent être réservées de façon non-préemptive à intervalles strictement constants sur un horizon de temps infini.

En grande majorité, les travaux sur l'allocation de partitions périodiques portent sur le cas où la préemption est permise (Leung et Merril, 1980; Labetoulle, 1974; Liu et Layland, 1973). En comparaison, le problème de l'ordonnancement périodique non-préemptif a reçu beaucoup moins d'attention (George *et al.*, 2006) et a essentiellement été considéré dans le cas particulier d'une périodicité lâche (Korst *et al.*, 1997). Un des premiers travaux sur l'ordonnancement non-préemptif de partitions strictement périodiques est celui de Jan Korst pour le traitement temps réel de signaux vidéo (Korst, 1992; Korst *et al.*, 1996). Depuis, plusieurs articles traitant de ce problème sont apparus (Meumeu Yomsi et Sorel, 2006; Kermia et Sorel, 2007; Marouf et Sorel, 2010; Eisenbrand *et al.*, 2010).

Les travaux présentés dans ce chapitre ont permis d'apporter une solution au problème d'ordonnancement de partitions sans considérer d'hypothèse restrictive sur les périodes comme cela a été fait dans (Eisenbrand *et al.*, 2010). Pour cela, une formulation sous la forme d'un programme linéaire en nombres entiers intégrant des contraintes temporelles et de ressource a été proposée (Al Sheikh *et al.*, 2010). Ensuite, afin de permettre le passage à l'échelle, une heuristique a été développée en s'inspirant de la théorie des jeux dans laquelle chaque partition adapte son ordonnancement pour maximiser sa propre fonction d'utilité, définie par une marge d'évolution (Al Sheikh *et al.*, 2011b). La convergence de cet algorithme vers un point d'équilibre pour lequel aucune partition n'a intérêt à modifier sa stratégie est montrée et l'existence d'au moins un point d'équilibre globalement optimal est établie. Les résultats numériques obtenus montrent que cet algorithme est beaucoup plus rapide que la méthode exacte et fournit une bonne approximation. Pour améliorer encore

la qualité des solutions obtenues, cette heuristique a été implémentée conjointement à un algorithme multi-start qui permet d'obtenir des garanties probabilistes sur l'optimalité des équilibres atteints.

### 4.1.1 Ordonnancement monoprocesseur

Le problème se modélise par un ensemble $\Pi = \{1, \ldots, N\}$ de $N$ partitions strictement périodiques, pour lesquelles nous cherchons un ordonnancement non-préemptif permettant de garantir qu'il n'y a aucun recouvrement temporel dans leurs exécutions. Chaque partition $i \in \Pi$ est caractérisée par sa période $p_i$ et par son budget de temps $e_i$, qui représente la durée d'exécution maximale accordée à la partititon. Posons $\mathcal{O}_i = \{0, 1, 2, \ldots, p_i - 1\}$. Nous notons $o_i \in \mathcal{O}_i$ la date de première exécution de la partition $i$, aussi appelée offset. Nous définissons $\mathbf{o} = [o_1, \ldots, o_N]$ comme étant le vecteur des offsets et $\mathcal{O} = \times_{i=1}^{N} \mathcal{O}_i$ comme l'ensemble des vecteurs d'offsets possibles. Les partitions étant strictement périodiques, pour un vecteur d'offsets $\mathbf{o}$, l'instance $k$ de la partition $i$ (ou $k$ème exécution) s'exécute dans l'intervalle

$$I_k^i(o_i) = [o_i + k\,p_i, o_i + k\,p_i + e_i]\,. \tag{4.1}$$

Le problème que nous considérons consiste à affecter une date de première exécution à chaque partition de telle manière qu'il n'y ait aucun recouvrement de leurs exécutions, c'est-à-dire à déterminer un vecteur d'offsets $\mathbf{o} \in \mathcal{O}$ tel que $I_k^i(o_i) \cap I_l^j(o_j) = \emptyset$ pour tous $k, l \in \mathbb{Z}$.

Afin de garantir qu'il n'y ait pas de recouvrement entre deux exécutions, nous définissons la distance minimale séparant les débuts d'exécution de différentes partitions soit :

**Lemme 4.1.**

$$\min_{k,l \in \mathbb{Z}} |(o_j + l p_j) - (o_i + k p_i)| = \min\left[(o_j - o_i)\%g_{i,j}, (o_i - o_j)\%g_{i,j}\right] \tag{4.2}$$

*avec $g_{i,j}$ le plus grand commun diviseur de $p_i$ et $p_j$ et avec le symbole % utilisé comme notation abrégée de l'opérateur modulo, c'est-à-dire que $a\%b$ doit être lu comme $a \mod b$.*

De ce résultat une condition nécessaire et suffisante pour l'ordonnançabilité de deux partitions a été décrite dans le théorème suivant :

**Théorème 4.1.** *(Korst, 1992) Les exécutions de deux partitions $i$ et $j$ ne se recouvrent pas si et seulement si $e_i \leq (o_j - o_i)\%g_{i,j}$ et $e_j \leq (o_i - o_j)\%g_{i,j}$, ou, de façon équivalente, si et seulement si $e_i \leq (o_j - o_i)\%g_{i,j} \leq g_{i,j} - e_j$.*

Ainsi, le terme $\frac{(o_j - o_i)\%g_{i,j}}{e_i}$ représente le facteur multiplicatif maximal par lequel la durée d'exécution $e_i$ de la partition $i$ peut être multipliée sans interférer avec les exécutions de la partition $j$. Il est possible de voir ce terme comme la marge admissible sur la durée $e_i$ pour que le système reste ordonnançable. Si ce terme est supérieur ou égal à 1 pour toutes partitions $i, j \neq i$, cela signifie que le vecteur d'offsets $\mathbf{o}$ est admissible.

Une condition suffisante pour qu'un ordonnancement défini par $\mathbf{o}$ soit admissible est donc que $\min_{i \neq j} \frac{(o_j - o_i)\%g_{i,j}}{e_i} \geq 1$. Ainsi, pour rechercher un ordonnancement admissible, il apparaît intéressant de chercher à maximiser ce minimum, soit en introduisant formellement cette marge comme étant

$$d_{ij}(\mathbf{o}) = \min\left(\frac{(o_j - o_i)\%g_{i,j}}{e_i}, \frac{(o_i - o_j)\%g_{i,j}}{e_j}\right), \tag{4.3}$$

le problème d'ordonnancement peut être formulé sous la forme :

$$\begin{aligned} &\text{maximiser } \min_{i,j\neq i} d_{ij}(\mathbf{o}), \\ &\text{sous la contrainte } \mathbf{o} \in \mathcal{O}. \end{aligned} \tag{OPT}$$

Remarquons que si la valeur optimale de ce problème est strictement supérieure à 1, cela permet de garantir une marge d'évolution sur les budgets de temps. Ainsi, tant que les durées des partitions restent inférieures à cette marge, l'ordonnancement est valide.

Le problème d'ordonnancement monoprocesseur pour des partitions strictement périodiques s'écrit alors sous la forme du programme linéaire en nombres entiers suivant :

$$\begin{array}{ll} \text{maximiser} & \alpha \\ \text{sous les contraintes} & \mathbf{o} \in \mathcal{O}, \\ & (o_j - o_i) - q_{j,i}\, g_{i,j} \geq \alpha\, e_i,\ \forall (i,j) \in \Pi^2, \\ & (o_j - o_i) - q_{j,i}\, g_{i,j} \leq g_{i,j} - \alpha\, e_j,\ \forall (i,j) \in \Pi^2, \\ & o_i \in [0, p_i),\ \forall i \in \Pi, \end{array}$$

où la variable $q_{j,i}$ représente le quotient entier $\left\lfloor \frac{o_j - o_i}{g_{i,j}} \right\rfloor$.

Bien que ce programme linéaire en nombres entiers puisse être résolu numériquement par des solveurs classiques, il n'est possible de le résoudre en pratique que sur des exemples de taille modeste – rappelons que le problème est NP complet au sens fort –, ce qui nous a conduit à en proposer des méthodes de résolution *ad hoc*.

### 4.1.2 Algorithme de la meilleure réponse dans le cas monoprocesseur

L'algorithme proposé pour résoudre le problème d'optimisation (OPT) est inspiré d'un algorithme issu de la théorie des jeux (Fudenberg et Tirole, 1991). Pour cet algorithme, nous identifions les partitions à des joueurs jouant un jeu séquentiel. Chaque partition adapte à son tour de jeu, sa stratégie via son offset, et cela en fonction des offsets des autres partitions. Le jeu se poursuit jusqu'à ce que le vecteur d'offsets converge vers un point d'équilibre, appelé équilibre de Nash, dans lequel plus aucune partition n'a interêt à changer sa stratégie.

Notons $o_j^n$ la stratégie, c'est-à-dire l'offset, de la partition $j$ au début de l'itération $n$ et supposons qu'à cette itération, c'est au tour de la partition $i$ de jouer. Son offset va être recalculé de manière à maximiser sa distance relative par rapport aux autres partitions en résolvant le problème suivant :

$$\begin{aligned} &\text{maximiser } \min_{j\neq i} d_{i,j}(x, \mathbf{o}_{-i}^n) \\ &\text{sous la contrainte } x \in \mathcal{O}_i, \end{aligned} \tag{SCHD-$i$}$$

où, suivant la notation habituelle en théorie des jeux, $\mathbf{o}_{-i} = [o_1, o_2, \ldots, o_{i-1}, o_{i+1}, \ldots, o_N]$ est le vecteur d'offsets de tous les joueurs autres que $i$.

Suite à cette itération, le nouvel offset de la partition $i$, $o_i^{n+1}$, prend la valeur de la solution optimale du problème SCHD-$i$, c'est-à-dire sa meilleure réponse $x$. Si cette dernière n'est pas unique, la partition retient le plus petit offset parmi ceux donnant la meilleure réponse.

Introduisons les notations

$$\begin{aligned} \alpha_i^n &= \min_{j\neq i} d_{i,j}(\mathbf{o}^n) \\ \mathcal{S}_i(\mathbf{o}_{-i}^n) &= argmax_{x\in\mathcal{O}_i} \min_{j\neq i} d_{i,j}(x, \mathbf{o}_{-i}^n), \end{aligned}$$

où $\alpha_i^n$ est l'utilité du joueur $i$ après l'itération $n$, et $\mathcal{S}_i(\mathbf{o}_{-i}^n)$ est l'ensemble des meilleures réponses de ce joueur.

L'algorithme 3 décrit le processus global de recherche de la meilleure solution. L'étape 4 de l'algorithme, $n\%N + 1$ donne l'index du joueur qui doit adapter sa stratégie à l'itération $n$. Ensuite, si le joueur $i$ ne peut améliorer son utilité $\alpha_i^n$, alors il ne change pas sa stratégie, c'est-à-dire que $o_i^{n+1} = o_i^n$. Cette hypothèse, bien que non restrictive, est utile pour démontrer la convergence de l'algorithme.

**Algorithme 3** Meilleure Réponse monoprocesseur

```
1: procédure BestResponse(o^0)
2:     n ← 0
3:     tant que o^n ≠ o^{n−N} faire
4:         pour i = 1 to N faire
5:             si i = n%N + 1 and max_x min_{j≠i} d_{i,j}(x, o^n_{−i}) > α^n_i alors
6:                 o_i^{n+1} ← min argmax(SCHD-i)
7:             sinon
8:                 o_i^{n+1} ← o_i^n
9:             fin si
10:        fin pour
11:        n ← n + 1
12:    fin tant que
13:    retourne o^n
14: fin procédure
```

Deux propriétés importantes de cet algorithme sont formulées dans les théorèmes suivants. Nous rappelons que les démonstrations ne sont pas données ici et qu'un lecteur intéressé peut consulter (Al Sheikh, 2011).

**Théorème 4.2.** *L'algorithme de la meilleure réponse converge vers un point d'équilibre.*

**Théorème 4.3.** *Il existe au moins un point d'équilibre qui est aussi une solution optimale du problème* (OPT).

En conséquence de ces deux théorèmes, si le point de départ est choisi de manière appropriée, l'algorithme de la meilleure réponse converge vers une solution globalement optimale.

Une borne supérieure sur le nombre d'itérations est aussi produite et est une estimation pessimiste qui est exponentielle en nombre de partitions. En pratique, pour toutes les expérimentations effectuées, l'algorithme converge en quelques dizaines d'itérations.

**Proposition 4.1.** *Soit* $\alpha_{max} = \max_i \min_{j\neq i} \frac{g_{i,j}}{e_i+e_j}$ *et* $\Delta = \min_{j,k} \frac{1}{ppcm(e_j,b_k)}$. *L'algorithme de la meilleure réponse converge en au plus* $\binom{N+K}{K}N$ *itérations, où* $K = \lceil \alpha_{max}\Delta^{-1} \rceil$.

Le point critique de cet algorithme est le calcul de la meilleure réponse d'un joueur. Il peut être calculé en effectuant une recherche linéaire, ce qui nécessite $O(p_i)$ opérations, mais il est en fait possible de déterminer la meilleure réponse de façon beaucoup plus efficace pour des offsets à valeurs réelles en utilisant le résultat suivant :

**Théorème 4.4.** *Pour des offsets de valeurs réelles, c'est-à-dire* $\mathcal{O}_i = \{0, 1, ..., p_i - 1\}$, *alors* $\mathcal{S}_i(\mathbf{o}_{-i}) \subset \mathcal{I}_i(\mathbf{o}_{-i}) \subset \mathcal{O}_i$, *avec*

$$\mathcal{I}_i(\mathbf{o}_{-i}) = \bigcup_{(j,k)\in(\Pi\setminus\{i\})^2} \left\{ x : \frac{(x - o_j)\%g_{i,j}}{e_j} = \frac{(t_k - x)\%g_{i,k}}{b_k} \right\}$$

*l'ensemble des points d'intersection associés à la partition* $i$.

Ainsi, la solution de (SCHD-$i$) est obtenue en restreignant la recherche aux points de l'ensemble $\mathcal{I}_i(\mathbf{o}_{-i})$. Dans la mesure où le problème original n'est défini que pour des offsets entiers, l'heuristique décrite dans l'Algorithme 3 va chercher la meilleure réponse de la partition $i$ dans les entiers directement inférieurs et supérieurs aux points de l'ensemble $\mathcal{I}_i(\mathbf{o}_{-i})$, plutot que d'examiner tous les $p_i$ points possibles. Il est de plus possible de génerer très efficacement les points de $\mathcal{I}_i(\mathbf{o}_{-i})$ grâce à la méthode décrite dans (Al Sheikh *et al.*, 2011b).

### 4.1.3 Ordonnancement multiprocesseur

Cette section propose d'étendre l'algorithme de la meilleure réponse pour l'ordonnancement monoprocesseur au cas multiprocesseur. Soit $\mathcal{P} = \{1, \ldots, P\}$ un ensemble de $P$ processeurs, le processeur $k$ étant caractérisé par sa capacité mémoire $\mu_k$ et par le nombre maximal $H_k$ de partitions qu'il peut accueillir. Un ordonnancement n'est plus seulement décrit par la donnée du vecteur d'offsets $\mathbf{o}$, mais également par l'affectation d'un processeur à chacune des partitions. Cette affectation peut être représentée par un vecteur de variables binaires $\mathbf{a} = (a_{i,k})_{i\in\Pi, k\in\mathcal{P}}$ telles que $a_{i,k} = 1$ si la partition $i$ est affectée au processeur $k$, et $a_{i,k} = 0$ sinon. Le problème d'ordonnancement multiprocesseur est alors formulé comme un programme linéaire en nombres entiers de la façon suivante (certaines contraintes spécifiques aux applications avioniques sont omises ici, voir (Al Sheikh *et al.*, 2010) pour plus de détails) :

$$\text{maximiser } \alpha(\mathbf{a}, \mathbf{o}) \tag{4.4}$$

$$\text{s.c.} \sum_{p_k\in\mathcal{P}} a_{i,k} = 1 \quad ,\forall i \in \Pi, \tag{4.5}$$

$$\sum_{i\in\Pi} a_{i,k}\, m_i \leq \mu_k \quad ,\forall k \in \mathcal{P}, \tag{4.6}$$

$$\sum_{i\in\Pi} a_{i,k} \leq H_k \quad ,\forall k \in \mathcal{P}, \tag{4.7}$$

$$(o_j - o_i) - q_{j,i}\, g_{i,j} \geq \alpha\, e_i - (2 - a_{i,k} - a_{j,k})\, Z \quad ,\forall k, \forall (i,j), \tag{4.8}$$

$$(o_j - o_i) - q_{j,i}\, g_{i,j} \leq g_{i,j} - \alpha\, e_j + (2 - a_{i,k} - a_{j,k})\, Z \quad ,\forall k, \forall (i,j), \tag{4.9}$$

$$a_{i,k} \in \{0, 1\} \quad ,\forall k, \forall i, \tag{4.10}$$

$$o_i \in [0, p_i) \quad ,\forall i \in \Pi, \tag{4.11}$$

$$q_{j,i} \in \mathbb{Z} \quad ,\forall (i,j), \tag{4.12}$$

Comme dans le cas monoprocesseur, la fonction objectif (4.4) représente le minimum des marges d'évolution des partitions. Les contraintes (4.5) imposent le choix d'un seul processeur par partition, tandis que les contraintes (4.6) et (4.7) sont celles associées aux capacités mémoire et en nombre de partitions des processeurs. Les contraintes (4.8) et (4.9) expriment la condition d'ordonnançabilité (voir théorème 4.1) pour les couples de

partitions affectées au même processeur ($Z$ est une grande constante qui permet de garantir que ces contraintes ne sont actives que si $a_{i,k} = a_{j,k} = 1$). Les contraintes (4.10), (4.11) et (4.12) décrivent le domaine des variables. Cette formulation linéaire n'est évidemment utilisable que pour des problèmes de taille modeste.

### 4.1.4 Algorithme de la meilleure réponse dans le cas multiprocesseur

L'algorithme de la meilleure réponse pour l'ordonnancement monoprocesseur s'étend au cas multiprocesseur de la façon suivante. À son tour, une partition $i$ va calculer sa meilleure réponse sur chacun des processeurs, les uns après les autres. Puis, elle va sélectionner le processeur et l'offset sur ce processeur lui permettant de maximiser son utilité qui est définie de la façon suivante

$$\alpha_i^n = \min_{\{j:j\neq i, a_{i,k}^n = a_{j,k}^n \forall k\}} d_{i,j}(\mathbf{o}^n), \tag{4.13}$$

et qui représente la marge d'évolution de la partition $i$ par rapport aux partitions ordonnancées sur le même processeur qu'elle et pour des vecteurs d'offsets $\mathbf{o}^n$ et d'allocation $\mathbf{a}^n$ donnés. Comme dans le cas monoprocesseur, on peut montrer que cet algorithme converge vers un point d'équilibre et qu'il existe au moins un point d'équilibre qui est globalement optimal.

### 4.1.5 Un exemple de résultat

Pour illustrer les résultats obtenus, considérons le problème d'ordonnancement monoprocesseur avec 20 partitions non harmoniques dont les caractéristiques temporelles sont indiquées dans le tableau 4.1. Pour cet exemple, l'algorithme de la meilleure réponse fournit une marge d'évolution minimale égale à $\alpha = 1.41$ en 2.83 secondes, tandis que la résolution avec ILOG CPLEX de la formulation linéaire en nombres entiers ne permet d'obtenir qu'une marge de $\alpha = 1.11$ sans avoir terminée au bout d'une heure de calcul. De manière générale, l'ensemble des expérimentations effectuées ont montré que l'algorithme de la meilleure réponse permet d'obtenir des solutions de bonne qualité et généralement admissibles avec des temps calculs très inférieurs à une résolution exacte basée sur la programmation linéaire en nombres entiers.

TABLE 4.1 – Exemple monoprocesseur avec 20 partitions non harmoniques (hyperpériode de 756000).

| partition | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Budget de temps | 10 | 30 | 30 | 10 | 10 | 10 | 10 | 10 | 30 |
| Période | 1200 | 1200 | 3600 | 1200 | 1200 | 1500 | 4200 | 1000 | 2000 |

| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 10 | 45 | 40 | 40 | 60 | 80 | 30 | 10 | 60 | 40 |
| 4000 | 1200 | 2400 | 2000 | 4000 | 3000 | 3000 | 2700 | 200 | 1800 | 1800 |

### 4.1.6 Algorithme multi-start

La qualité des solutions obtenues avec l'algorithme de la meilleure réponse peut être améliorée en utilisant une méthode multi-start (Martí, 2003). En effet, l'espace des solutions du

problème d'ordonnancement est divisé en région d'attraction, l'algorithme de la meilleure réponse conduisant vers le même équilibre pour tous les points initiaux appartenant à la même région. Une méthode multi-start permet de générer aléatoirement des points de départ de l'heuristique appartenant à des régions d'attraction différentes. Des règles Bayesiennes (Boender et Rinnooy Kan, 1987) sont utilisées pour arrêter l'exploration aléatoire lorsque des garanties probabilistes suffisantes sont obtenues sur l'optimalité de la meilleure solution trouvée.

Dans le cas monoprocesseur, la méthode multi-start s'exécute en quelques minutes pour des problèmes d'une quarantaine de partitions. Elle nous a permis de réduire l'écart relatif moyen à l'optimum à moins de 0.25%. Dans le cas multiprocesseur, nous avons obtenu un temps de calcul moyen de 16 minutes pour des problèmes avec environ 600 partitions et 48 modules. L'écart à l'optimum est passé en moyenne à moins 10% dans les cas général (et en dessous de 4% pour le cas harmonique). Nous avons observé que dans le cas multiprocesseur les points d'équilibre optimaux peuvent se trouver dans de petites régions d'attraction, ce qui peut nécessiter un critère d'arrêt plus strict que celui que nous avons utilisé.

## 4.2 Dimenssionnement optimal des liens virtuels dans un réseau AFDX

Le réseau déployé sur une plate-forme IMA est basé sur une architecture Avionic Full-Duplex Switched Ethernet (AFDX) qui assure la ségrégation des communications par la définition de liens virtuels ou VL (pour Virtual Link). Le routage d'un VL correspond en fait à un arbre multicast (voir figure 4.2) entre le module de traitement où est située la partition émettrice et les modules où sont situés les partitions réceptrices. Chaque VL est caractérisé par deux quantités : le BAG (Bandwidth Allocation Gap) qui définit l'intervalle de temps minimal séparant l'émission de deux trames consécutives, et la taille maximale des trames Ethernet envoyées, notée MFS (pour Maximum Frame Size). Ces deux paramètres permettent de limiter le taux de transmission d'un VL afin de réduire son interférence avec les autres VLs.

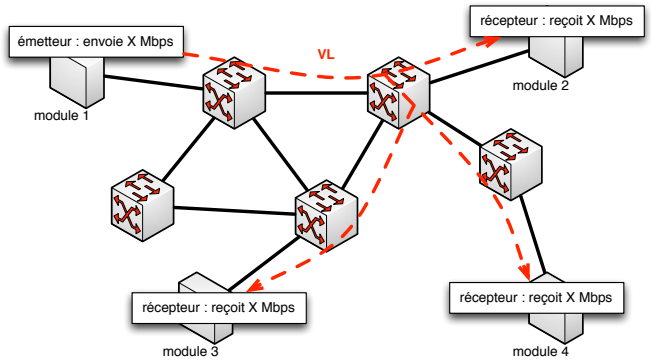



FIGURE 4.2 – Lien virtuel issu d'un module et acheminant des données vers trois autres.

Pour chaque message envoyé par un module, il est nécessaire de configurer le BAG et le MFS du VL qui lui est dédié tout en prenant en considération les exigences au niveau application à savoir la taille du message et sa latence maximale, c'est-à-dire le délai acceptable entre sa production et sa réception. Remarquons que pour garantir la latence d'un message, il faut estimer son temps de traversé (le temps pour une trame de taille maximale pour traverser le réseau) qui lui même dépend de la configuration des autres VLs et du

routage. En pratique, ce temps est très faible par rapport aux latences admissibles. Lauer *et al.* (2011), Charara (2007) et Scharbarg *et al.* (2009) estiment les temps de traversée inférieure à la milliseconde alors que les latences sont de l'ordre de la dizaine, voire centaine, de millisecondes. Ce qui nous a conduit à adopter une démarche pragmatique, à savoir borner de manière assez large le temps de traversée avec une valeur constante $\Delta$, puis vérifier que cette borne est satisfaite pour une configuration complète obtenue par les méthodes exposées ci-après.

Lors de la configuration des VLs, trois problèmes peuvent se poser :

— quelle valeur de BAG et MFS choisir pour minimiser la bande passante tout en respectant les exigences de latence ?

— est-il souhaitable de regrouper des messages dans un même VL si les messages ont le même émetteur et mêmes récepteurs ?

— comment définir les routes en garantissant que la charge sur chaque lien n'excède pas sa capacité et en garantissant une distribution des charges uniformes ?

Les travaux présentés dans la suite apportent des solutions pour ces trois questions. Dans un premier temps, nous montrons comment choisir les valeurs de BAG et MFS pour les VLs afin de minimiser leur bande passante en respectant les contraintes de latence. Nous considérons ensuite le cas où un émetteur envoie plusieurs messages à un groupe de récepteurs. Nous proposons une solution analytique au problème ainsi qu'un algorithme efficace pour agréger les messages afin de réduire la consommation de la bande passante. Pour terminer, nous formulons sous la forme d'un programme linéaire en nombres entiers le problème de routage afin de minimiser la capacité résiduelle des liens. Toutes ces approches permettent de réduire l'utilisation de la bande passante et ainsi de rendre plus flexible l'ajout de nouveau VLs ou de modifier ceux déjà existants.

### 4.2.1 Paramètres optimaux des liens virtuels

Nous considérons la transmission d'un unique message à partir d'une source pour un ensemble de destinations. Soit $s$ la taille du message en octet. Le message peut être fragmenté en $n$ trames, chacune ayant un entête de $c$ octets. La taille maximum de la charge utile d'une trame est de $f \in \mathbb{N}$ octets, et $f_{min} \leq f \leq f_{max}$. Pour un réseau AFDX, nous avons $c = 47$ octets, $f_{min} = 17$ octets et $f_{max} = 1471$ octets, ainsi la taille minimale d'une trame est de 64 octets et au maximum 1518 octes. La charge utile $f$ et le nombre de trames doit respecter $n\, f \geq s$ pour acheminer la totalité du message.

Le délai entre la transmission de deux trames consécutive est $BAG = 2^k$ ms, où $k \in \{0, 1, \ldots, 7\}$, ainsi le délai total entre la transmission de la première trame du message et sa dernière est $(n-1)\, BAG$ ms. Soit $\Delta$ la borne maximale sur le temps de traversée du réseau, alors le délai total de transmission de la première trame à partir de la source jusqu'à la réception de la dernière trame par le récepteur est borné par $(n-1)\, BAG + \Delta$. Nous faisons l'hypothèse que le délai total s'exprime sous la forme $\delta + \Delta$, ainsi les paramètres $n$ et $k$ doivent être tels que $(n-1)\, 2^k \leq \delta$.

Comme au moins $c+f$ octets sont transmis en $BAG$ secondes, la bande passante à réserver pour cette communication est $bw = (c+f)/BAG$. Le problème pour trouver les paramètres $n$, $f$ et $BAG$ tels que la bande passante soit minimum, tout en garantissant le respect de

la contrainte de latence, peut être formulé par

$$\begin{aligned} \text{minimiser } & bw = \frac{f+c}{2^k} \\ \text{sous les contraintes } & (n-1)\,2^k \leq \delta, \\ & n\,f \geq s, \\ & f_{min} \leq f \leq f_{max}, \\ & k \in \{0,1,\ldots,7\}, \\ & n, f \in \mathbb{N}. \end{aligned} \tag{BW}$$

Un premier résultat, a été de prouver que ce problème a une solution si et seulement si $\left\lceil \frac{s}{f_{max}} \right\rceil \leq 1+\delta$. Ensuite nous avons montrer qu'il est possible de déduire $f$ et $k$ si le nombre optimal de trames $n^*$ est connu.

Définissons $n_{min} = \left\lceil \frac{s}{f_{max}} \right\rceil$ et $n_{max} = 1+\delta$ et admettons que $n_{min} \leq n_{max}$ et que l'ensemble de toutes les solutions de $n$, $\Omega = \{n_{min}, n_{min}+1, \ldots, n_{max}\}$, n'est pas vide. Ce qui permet d'énoncer la proposition suivante sur les paramètres optimaux de transmission en fonction du nombre de trames $n$ :

**Proposition 4.1.** *Soit* $f(n) = \max(\lceil \frac{s}{n} \rceil, f_{min})$, $k(n) = \min(7, \left\lfloor \log_2 \left( \frac{\delta}{n-1} \right) \right\rfloor)$, *et* $bw(n) = (f(n)+c)\,2^{-k(n)}$ *pour* $n \in \Omega$. *Soit* $n^*$ *un minimum de* $bw(n)$ *dans* $\Omega$, *c'est-à-dire que* $bw(n) \geq bw(n^*)$ *pour tous les* $n \in \Omega$. *Alors,* $(n^*, f(n^*), k(n^*))$ *est une solution optimale du problème* (BW).

Suivant cette proposition, si $n$, le nombre de trames pour envoyer un message, est connu, alors la bande passante à réserver sera au moins $bw(n)$. Pour ce minimum, la taille de trame $f(n)$ et la valeur du BAG $2^{k(n)}$ se déduisent immédiatement. La proposition 4.1 implique donc que trouver la solution optimale du problème (BW) est équivalent à trouver le minimum de la fonction $bw(n)$ dans l'ensemble $\Omega$.

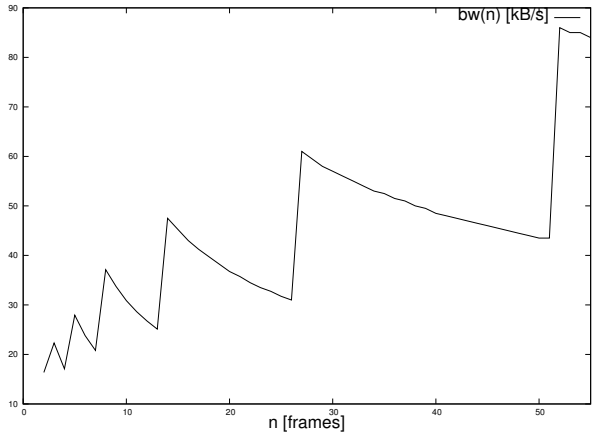



FIGURE 4.3 – Function $bw(n)$ pour $\delta$=100 ms et $s$=2000 bytes.

La figure 4.3 représente la fonction $bw(n)$ pour $s$=2000 octets avec $\delta$=100 ms. Nous constatons que cette fonction a des sauts positifs à certains points et qu'elle est décroissante entre ces points. La démonstration consiste à montrer que ces points appartiennent à la séquence $\{n_q\}_{q\in\mathbb{N}}$ définie par

$$n_q = 1 + \left\lfloor 2^{-q}\delta \right\rfloor \quad , q \in \mathbb{N}, \tag{4.14}$$

et qu'ils correspondent au nombre de trames pour lequel le BAG optimal est divisé par 2. Remarquons que $n_q \geq n_{q+1}$ pour tous $q \in \mathbb{N}$ et que, la fonction $bw(n)$ étant décroissante

dans les intervalles $(n_7, n_6], (n_6, n_5], \ldots, (n_1, n_0]$, le minimum de cette fonction est atteint à un point $n^*$ dans $\{n_0, n_1, \ldots, n_7\}$. En fait, nous avons montré que $bw(n)$ est minimum pour le minimum des valeur de la séquence $\{n_q\}_{q\in\mathbb{N}}$ dans $\Omega$, ce qui au final se traduit par le théorème suivant :

**Théorème 4.5.** *Si $s \leq f_{max}$, alors $n^* = n_7$, $f^* = f(n_7)$ et $k^* = 7$ est une stratégie optimale. Sinon, $k^* = \min\left(7, \left\lfloor \log_2\left(\frac{\delta}{n_{min}-1}\right)\right\rfloor\right)$, $n^* = 1 + \left\lfloor 2^{-k^*}\delta \right\rfloor$ et $f^* = f(n^*)$ est la stratégie optimale.*

Ce résultat montre que la stratégie optimale pour envoyer un message est toujours obtenue en envoyant $n_7$ trames et avec $k = 7$ quand la taille de message est plus petite ou égale à $f_{max}$. Sinon la stratégie optimale est obtenue en fixant $k = \min\left(7, \left\lfloor \log_2\left(\frac{\delta}{n_{min}-1}\right)\right\rfloor\right)$, avec $n_{min} = \left\lceil \frac{s}{f_{max}} \right\rceil$ et en fragmentant le message en $n_k$ trames de taille $f(n_k)$ qui seront transmises les unes après les autres tous les $2^k$ ms.

### 4.2.2 Stratégie optimale pour l'envoi de messages multiples

Nous considérons maintenant la situation où l'émetteur envoie plusieurs messages aux mêmes destinataires. Soit $\mathcal{M} = \{1, \ldots, M\}$ l'ensemble des messages envoyés par un émetteur. Pour le message $i \in \mathcal{M}$, $s_i$ représente sa taille en octets et $\delta_i$ le délai maximum en milliseconde entre le début de son envoi et sa réception.

L'envoie de plusieurs messages peut suivre plusieurs stratégies : envoyer chaque message séparément dans son propre VL, agréger toutes les données dans un seul message avec un unique VL, ou une stratégie mixte. Nous pouvons donc partitionner l'ensemble $\mathcal{M}$ en $K \in \{2, \ldots, M-1\}$ sous-ensembles $\mathcal{A}_1, \ldots, \mathcal{A}_K$ tels que $\mathcal{A}_i \cap \mathcal{A}_j = \emptyset$ pour $i \neq j$ et $\cup_i \mathcal{A}_i = \mathcal{M}$, et envoyer tous les messages appartenant à chaque sous-ensemble $\mathcal{A}_i$ dans un unique super-message de taille $s(\mathcal{A}_i) = \sum_{m\in\mathcal{A}_i} s_m$. Remarquons que dans ce cas une borne sur la latence du super-message $i$ est $\delta(\mathcal{A}_i) = \min_{m\in\mathcal{A}_i} \delta_m$.

Nous cherchons la partition qui minimise la bande passante. Pour chaque sous-ensemble $\mathcal{A} \subset \mathcal{M}$, notons $bw(\mathcal{A})$ la bande passante minimum pour envoyer un super-message de taille $s(\mathcal{A}) = \sum_{m\in\mathcal{A}} s_m$ avec une latence plus petite que $\delta(\mathcal{A}) = \min_{m\in\mathcal{A}} \delta_m$. Les paramètres optimaux $k(\mathcal{A})$, $n(\mathcal{A})$ et $f(\mathcal{A})$ pour le VL associé sont fournis par le théorème 4.5. Nous notons $\mathcal{P}(\mathcal{A})$ l'ensemble des partitions d'un sous-ensemble $\mathcal{A}$ de $\mathcal{M}$. Le problème pour trouver une partition $\{\mathcal{A}_i\}_{i=1,\ldots,K}$ de $\mathcal{M}$ qui minimise la bande passante totale $\sum_{i=1}^{K} bw(\mathcal{A}_i)$ se formule par

$$\begin{aligned} &\text{minimiser} \sum_{i=1}^{K} bw(\mathcal{A}_i) \\ &\text{sous les contraintes } \{\mathcal{A}_i\}_{i=1,\ldots,K} \in \mathcal{P}(\mathcal{M}) \\ &K \in \{1, \ldots, M\} \end{aligned} \tag{PART}$$

Une solution au problème PART est bien une partition de l'ensemble $\mathcal{M}$ comprenant tous les messages. Le nombre de solutions possibles est donné par la cardinalité de $\mathcal{P}(\mathcal{M})$ qui est le nombre de Bell d'ordre $M$ :

$$|\mathcal{P}(\mathcal{M})| = \frac{1}{e}\sum_{k=0}^{\infty} \frac{k^M}{k!}. \tag{4.15}$$

Il n'est donc pas raisonnable de trouver la solution optimale par une simple énumération si $M$ est de grande taille.

### 4.2.3 Cas particuliers

Dans un premier temps, nous proposons un ensemble de propriétés pour un ensemble de messages respectant l'hypothèse suivante.

**Hypothèse 4.1.** *Pour tous les messages* $m \in \mathcal{M}$, $f_{min} \leq s(m) \leq f_{max}$ *et* $\delta(m) \leq 2^7$.

Sous cette hypothèse, nous montrons que pour $\mathcal{A} \subset \mathcal{M}$ et $\{\mathcal{A}_i\}_{i=1,\dots,K}$ une partition de $\mathcal{A}$ telle que $s(\mathcal{A}_i) \leq f_{max}$ pour tous $i$, alors nous avons $k(\mathcal{A}_i) = 7$, $n(\mathcal{A}_i) = 1$ pour tous $i$, et $\sum_{i=1}^{K} bw(\mathcal{A}_i) = 2^{-7}\ (s(\mathcal{A}) + Kc)$.

Cette propriété est particulièrement intéressante car elle implique que la bande passante réservée dépend seulement du nombre de super-messages $K$ tant que leur taille ne dépasse pas $f_{max}$. Nous avons donc immédiatement

$$\frac{bw(\mathcal{A})}{\sum_{i=1}^{K} bw(\mathcal{A}_i)} = 2^{7-k(\mathcal{A})}\ \frac{f(\mathcal{A}) + c}{s(\mathcal{A}) + Kc}, \tag{4.16}$$

pour tous les sous-ensembles $\mathcal{A} \subset \mathcal{M}$ et n'importe quelle partition $\{\mathcal{A}_i\}_{i=1,\dots,K}$ de $\mathcal{A}$ telle que $s(\mathcal{A}_i) \leq f_{max}$ pour tous $i$.

Ce premier résultat montre que tant que $s(\mathcal{A}) \leq f_{max}$, la bande passante réservée est minimum en envoyant tous les messages de $\mathcal{A}$ dans un unique VL. En pratique, la plupart des messages envoyés sont de petite taille et avec des latences plus petite que 128 ms, ce qui permet d'affirmer grâce au résultat précédent que la solution optimale pour minimiser la bande passante est dans ce cas d'agréger tous les messages dans un unique super-message.

Mais, est-il toujours optimal d'agréger tous les messages ? La réponse est non et il est prouvé que pour deux messages respectant l'hypothèse 4.1, mais pour lesquels la somme des messages est plus grande que $f_{max}$, nous avons $bw(\mathcal{A}) \geq bw(\mathcal{A}_1) + bw(\mathcal{A}_2)$.

Ce résultat implique que l'agrégation de messages de petite taille dans un super-message n'est pas trivial si la taille de ce dernier dépasse $f_{max}$ et que dans le cas général, trouver une solution au problème PART n'est pas immédiat et nécessite la mise en place d'un algorithme dédié.

### 4.2.4 Algorithme *branch-and-bound*

Pour résoudre le problème PART, nous avons utilisé une approche de type *branch-and-bound*. L'algorithme 4 donne le pseudo-code mis en œuvre. Il est récursif et prend en entrée une solution partielle définie par l'ensemble $\mathcal{S}$ des super-messages existants et l'ensemble $\mathcal{N}$ des messages qui non pas encore été considérés (l'algorithme est appelé pour la première fois avec $\mathcal{S} = \emptyset$ et $\mathcal{N} = \mathcal{M}$). Pour une solution partielle, si $\mathcal{N} = \emptyset$, alors la solution complète $\mathcal{S}$ remplace la meilleure solution connue *sol* (ligne 14) si la bande passant est réduite. Si au contraire $\mathcal{N} \neq \emptyset$, alors une borne inférieure est utilisée pour élaguer les solutions de l'arbre (ligne 2). Sinon, l'algorithme considère le premier message $\mathcal{N}_1$ dans $\mathcal{N}$. Il explore alors récursivement tous les sous-arbres de la solution partielle obtenue en allouant $\mathcal{N}_1$ à l'un des super-messages existant (lignes 6 à 8) ou à un nouveau super-message (ligne 10 et 11).

L'algorithme est initialisé avec une borne supérieure calculée par l'algorithme glouton présenté dans la partie 4.2.5. Comme nous le verrons ci-après, cette heuristique fournit souvent une solution proche de l'optimum.

**Algorithme 4** Algorithme *branch-and-bound* pour la constructione des super-messages

```
1: procédure BandB(S : super-messages, N : messages restants)
2:     si N ≠ ∅ alors
3:         si lowerbound(N) + bw(S) ≥ bw(sol) alors
4:             retourne                                              ▷ Prune
5:         fin si
6:         pour i ← 1 to |S| faire
7:             C ← S                    ▷ C = ⋃C_j represente l'ensemble de super-messages
8:             C_i ← C_i ∪ {N_1}              ▷ Ajout de N_1 au super-message C_i existant
9:             BandB(C, N \ {N_1})
10:        fin pour
11:        S ← S ∪ {{N_1}}                                           ▷ Ajout du message seul
12:        BandB(S, N \ {N_1})
13:    sinon
14:        si bw(sol) > bw(S) alors
15:            sol ← S
16:        fin si
17:    fin si
18: fin procédure
```

L'évaluation de la borne inférieure se base sur l'inéquation

$$\sum_{i=1}^{K} bw(\mathcal{A}_i) > \sum_{m \in \mathcal{A}} \left\lceil \frac{s(m)}{n(m)} \right\rceil 2^{-k(m)} \qquad (4.17)$$
$$\forall \{\mathcal{A}_i\}_{i=1,\dots,K} \in \mathcal{P}(\mathcal{A}), \forall \mathcal{A} \subset \mathcal{M}$$

qui malheureusement n'est qu'une conjecture. Cependant, pour un ensemble de messages respectant l'hypothèse 4.1, nous avons montré qu'elle est vraie pour tous $\mathcal{A} \subset \mathcal{M}$, ce qui nous donne une certaine confiance dans son utilisation.

Un autre point important pour l'efficacité d'un algorithme *branch-and-bound* concerne l'ordre d'énumération des variables. Nous avons choisi de trier les messages suivant un ordre décroissant de taille. En pratique, nous avons observé que commencer par affecter les messages de taille importante conduisait à élaguer plus rapidement les branches. Intuitivement cela permet de séparer les gros messages dans des super-messages pour ensuite y ajouter les petits messages afin d'arriver au plus proche de $f_{max}$.

### 4.2.5 Algorithme glouton

Afin d'initialiser le *branch-and-bound*, nous avons proposé un algorithme glouton (voir algorithme 5) similaire à un algorithme *best-fit* utilisé dans les problèmes de bin-packing. Il considère les messages triés par ordre de taille décroissante (ligne 2) et les assigne dans un super-message existant (ligne 5 à 9) ou crée un nouveau super-message (ligne 9) dans le but de minimiser la bande passante de la solution partielle courrante (ligne 12).

Cet algorithme glouton est réalisé en $O(M^2)$ et est donc négligeable pour des instances de quelques dizaines de messages, ce qui est le cas pour des applications réelles.

**Algorithme 5** Algorithme glouton pour la construction des super-messages

**procédure** GREEDY($\mathcal{M}$ : messages)
  $\mathcal{S} \leftarrow \emptyset$ ▷ Les super-messages d'une solution partielle
  tri de $\mathcal{M}$ ▷ Le critère de tri choisi est par ordre de taille décroissante
  **pour** $\forall m \in \mathcal{M}$ **faire**
    $Q \leftarrow \emptyset$
    **pour** $i \leftarrow 1$ **to** $|\mathcal{S}|$ **faire**
      $\mathcal{C} \leftarrow \mathcal{S}$ ▷ $\mathcal{C} = \bigcup \mathcal{C}_j$ représente l'ensemble des super-messages
      $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{m\}$ ▷ Ajoute $m$ au super-message existant $\mathcal{C}_i$
      $Q \leftarrow Q \cup \{\mathcal{C}\}$
    **fin pour**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{\{m\}\}$ ▷ Message alone
    $Q \leftarrow Q \cup \{\mathcal{S}\}$
    $\mathcal{S} \leftarrow \text{argmin}_{Q_x \in Q}(bw(Q_x))$ ▷ Meilleure solution partielle
  **fin pour**
  **retourne** $\mathcal{S}$
**fin procédure**

### 4.2.6 Routage

Une fois les VL définis et paramètrés, il est nécessaire de spécifier leur routage dans le réseau de commutation AFDX en remplissant des tables de configuration. Ce processus de routage doit évidemment prendre en compte les ressources réseaux disponibles.

Le problème revient à déterminer un arbre multicast par VL. Dans le cas où il y a un seul VL, ce problème est équivalent à celui de l'arbre de Steiner (Gilbert et Pollak, 1968; Hwang *et al.*, 1992) qui est connu pour être NP difficile. Un autre cas particulier est celui où chaque VL a une seule destination, ce qui conduit à un problème de monoroutage qui est également NP difficile (Pióro et Medhi, 2004). Dans le cas général, le problème a été relativement peu traité, les travaux les plus proches concernant la détermination d'un arbre de routage multicast dans un réseau déjà chargé (Seok *et al.*, 2002).

Nous représentons le réseau AFDX par un graphe $(\mathcal{N}, \mathcal{E})$ ayant un ensemble $\mathcal{N} = \{1, \ldots, N\}$ de noeuds (modules de traitement inclus) et un ensemble $\mathcal{E}$ de liens. On note $c_e$ la capacité du lien $e \in \mathcal{E}$. Ce réseau doit acheminer un ensemble $\mathcal{V} = \{1, \ldots, V\}$ de VL, chaque VL $v$ étant caractérisé par :

— un noeud source (module de traitement) $src(v) \in \mathcal{N}$,

— un ensemble de noeuds destination (modules de traitement) $dest(v) \subseteq \mathcal{N} \setminus \{src(v)\}$,

— et une bande-passante $bw_v$.

L'objectif est de minimiser le taux d'utilisation maximal des liens du réseau, c'est-à-dire

$$\text{Minimiser } \left[\rho = \max_{e \in \mathcal{E}} \left(\frac{y_e}{c_e}\right)\right],$$

où $y_e$ représente le trafic total sur le lien $e$. Ce critère d'optimisation a été choisi suite à des études ayant montré que le délai réseau n'intervient que pour une faible part dans le délai de transmission au pire d'un message, le choix du BAG et les périodes des partitions réceptrices étant les paramètres dominants (Lauer *et al.*, 2011). Ce critère a de plus l'avantage de garantir une certaine marge d'évolution sur les ressources du réseau, qui peut être nécessaire si de nouveaux VL doivent être déployés.

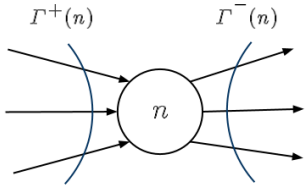



FIGURE 4.4 – Liens entrants et sortants du noeud $n$.

Notons $\Gamma^+(n)$ et $\Gamma^-(n)$ l'ensemble des liens entrants et sortants du noeud $n$ respectivement, comme illustré sur la Figure 4.4.

En adoptant une formulation noeud-lien, le problème peut être écrit comme un programme linéaire en nombres entiers :

$$\text{minimiser } \rho \tag{ROU}$$

$$\text{sous les contraintes } \sum_{e\in\Gamma^+(n)} x_v^e - \sum_{e\in\Gamma^-(n)} x_v^e = h_{n,v}, \forall n, v, \tag{4.18}$$

$$y_v^e\, M \geq x_v^e, \forall v, e, \tag{4.19}$$

$$y_v^e \leq x_v^e, \forall v, e, \tag{4.20}$$

$$\sum_{e\in\Gamma^+(n)} y_v^e \leq 1, \forall n, v, \tag{4.21}$$

$$y_e = \sum_{v\in\mathcal{V}} y_v^e, \forall e, \tag{4.22}$$

$$y_e \leq c_e\, \rho, \forall e, \tag{4.23}$$

$$y_v^e \in \{0,1\}, \forall v, e, \tag{4.24}$$

$$x_v^e \in \{0,\ldots,K_v\}, \forall v, e, \tag{4.25}$$

Dans cette formulation, la variable entière $x_v^e$ correspond au nombre de destinations du VL $v$ qui seront jointes en passant par le lien $e$. Les constantes $h_{n,v}$ intervenant dans les contraintes de conservation (4.18) sont données par

$$h_{n,v} = \begin{cases} -K_v & \text{si } i = src(v), \\ 1 & \text{si } i \in dst(v), \\ 0 & \text{sinon,} \end{cases}$$

où $K_v$ est le nombre de destinations du VL $v$. La variable binaire $y_v^e$ est calculée à partir de $x_v^e$ grâce aux contraintes (4.19) et (4.20). Elle indique si le lien $e$ est utilisé par le VL $v$. Les contraintes (4.21) imposent un routage en arbre pour les VL. La variable $y_e$ dont la valeur est définie par la contrainte (4.22) correspond au trafic sur le lien $e$. Elle est utilisée pour calculer la charge maximale $\rho$ des liens dans la contrainte (4.23). Cette formulation permet de résoudre de manière exacte des problèmes de taille assez conséquente dans des temps raisonnables. Elle n'apporte toutefois aucune garantie sur la longueur des arbres multicast générés.

### 4.2.7 Quelques résultats

Les expérimentations conduites sur les algorithmes d'agrégation des messages ont montré que l'algorithme de *branch-and-bound* arrive à produire des solutions rapidement (moins

d'une seconde pour une instance de 30 messages) en grande partie grâce à l'algorithme glouton qui permet de l'initialiser. En effet, cet algorithme produit des solutions proche de l'optimum (au pire de 2.6% sur l'ensemble des expérimentations conduites) avec des temps de l'ordre de la dizaine de milli-secondes pour 30 messages. Comparé avec des solutions naïves qui consistent à créer un VL pour chaque message, le gain sur la bande passante est de l'ordre de 17%. Pour un approche moins pessimiste qui consiste à regrouper tous les messages dans un seul VL, le gain est plutôt de l'ordre de 10%.

En ce qui concerne le routage, l'utilisation du solveur CPLEX a montré de très bonne performance. Sur des instances comportant 1000 VL, 7 switches AFDX et 6 modules, une solution a toujours été trouvée en moins de 10 s. Cette expérimentation a été confortée sur un exemple industriel provenant du projet ANR SATRIMMAP.

## 4.3 Bilan et analyse retrospective

Ce travail a permis d'appliquer des méthodes d'ordonnancement dans un cadre industriel en respectant les contraintes d'un standard, ici ARINC. L'étude sur l'optimisation de l'ordonnancement sur une plate-forme avionique IMA, nous a conduit à formaliser le problème et à le traiter avec des heuristiques originales basées sur la théorie des jeux. Nous avons aussi montré qu'il est possible d'optimiser l'utilisation de la bande passante d'un réseau AFDX en dimensionnant les liens virtuels et en les routant efficacement.

Cette étude est révélatrice pour la communauté temps réel du gain apportée par une ouverture vers d'autres domaines, ici l'aide à la décision et la théorie des jeux. Cela est d'autant plus vrai que l'étude réalisée par Pira et Artigues (2013) montre les gains calculatoires obtenus en laissant des spécialistes de la recherche opérationnelle traiter en profondeur le problème.

Ce travail est a permis d'apporter une solution viable à la problématique de configuration d'une plate-forme d'exécution et du réseau dans le cadre de l'avionic. Cependant, afin de couvrir l'ensemble du processus décrit par la figure 4.1, il manque une formalisation des retours en cas d'échec pour pouvoir relaxer le problème. Cette formalisation passerait par l'identification des paramètres contraignant la solution et le moyen d'introduire des contraintes pertinentes. Cette étude serait très similaire à celle menée dans le chapitre précédent et la ppc avec explications pourrait être un solution.

CHAPITRE 5

# Analyse d'ordonnançabilité pour AUTOSAR

Le consortium AUTOSAR (AUTOmotive Open System ARchitecture) est un acteur majeur dans l'industrie automobile. Il propose un standard pour les architectures logicielles et matérielles embarquées automobiles afin de garantir les propriétés de sûreté lors de la conception et de l'implémentation.

Le travail présenté dans cette partie étudie le standard AUTOSAR du point de vue des théories de l'ordonnancement temps réel. Ce travail a été mené en collaboration avec Sébastien Faucou, Yvon Trinquet et Anne-Marie Déplanche et a donné lieu à plusieurs publications (Hladik *et al.*, 2007a,b; Faucou *et al.*, 2008; Hladik *et al.*, 2009b). Il ne concerne que des architectures monoprocesseurs. Le cas multiprocesseur a été abordé dans le cadre du projet ANR RESPECTED et est présenté dans le chapitre 7.

La partie 5.1 présente rapidement AUTOSAR OS en ne se focalisant que sur les aspects dynamiques de l'ordonnancement. La partie suivante concerne l'analyse d'ordonnançabilité proprement dire et les résultats obtenus. Dans les travaux (Hladik *et al.*, 2007a; Faucou *et al.*, 2008) d'autres problématiques ont été abordées qui ne seront pas présentées ici. En particulier, un travail a été fait sur l'identification des services de surveillance des budgets temporels des tâches et leur impact sur l'ordonnançabilité. Nous en avions conclu que des ambiguïtés présentes dans le standard ne permettaient pas de produire une analyse exacte. Depuis, le standard a été revu, levant ces ambiguïtés.

## 5.1 Présentation de la spécification AUTOSAR OS

Les éléments décrits par la suite se basent sur les documents AUTOSAR Operating System version 2.0.1 (`https://www.autosar.org`) et OSEK/VDX operating system version 2.2.2 (`http://www.osek-vdx.org`). Des versions plus récentes des standards ont été publiées depuis le travail réalisé, mais les concepts liés à l'ordonnancement présentés ci-après n'ont pas changés.

Une des particularité d'AUTOSAR OS est que l'ensemble des applications sont définies statiquement, c'est-à-dire que tous les objets actifs (tâches, mutex, etc.) sont configurés hors-ligne, l'espace mémoire étant alloué au démarrage du système. Dans la suite, nous ne présentons que les mécanismes liés à l'ordonnancement.

### 5.1.1 Gestion des tâches

Les mécanisme de gestion des tâches d'AUTOSAR OS sont issus d'OSEK/VDX et offrent les mêmes services. Deux types de tâches sont distingués :

— les tâches dites basiques ont un code séquentiel et ne font pas d'appel système bloquant, c'est-à-dire qu'il n'y a pas d'attente d'un événement de synchronisation dans leur flot d'exécution. Leurs seuls point de synchronisation sont au début (activation) et à la fin de leur exécution (terminaison). Il est possible de mémoriser des demandes d'activation même si la tâche est déjà active. Chaque activation de tâches est mémorisée dans la file des tâches prêtes et l'ordre d'exécution est déterminé par une priorité fixe et un second critère de type FIFO en cas d'égalité.

— les tâches dites étendues sont composées d'un ou plusieurs blocs séparés par des invocations à des services de synchronisation, ce qui peut les conduire dans un état d'attente. La synchronisation est basée sur un mécanisme d'événement privé, c'est-à-dire que seule la tâche possédant l'événement peut explicitement se mettre en attente sur cet événement. Les occurrences des événements sont produites aussi bien par des tâches basiques ou étendues que par des OsIsr (voir ci-après). Pour une tâche étendue, il est aussi possible de mémoriser plusieurs demande d'activation pendant son exécution et la politique d'ordonnancement est à priorités fixes avec FIFO pour départager les égalités.

### 5.1.2 Gestion des Interrupt Service Routine

Le contrôle de l'exécution des traitements liés aux interruptions dans le système est réalisé à l'aide des OsIsr (*Interrupt Service Routine*). Les OsIsr sont invoqués suite à des événements externes provenant d'interruptions matérielles. Chaque OsIsr a une priorité statique qui est considérée comme plus élevée que toutes celles des tâches. AUTOSAR OS en distingue deux types : la catégorie 1 ne permet pas d'appel aux services de l'OS dans le code du traitement de l'interruption, alors que la catégorie 2 (notée OsIsr2) l'autorise.

AUTOSAR ne spécifiant pas l'implémentation à suivre, un choix de conception pour programmer un système AUTOSAR peut être de considérer les OsIsr2 comme des tâches avec des niveaux de priorités plus élevés. C'est le choix qui a été fait pour *Trampoline* (Bechennec *et al.*, 2006) (`http://trampoline.rts-software.org`), un projet open-source universitaire qui fournit l'implémentation d'un noyau compatible AUTOSAR OS.

Comme nous le verrons dans la suite, du point de vue de l'analyse d'ordonnançabilité, aucune différence n'est faite entre une tâche et une OsIsr2.

### 5.1.3 Politique d'ordonnancement

Chaque tâche a une priorité statique qui lui est assignée lors de la configuration. Une politique d'ordonnancement de type priorités fixes (appelée *Highest Priority First*) est utilisée avec pour second critère une politique FIFO pour les tâches ayant le même niveau de priorité.

Pour une application, l'ordonnancement peut être : entièrement non préemptif, entièrement préemptif, ou préemptif mixte. Dans le dernier cas, chaque tâche a son propre mode, préemptif ou non (spécifié de manière statique lors de la configuration).

Une notion de groupe de priorités existe aussi si des tâches partagent une ressource interne. Dans ce cas, cette ressource interne est automatiquement bloquée lorsqu'une tâche du groupe commence son exécution et la relâche quand elle se termine. Les tâches appartenant à un même groupe sont ainsi en exclusion pendant leur exécution. Par contre, les règles habituelles de préemption sont toujours actives pour les tâches qui n'appartiennent pas au groupe.

### 5.1.4 Gestion du partage de ressources

L'accès concurrent à des ressources partagées suit le protocole OSEK-PCP (*Priority Ceiling Protocol*) qui est une simplification du protocole PCP proposé par Sha *et al.* (1990) et étendu par Liu (2000). Il est aussi connu sous le terme *Immediate Priority Ceiling Protocol* (IPCP). Chaque ressource est associée à une priorité strictement supérieure à toutes les tâches ayant accès à cette ressource. Quand une tâche prend la ressource, sa priorité est immédiatement passée au niveau de priorité de la ressource, assurant ainsi que les autres tâches qui partagent cette ressource ne pouront pas y accéder. Ce protocole évite l'interblocage et l'inversion de priorité.

### 5.1.5 Alarme, compteurs et tables d'ordonnancement

AUTOSAR OS utilise les services d'alarme et de compteur d'OSEK/VDX et ajoute un nouveau concept, celui de table d'ordonnancement (*schedule table*).

Les alarmes et les compteurs permettent de produire des événements récurrents. Un compteur est un objet permettant de compter des événements à partir d'une source périodique telle qu'un timer matériel, ou asynchrone comme une interruption externe. Une alarme est associée à un compteur et expire quand ce compteur atteint un seuil provoquant une action prédéfinie telle que l'activation d'une tâche ou l'émission d'un événement de synchronisation. Une alarme peut être configurée comme étant cyclique ou fugace et avoir une référence temporelle soit absolue (par rapport au démarrage de l'application), soit relative à la valeur d'un compteur. Une alarme peut aussi être activée ou désactivée pendant l'exécution par l'appel de services dédiés.

Les tables d'ordonnancement sont une extension du concept d'alarme. Elles sont aussi liées à un compteur et définissent un ensemble de points d'expiration qui correspondent à des valeurs du compteur. Quand un point d'expiration est atteint, une ou plusieurs actions sont réalisées (activation de tâches ou production d'un événement). Une table d'ordonnancement fournit ainsi un mécanisme de synchronisation temporel (ou événementiel) entre les tâches.

## 5.2 Modélisation d'une application AUTOSAR OS

Dans la suite, le terme « tâche » est employé dans le sens commun des études d'ordonnancement et désigne les exécutions des tâches et des OsIsr2.

Un système de tâches AUTOSAR est modélisé par un triplet $\langle \mathcal{T}, \Lambda, src \rangle$ avec l'ensemble $\mathcal{T} = \{1..N\}$ composé de $N$ tâches, l'ensemble $\Lambda = \{1..M\}$ formé de $M$ sources d'activations et une relation $src : \mathcal{T} \to \Lambda$ associant à chaque tâche de $\mathcal{T}$ une source de $\Lambda$.

La tâche $i$ est définie par un tuple $\langle \pi_i, o_i, d_i, E_i \rangle$ avec : $\pi_i \in \mathbb{N}$ sa priorité ; $o_i \in \mathbb{N}$ l'offset de la tâche par rapport à l'occurrence de sa source d'activation ; $d_i \in \mathbb{N}$ l'échéance relative de la tâche par rapport à sa date d'activation ; $E_i = \{b_{ik} = \langle e_{i,k}, \gamma_{i,k} \rangle\}_{1 \leq k \leq L_i}$ l'ensemble des blocs d'exécution qui composent la tâche $i$.

Chaque bloc $b_{i,k}$ est caractérisé par son pire temps d'exécution $e_{i,k} \in \mathbb{N}$ et un seuil de préemption $\gamma_{i,k} \in [\pi_i, P]$ avec $P$ la plus grande priorité du système. Le premier bloc de $E_i$ est associé à une exécution complète de la tâche, nous avons donc $e_{i,0}$ qui représente le pire temps d'exécution de la tâche et $\gamma_{i,0}$ la priorité de la tâche ou la priorité plafond d'une ressource interne si la tâche fait partie d'un groupe. Chaque autre bloc modélise une section critique, c'est-à-dire une section de code délimitée par la prise et la libération d'une ressource. Remarquons que ces blocs sont soit imbriqués, soit séquentiels, mais ne peuvent pas s'entrelacer.

Une source d'activation $k$ est un couple $\langle p_k, \omega_k \rangle$ pour lequel : $p_k \in \mathbb{N}$ est le temps minimal d'inter-arrivée entre deux occurences de la source ; $\omega_k \in \mathbb{N} \cup \{\perp\}$ est l'offset de la première occurrence de la source par rapport à la date de démarrage du système (le terme $\perp$ est utilisé si cette date n'est pas connue).

Une tâche $i$ est dite associée à une source d'activation $k$ si $src(i) = k$. Cela signifie que la tâche $i$ est activée $o_i$ unités de temps après chaque occurrence de la source $k$. Dans la suite, nous noterons $\theta_k = \{i \in \mathcal{T} \,|\, src(i) = k\}$ l'ensemble des tâches qui sont attachées à la même source d'activation.

Ce modèle permet de capturer le comportement temporel d'une application AUTOSAR. Les tâches AUTOSAR et les OsIsr2 sont modélisées par les tâches de $\mathcal{T}$. Leurs conditions d'activation sont capturées par les sources d'activation de $\Lambda$. Le partage des ressources, la notion de groupe, ainsi que le caractère préemptif ou non d'une tâche, sont modélisés par les blocs et leur seuil de préemption. Ainsi, une tâche préemptive a une valeur $\gamma_{i,0}$ égale à $\pi_i$ ou égale à la priorité de son groupe, alors qu'une tâche non préemptive aura une valeur égale à $P$.

## 5.3 Analyse d'ordonnançabilité pour AUTOSAR OS

L'analyse d'ordonnançabilité proposée suit la démarche initiée par Joseph et Pandya (1986) et étendue par Audsley (1990). Cette approche consiste à calculer une borne supérieure sur le temps de réponse de chaque tâche pendant une période d'activité critique et à vérifier que cette valeur reste inférieure à leur échéance. L'intérêt majeure de cette technique réside dans la facilité d'intégration des différentes hypothèses faites sur le modèle. L'analyse que nous proposons s'appuie sur les travaux de : Lehoczky (1990) pour étendre les calculs aux échéances plus grandes que les périodes ; Sha *et al.* (1990) et Baker (1990) pour le partage de ressource ; Tindell (1994) et Palencia et González Harbour (1999) pour les tables d'ordonnancement (notion similaire aux transactions de tâches) ; Wang et Saksena (1999) pour les seuils de préemption ; Bimbard et George (2006) pour l'utilisation d'une politique FIFO comme second critère d'ordonnancement.

### 5.3.1 Définitions

La notion de période d'activité de niveau $\pi$ introduite par Lehoczky (1990) sert de base à l'ensemble de l'analyse proposée. Elle est définie comme étant un intervalle de temps

maximum pendant lequel une unité de traitement exécute de manière continue des travaux qui ont un seuil de préemption inférieur ou égal à $\pi$.

L'analyse proposée consiste à explorer l'ensemble des périodes d'activité que peut générer un système afin de trouver une borne sur les temps de réponse des tâches. Pour cela, nous définissons les moments d'activation des sources de $\Lambda$ par rapport au début de la période d'activité (Tindell, 1994; Palencia et González Harbour, 1999) : un vecteur de phase $\mathbf{\Phi}$, associé à une période d'activité de niveau $\pi$ commençant à la date $t_0$, est un ensemble de valeurs $\langle \phi_1, \ldots, \phi_m \rangle$ telles que $\phi_k$ est la date d'occurrence de la source d'activation $k$ survenant au plus tôt à ou après $t_0$.

Partant de cette définition, il est possible de calculer la phase d'activation de la tâche $i$ associée à la source $k$ par rapport à une période d'activité $\langle t_0, \mathbf{\Phi} \rangle$, c'est-à-dire calculer le délai entre $t_0$ et la première activation de la tâche $i$ dans la période d'activité. Ce délai est donné par

$$\varphi_i(\phi_k) : (0, p_k] \to [0, p_k) = (o_i - \phi_k) \mod p_k. \tag{5.1}$$

Nous avons donc la date de la $q$ième activation de la tâche $i$ associée à la source d'activation $k$ qui est égale à

$$a_{i,q}(\phi_k) = \varphi_i(\phi_k) + (q-1)p_k. \tag{5.2}$$

Remarquons que $t_0$ et $\mathbf{\Phi}$ définissent complètement les séquences d'activation des tâches pendant une période d'activité de niveau $\pi$. En considérant $t_0$ égal à 0, seule le vecteur $\mathbf{\Phi}$ sera utilisée dans la suite pour caractériser entièrement une période d'activité.

Par la suite, nous aurons besoin de manipuler des ensembles de tâches en fonction de leur priorité par rapport à celle de la tâche $i$, soit :

- les tâches plus prioritaires : $hp(i) = \{j | j \in \mathcal{T}, \pi_j > \pi_i\}$,
- les tâches avec un seuil de préeemption plus élevé : $ht(i) = \{j | j \in \mathcal{T}, \pi_j > \gamma_{i,0}\}$,
- les tâches ayant la même priorité : $sp(i) = \{j | j \in \mathcal{T}, \pi_j = \pi_i \wedge i \neq j\}$,
- l'ensemble des tâches plus prioritaires et de même priorité (incluant la tâche $i$) : $hsp(i) = hp(i) \cup sp(i) \cup \{i\}$.

### 5.3.2 Temps de réponse maximal d'une tâche dans une période d'activité

Avant de proposer une formulation du temps de réponse maximal d'une tâche dans une période d'activité, nous définissons deux fonctions permettant de borner la charge processeur sur un intervalle de temps. Nous ne présentons ici que les principaux résultats permettant de comprendre la démarche, les démonstrations ne sont pas exposées et sont disponibles dans les travaux publiés (Hladik *et al.*, 2007a,b; Faucou *et al.*, 2008; Hladik *et al.*, 2009b).

Pour la tâche $i$ associée à la source d'activation $k$, la fonction de charge fermée à droite est définie par :

$$\underline{w}_i(\phi_k, t) : (0, p_k] \times \mathbb{R} \to \mathbb{R} = \left(1 + \left\lfloor \frac{t - \varphi_i(\phi_k)}{p_k} \right\rfloor\right) e_{i0} \tag{5.3}$$

et celle ouverte à droite par :

$$\overline{w}_i(\phi_k, t) : (0, p_k] \times \mathbb{R} \to \mathbb{R} = \left(\left\lceil \frac{t - \varphi_i(\phi_k)}{p_k} \right\rceil\right) e_{i0} \tag{5.4}$$

La distinction entre les deux fonctions porte sur la prise en compte ou non des activations survenant à la date $t$. Contrairement à $\overline{w}_i(\phi_k, t)$, la fonction $\underline{w}_i(\phi_k, t)$ ne considére pas les tâches dont l'activation survient à la date $t$.

Ces deux fonctions bornent la charge de travail sur un intervalle de durée $t$ par rapport au début d'une période d'activité. Le fait d'avoir une borne et non une valeur exacte est dû aux tâches asynchrones dont les délais entre les activations sont bornés. Il est donc possible de formuler une borne sur les dates de démarrage d'une tâche ainsi que sur sa date de terminaison dans une période d'activité. La date de démarrage d'une tâche étant la date à laquelle une tâche active commence un nouveau travail.

**Lemme 5.1.** *Pour une période d'activité $\langle \mathbf{\Phi} \rangle$, une borne supérieure sur la date de démarrage de la qème activation de la tâche $i$ est donnée par l'équation suivante :*

$$\begin{aligned} s_{i,q}(\mathbf{\Phi}) &= \underbrace{B_i}_{(1)} + \underbrace{(q-1)e_{i,0}}_{(2)} + \underbrace{\sum_{j \in hp(i)} \underline{w}_j(\phi_{src(j)}, s_{i,q}(\mathbf{\Phi}))}_{(3)} \\ &\quad + \underbrace{\sum_{j \in sp(i)} \underline{w}_j(\phi_{src(j)}, a_{i,q}(\phi_{src(i)}))}_{(4)} \end{aligned} \tag{5.5}$$

*avec*

$$B_i = \max\{e_{j,k} | \pi_j < \pi_i \leq \gamma_{jk}\}.$$

La partie (1) de l'équation (5.5) représente l'influence des tâches moins prioritaires que la tâche $i$ qui partagent une ressource avec elle (ressource commune ou ressource d'exécution). Le terme (2) intègre le nombre de fois où la tâche s'est déjà exécutée dans la période d'activité. La partie (3) représente le retard induit par les tâches plus prioritaires qui ont été activées dans la période d'activité avant que la tâche $i$ ne commence son exécution. Enfin, (4) est introduit pour prendre en considération les tâches de même niveau de priorité qui ont été activées avant ou en même temps que la tâche $i$.

**Lemme 5.2.** *Pour une période d'activité $\langle \mathbf{\Phi} \rangle$, une borne supérieure sur la date de terminaison de la qème activation de la tâche $i$ est donné par :*

$$\begin{aligned} f_{i,q}(\mathbf{\Phi}) &= B_i + qe_{i0} + \sum_{k \in ht(i)} \overline{w}_j \left(\phi_{src(j)}, f_{i,q}(\mathbf{\Phi})\right) + \sum_{j \in hp(i) \setminus ht(i)} \underline{w}_j(\phi_{src(j)}, s_{i,q}(\mathbf{\Phi})) \\ &\quad + \sum_{j \in sp(i)} \underline{w}_j \left(\phi_{src(j)}, a_{i,q}(\phi_{src(i)})\right) \end{aligned} \tag{5.6}$$

Dans l'équation (5.6), le troisième terme représente l'interfèrence des tâches plus prioritaires pendant l'exécution de la tâche $i$. De là, on en déduit immédiatement le temps de réponse pour la $q$ième activation de la tâche $i$ dans une période d'activation $\langle \mathbf{\Phi} \rangle$

$$r_{i,q}(\mathbf{\Phi}) = f_{i,q}(\mathbf{\Phi}) - a_{i,q}(\Phi_k). \tag{5.7}$$

Pour évaluer les temps de réponse de toutes les activations d'une tâche dans une période d'activité, il est nécessaire de borner le nombre d'activations. Pour cela, la durée d'une période d'activité est évaluée en considérant les charges maximales induites par les activations des tâches. Nous obtenons alors le lemme suivant :

**Lemme 5.3.** *La durée d'une période d'activité $\langle \mathbf{\Phi} \rangle$ de niveau $\pi_i$ est bornée par*

$$L_i(\mathbf{\Phi}) = B_i + \sum_{j \in hsp(i)} \overline{w}_j(\phi_{src(j)}, L_i(\mathbf{\Phi})) \tag{5.8}$$

Ce qui nous fournit une borne sur le nombre d'activations à calculer pour trouver le plus grand temps de réponse d'une tâche $i$ dans une période d'activité, soit :

**Théorème 5.1.** *Pour une période d'activité $\langle \mathbf{\Phi} \rangle$ de niveau $\pi_i$, le temps de réponse maximum d'une tâche $i$ associée à une source d'activation $k$ est borné par :*

$$R_i(\mathbf{\Phi}) = \max_{q \in Q_i(\mathbf{\Phi})} r_{i,q}(\mathbf{\Phi}) \tag{5.9}$$

*avec*

$$Q_i(\mathbf{\Phi}) = \left\lceil \frac{L_i(\mathbf{\Phi}) - \varphi_i(\phi_k)}{p_k} \right\rceil \tag{5.10}$$

### 5.3.3 Pire temps de réponse d'une tâche

Connaissant une borne sur le temps de réponse d'une tâche dans une période d'activité, il est possible d'évaluer son pire temps de réponse en considérant l'ensemble des périodes d'activités produites par le système $\langle T, \Lambda, src \rangle$. Remarquant que le temps de réponse dans une période d'activité $\langle \mathbf{\Phi} \rangle$ ne dépend que de $\mathbf{\Phi}$, nous avons pour le pire temps de réponse de la tâche $i$, $R_i$, l'expression suivante :

$$R_i = \max_{\mathbf{\Phi} \in \Gamma} R_i(\mathbf{\Phi}) \tag{5.11}$$

avec $\Gamma = \times_{k=1}^{M}(0..p_k]$ l'ensemble des valeurs que peut prendre le vecteur de phase.

Évaluer les temps de réponse pour l'ensemble des valeurs de $\Gamma$ n'est pas possible en terme de complexité. Par la suite, nous montrons qu'il est possible de ne pas considérer toutes les périodes d'activité, mais seulement un sous ensemble. Ce travail repose sur une constatation simple concernant les fonctions de charge, à savoir que pour une valeur de $t \in \mathbb{R}$ , l'argument maximum des fonctions $\underline{w}_i$ et $\overline{w}_i$ est égal à :

$$\phi^*_{k,i} = argmax_{\phi_k \in (0,p_k]} \underline{w}_i(\phi_k, t) = argmax_{\phi_k \in (0,p_k]} \overline{w}_i(\phi_k, t) = \begin{cases} p_k \text{ si } o_i \mod p_k = 0, \\ o_i \mod p_k \text{ sinon.} \end{cases}$$

et que les fonctions $\underline{w}_i$ et $\overline{w}_i$ croissent de manière monotone sur les intervalles $(0, \phi^*_{k,i}]$ et $(\phi^*_{k,i}, p_k]$.

Seul le cas d'un système où toutes les priorités et seuils de préemption sont différents a été prouvé. Le cas plus général avec la prise en compte de la politique FIFO comme second critère n'est qu'une conjecture inspirée des travaux de Bimbard et George (2006). Pour le cas sans FIFO, le pire temps de réponse doit être recherché dans une période d'activité où pour chaque source d'activations, une tâche de $hp(i) \cup \{i\}$ est activée au début de la période d'activité. Pour une source d'activation $k$ et une tâche $i$, nous définissons $X_{k,i}$ comme étant l'ensemble des phases à explorer, soit :

$$X_{k,i} = \{\phi^*_{k,j}\}_{j \in hsp(i) \cap \theta_k}$$

Cet ensemble a une taille au plus égale à $|\theta_k|$.

**Théorème 5.2.** *Pour un système $\langle \mathcal{T}, \Lambda, src \rangle$ pour lequel toutes les priorités et les seuils de préemption sont différents, la valeur maximum du temps de réponse de la tâche $i$, $R_i$, est obtenue pour un vecteur de phase $\mathbf{\Phi}$ appartenant à l'ensemble $\times_{k=1}^{M} X_{k,i}$ :*

$$R_i = \max_{\mathbf{\Phi} \in \times_{k=1}^{M} X_{k,i}} R_i(\mathbf{\Phi}) \tag{5.12}$$

Dans le cas général avec des priorités et des seuils de préemption quelconque, nous conjecturons que le pire temps de réponse d'une tâche $i$ est produit pour un vecteur de phase tel que :

— pour chaque source d'activation, une tâche de $hsp(i)$ est activée au début de la période d'activité, c'est-à-dire les ensembles $X_{k,i}$,

— pour la source d'activiation $k$ associée à la tâche $i$, il faut en plus du cas précédent, intégrer les phases où la $q$ème activation de la tâche $i$ survient en même temps que celle d'une tâche ayant le même niveau de priorité. L'ensemble ainsi décrit, que nous notons $Z_i$, est composé des solutions $z \in (0, p_k]$ de l'équation :

$$q.p_k + \varphi_i(z) = r.p_l + \varphi_j(\phi_l)$$

avec $q, r \in \mathbb{N}, l \in [1..M] - k, j \in sp(i) \cap \theta_l$ et $\phi_l \in X_{l,i}$.

**Conjecture 5.1.** *Pour un système $\langle \mathcal{T}, \Lambda, src \rangle$ avec une politiques d'ordonnancement FIFO comme second critère, la valeur maximum $R_i$ est obtenue pour un vecteur de phase $\mathbf{\Phi}$ appartenant à l'ensemble $\times_{k=1}^{M} \bar{X}_{k,i}$ :*

$$R_i = \max_{\mathbf{\Phi} \in \times_{k=1}^{M} \bar{X}_{k,i}} R_i(\mathbf{\Phi}) \tag{5.13}$$

*avec $\bar{X}_{k,i} = X_{k,i}$ pour $k \neq src(i)$ et $\bar{X}_{k,i} = X_{k,i} \cup Z_i$ pour $k = src(i)$.*

## 5.4 Discussion sur la complexité et expérimentations

Le nombre de phases à explorer pour le calcul de l'équation (5.12) est au pire égal à $\prod_{k=1}^{M} |\theta_k|$. Pour le cas d'un système de tâches indépendantes non-concrètes, c'est-à-dire pour lequel toutes les sources d'activations ne sont associées qu'à une seule tâche ($|\theta_k| = 1$) et que l'on ne connait pas leur date de démarrage ($\omega_k = \perp$), nous retrouvons bien l'instant critique où toutes les tâches sont activées au début de la période d'activité, ce qui rend l'évaluation simple et rapide. Par contre, dans le cas général, cette complexité peut devenir importante rendant impossible l'évaluation [1].

Ce problème est classique pour les méthodes d'analyse d'ordonnancement avec des transactions (Tindell, 1994) qui est l'équivalent des tables d'ordonnancement d'AUTOSAR. La littérature fait apparaître deux approches pour contourner la difficulté. La premier consiste a produire un calcul approximé avec un complexité réduite. Cette méthode a été proposé pour la première fois par Tindell (1994) puis formalisée et étendue par Palencia et González Harbour (1998). Plus récemment, Mäki-Turja et Nolin (2004b) réduisent la sur-approximation en linéarisant les fonctions d'arrondi $\lceil . \rceil$ et $\lfloor . \rfloor$.

La seconde approche consiste en un travail d'optimisation des algorithmes pour en réduire leur complexité calculatoire. Mäki-Turja et Nolin (2004a) représentent statiquement les

1. Pour $N$ tâches, $\prod_{k=1}^{M} |\theta_k|$ est borné par $e^{N/e}$. Plus précisément, la valeur maximale est égale à $3^{N/3}$ si $N \equiv 0 \,(\mod 3)$, $4 \cdot 3^{(N-4)/3}$ si $N \equiv 1 \,(\mod 3)$ et $2 \cdot 3^{(N-2)/3}$ si $N \equiv 2 \,(\mod 3)$.

fonctions cycliques en les pré-calculant puis en les utilisant pour l'évaluation des temps de réponse. L'avantage de cette méthode est de réduire fortement le nombre d'itérations et de calculs à faire pendant la recherche du point fixe, mais peut être coûteuse lors de l'initialisation.

La mise en œuvre de ces méthodes sur le modèle dédié à AUTOSAR est présentée dans (Hladik, 2016) et ne sera pas détaillée ici. Ce travail propose une première approximation des fonctions suivants la démarche de Tindell (1994) en réduisant la complexité à $\sum_{k=1}^{M} |\theta_k|$. La méthode de Mäki-Turja et Nolin (2004b) est aussi étendue au modèle AUTOSAR. Enfin l'approche consistant à accélérer l'estimation de la borne en pré-calculant des éléments récurrents a aussi été instancié, mais contrairement à (Mäki-Turja et Nolin, 2004a), elle introduit une sur-approximation.

Les expérimentations qui ont été réalisées montrent de manière attendue que la performance des algorithmes en précision est inverse à la performance en temps de calcul et cela même pour la méthode utilisant des tables pré-calculées. En effet, bien que cette méthode soit rapide, elle introduit dans le modèle une sur-estimation sur le pire temps de réponse qui n'est pas acceptable. Finalement, le meilleur compromis est trouvé pour la méthode s'inspirant de (Mäki-Turja et Nolin, 2004b).

## 5.5 Bilan et analyse retrospective

Ce travail a montré qu'il est possible d'étendre les méthodes d'analyse d'ordonnançabilité que l'on rencontre dans la littérature pour répondre à un besoin industriel. Cela passe par une connaissance approfondie des méthodes et une analyse des standards industriels afin de produire une abstraction efficace et suffisamment expressive pour prendre en considération la complexité d'un système réel. Cette maîtrise du standard, nous a aussi permis de poursuivre les travaux dans le cadre du projet ANR RESPECTED.

Des travaux complémentaires restent à faire, en particulier sur l'extension de l'analyse d'ordonnançabilité avec une politique d'ordonnancement mixte (priorités fixes et FIFO). Il serait aussi souhaitable d'appuyer ce travail sur des cas d'applications concrets afin d'illustrer une démarche complète de conception et de vérification dans le domaine automobile. Des travaux récents continuent dans cette voie comme ceux de Pollex *et al.* (2013) qui étendent l'analyse d'ordonnançabilité à avec des modèles où les variations de vitesse angulaire dans une application de contrôle moteur (c'est-à-dire les variations sur la récurrence des activations) est prise en considération.

Enfin il serait possible d'intégrer la méthode d'analyse exposée ici dans un outil de configuration de systèmes tel que celui proposé par Mehiaoui *et al.* (2013). Pour cela une formalisation du problème sous forme d'un problème d'optimisation linéaire mixte est une solution envisageable, mais nécessite la maîtrise de techniques d'optimisation bi-niveau mixtes pour être résolu. Un travail préparatoire sur ces aspects a été réalisé dans le cadre d'un stage de master avec Thomas Ridremont, mais n'a pas débouché sur des résultats concrets.

CHAPITRE 6

# Vérification de systèmes ordonnancés par méthode formelle

Ce chapitre est consacré à l'utilisation des méthodes formelles et du model-checking pour vérifier l'ordonnançabilité d'un système temps réel. Ce travail s'appuie sur le langage Pola développé par Florent Péres (2010) dans la cadre de sa thèse et est complétée par une étude sur l'utilisation de ce même langage pour valider un modèle AADL.

Ces travaux montrent comment utiliser des méthodes formelles pour vérifier des propriétés temporelles, en particulier au niveau de l'ordonnancement. Nous verrons dans une premier temps ce qu'est le langage Pola et comment il est possible de l'utiliser pour modéliser différents ordonnanceurs. Nous aborderons ensuite l'ensemble de la chaîne d'outillage mise en place et les résultats produits.

Une second partie s'attache à inscrire cette démarche dans un processus de conception s'appuyant sur le langage AADL. Les travaux publiés en relation avec ce chapitre sont (Péres *et al.*, 2009b,a; Hladik *et al.*, 2010; Péres *et al.*, 2011).

## 6.1 Pola : un langage dédié au domaine des systèmes temps réel vérifiables par model cheking

Pola (Péres, 2010) est un langage spécifique pour les systèmes de tâches temps réel ordonnancés. Son expressivité est limitée au domaine des systèmes temps réel, ainsi que par ses possibilités en terme de vérification, c'est-à-dire que toute notion ne s'intégrant pas dans une approche de vérification par *model-checking* en est exclue. Cette limitation a cependant un intérêt important : tout modèle exprimé en Pola est garanti vérifiable, dans les limites calculatoires du *model-checking*.

La technique du *model-checking* consiste en l'exploration exhaustive des états du système afin de décider si oui ou non une propriété, définie par une formule logique, est valide pour le modèle. De part son exhaustivité, cette technique souffre du problème, dit de l'*explosion combinatoire*, qui est une limite pratique où les états du modèle sont si nombreux qu'ils ne peuvent pas être représentés en mémoire. D'autre part, le *model-checking* est sujet à une impossibilité théorique : certaines caractéristiques, comme la préemption d'une tâche,

rendent impossible la vérification automatique de l'accessibilité des états, c'est ce que l'on appelle l'*indécidabilité* de l'accessibilité.

Malgré ces problématiques, le *model-checking* demeure une technique attrayante, car elle est applicable à une large gamme de modèles et est automatique, ce qui permet de concevoir une chaîne de vérification qui cache entièrement les détails du processus de vérification à l'utilisateur. Les outils accompagnant le langage Pola ont été conçus dans ce cadre.

La spécialisation de Pola au domaine de l'ordonnancement permet la modélisation d'un système de manière intelligible pour un utilisateur expert en temps réel, tout en cachant les détails de l'analyse à l'aide d'une chaîne de traduction et de vérification automatique. Ainsi, un modèle Pola est traduit en réseau de Petri étendu, puis la vérification proprement dite est effectuée à partir de cette traduction. La figure 6.1 présente l'ensemble de la chaîne de vérification qui peut être mise en oeuvre à partir d'une modélisation faite en Pola.

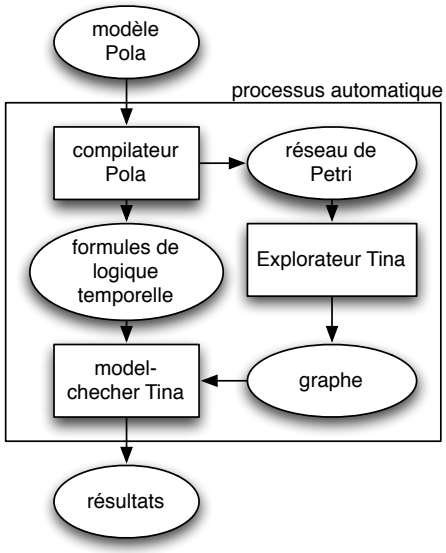



FIGURE 6.1 – Chaîne de vérification basée sur Pola

### 6.1.1 Présentation du langage Pola

Le langage Pola est composé de deux parties distinctes : une partie déclarative et une partie comportementale. La partie déclarative du langage permet de modéliser les ressources d'exécution et les tâches d'un système temps réel. Une tâche (**task**) est un flot d'exécution séquentiel qui peut être décomposé en actions (**action**) pour représenter son comportement interne. Les caractéristiques temporelles associée à une tâche sont sa période (**period**), son échéance (**deadline**) et son temps d'exécution (intervalle associé aux actions). La figure 6.2 donne un aperçu du méta-modèle du langage [1]. La sémantique de Pola est décrite en détail dans (Péres, 2010). Des exemples concrets d'utilisation du langage sont donnés dans les parties 6.1.2 et 6.2.

Une tâche est attachée à une politique d'ordonnancement qui contrôle son exécution. Se basant sur les travaux de Migge (1999), une politique d'ordonnancement est définie par une fonction d'ordre (« min » pour un ordre croissant ou « max » pour un ordre décroissant)

1. Un méta-modèle donne la syntaxe abstraire d'un langage, c'est-à-dire les règles de construction du modèle. Une syntaxe abstraite est utilsée pour représenter la structure fondamentale d'un langage et des relations entre les différents élements.

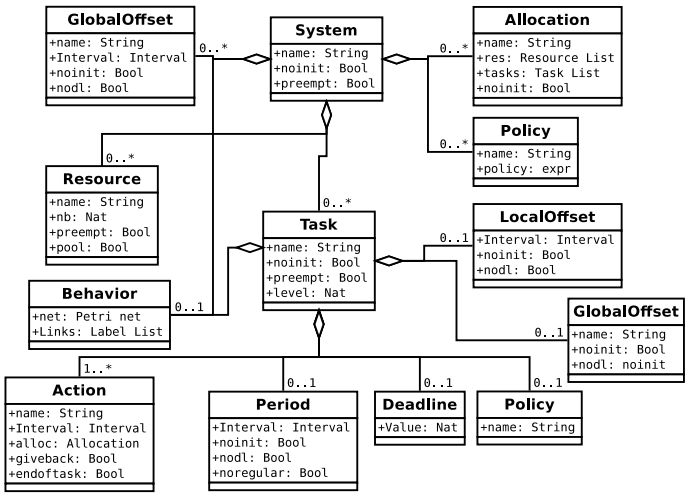



FIGURE 6.2 – Méta-modèle du langage Pola

entre les tâches dont le critère est évalué par une combinaison linéaire des caractéristiques des tâches, par exemple « $\min d$ » signifie qu'à un instant donné la tâche devant être exécutée est celle ayant la plus petite échéance courante. Les opérateurs d'ordre pouvant être utilisés sont min et max et les caractéristiques des tâches sont sa durée d'exécution ($C$), sa période ($P$), son échéance ($D$) et une priorité fixe ($L$). Ces paramètres (sauf la priorité) peuvent aussi être utilisés de manière dynamique dans les expressions, pour cela on utilise une écriture en minuscule : $c$, $p$, $d$.

Les politiques usuelles telles que *Rate Monotonic* ($\min P$) ou *Deadline Monotonic* ($\min D$) sont prises en compte, ainsi que toute politique *ad-hoc* dépendant d'un niveau de priorité fixé par l'utilisateur – lequel niveau est attribué à chacune des tâches. Les politiques dynamiques *Earliest Deadline First* (EDF) ($\min d$) ou *First In First Out* (FIFO) ($\min p$) peuvent être décrites en utilisant des caractéristiques dynamiques des tâches, mais ne peuvent pas être vérifiées avec l'implémentation courante de l'outil.

L'ordonnancement sous Pola est défini à l'aide de deux éléments : la politique d'ordonnancement, liée aux tâches comme expliqué ci-dessus, et l'allocation (**allocation**) définissant les ressources dont les tâches ont besoin pour s'exécuter. Une tâche ne peut s'exécuter que si toutes les ressources nécessaires sont libres (ou récupérables en préemptant les tâches qui les possèdent) et si elle est la plus prioritaire parmi les tâches actives pouvant s'exécuter sur cet ensemble de ressources (en admettant que l'ensemble de ressources ne permet d'exécuter qu'une tâche à la fois, sinon c'est le sous-ensemble de tâches les plus prioritaires qui est exécuté, selon les ressources disponibles).

Une partie comportementale permet d'ajouter des comportements non génériques à une tâche. Bien que s'éloignant de la démarche des langages spécifiques qui masque la complexité du langage formel, l'utilisateur peut décrire les comportements à l'aide d'un réseau de Petri dans une partie dédiée (introduite par **behavior**).

Les parties déclaratives et comportementales de Pola sont liées entre elles à l'aide d'*accesseurs* de deux types : ceux d'*état* et ceux d'*action*. Ils se traduisent, après transformation de Pola, par des étiquettes qui servent à synchroniser les réseaux de Petri issus des parties déclaratives et comportementales. Un accesseur d'état permet d'accéder à l'état d'une variable de Pola. Par exemple, l'état activée d'une tâche est accessible grâce à l'accesseur *sys.task.released*, avec *sys* le nom du système auquel appartient la tâche *task*. Un accesseur d'action rend accessible les actions qui se produisent dans un modèle Pola, comme

par exemple, l'allocation d'une ressource à une tâche par l'ordonnanceur, ou la fin d'une action.

### 6.1.2 Chaîne de vérification pour un modèle en Pola

Un modèle en Pola est automatiquement traduit en un modèle en réseau de Petri étendus par la notion de temps (Merlin et Farber, 1976), une notion de priorité (Berthomieu *et al.*, 2006), ou plus généralement une notion de contraintes sur les quantités temporelles du réseau avec les relations d'inhibitions et de permissions de (Agerwala et Flynn, 1973), ainsi que l'extension des chronomètres (Berthomieu *et al.*, 2004a) pour modéliser la préemption. Des formules de logique temporelle sont aussi produites afin de vérifier les propriétés attendues. Par exemple, « est-ce que les échéances des tâches sont toujours respectées ? » Ces deux éléments sont utilisés comme entrée de la boîte à outils Tina (Berthomieu *et al.*, 2004b) pour réaliser l'analyse du modèle (voir figure 6.1). L'analyse conclue au respect ou non des propriétés.

Pour illustrer cette démarche, nous utilisons un modèle simple décrit en Pola dans la figure 6.3. Cet exemple est composé d'une ressource `proc` (ligne 2) et de deux tâches `T1` (lignes 4-9) et `T2` (lignes 10-15). La politique d'ordonnancement est de type Rate Monotonic (ligne 3) ce qui implique que `T1` a une priorité plus élevée que `T2` car il a une période plus petite.

```
1  system simple is
2   res proc is preemptable
3   policy RM is min P
4   task T1 is
5    action a1 in [1,1] with alloc
6    period [2,2]
7    deadline 1
8    policy RM
9   end
10  task T2 is
11   action a1 in [1,1] with alloc
12   period [3,3]
13   deadline 2
14   policy RM
15  end
16  allocation alloc is
17   resources proc
18   tasks T1, T2
19  end
```

FIGURE 6.3 – Un exemple en Pola d'un système de deux tâches ordonnancées

Le modèle en réseau de Petri est fournie en entrée de Tina qui se charge d'explorer l'ensemble des comportements admissibles. Le résultat est une structure de Kripke qui représente l'ensemble des comportement du système.La structure de Kripke est ensuite utilisée comme entrée dans un model-checker et les trois propriétés basiques en logique temporelle linéaire suivantes sont ensuite vérifiées :

- **Est-ce qu'une tâche $T_i$ rate son échéance ?** $[]\neg DLMiss_{T_i}$ avec $DLMiss_{T_i}$ l'événement qui représente le non respect de l'échéance de la tâche $T_i$.
- **Est-ce qu'une tâche exécute au moins une fois chacune de ses actions ?** $\diamond Action_j^{T_i}$ avec $Action_j^{T_i}$ l'action $j$ de la tâche $T_i$.
- **Est-ce que chaque action s'exécute une infinité de fois ?** $[] \diamond Action_j^{T_i}$.

L'utilisateur ne manipule que le modèle en Pola pour obtenir le résultat de la chaîne de vérification. Le but est de cacher à l'utilisateur le processus de vérification. La figure 6.1 fournit la sortie de la chaîne de vérification. Dans l'exemple, aucune tâche ne rate son échéance et aucune action n'est considérée comme du code mort.

```
Generating ktz ...
# net noname, 15 places, 11 transitions                                  #
# bounded, not live, possibly reversible                                 #
# abstraction        count      props      psets       dead       live #
#      states           27         15         12          0         27 #
# transitions           34         10         11          3          8 #

Begin properties checking ...
**************************
*Deadline Misses Checking*
**************************
Loading graph behavior ... DONE
Is there any deadline miss in the system ?       NO
Does task simple.T1 miss its deadline ?          NO
Does task simple.T2 miss its deadline ?          NO
********************
*Deadcode Checking*
********************
Loading graph behavior ... DONE
For all possible executions, will the system execute action simple.T1.a1 ?   YES
For all possible executions, will the system execute action simple.T2.a1 ?   YES
Loading graph behavior ... DONE
Is live simple.T1.a1 ?    TRUE
Is live simple.T2.a1 ?    TRUE
```

Table 6.1 – Exemple de sortie de la vérification d'un modèle en Pola

Dans l'exemple présenté, la transformation en réseau de Petri est compacte, ce qui est rarement le cas en pratique. Un modèle peut être complexe avec de nombreuses places et transitions. La complexité provient, dans la majorité des cas, des périodes et des offsets s'ils ne sont pas des intervalles réduits à un point, c'est-à-dire si le système a ou non des offset connus et si les périodes sont strictes ou bien représentent une durée d'inter-arrivée. La complexité explose aussi avec le nombre de tâches du système.

Une autre point important concerne l'automatisation de l'approche. L'indécidabilité n'est pas pris en compte dans la démarche et c'est à l'utilisateur de mettre en place une procédure pour le vérifier. Pour cela, il peut vérifier que le réseau est borné par une valeur $k$ (valeur à fixer à la main) ce qui est faisable avec Tina.

## 6.2 Exemples d'utilisation de Pola

Afin d'illustrer l'expressivité du langage Pola, nous fournissons deux exemples s'inspirant d'une part d'un système basé sur OSEK/VDX et un autre inspiré de (Kandasamy *et al.*, 2003) qui modélise un réseau de communication. Dans (Péres *et al.*, 2011), un troisième exemple basé sur ARINC est présenté. Nous ne l'aborderons pas ici car il n'apporte pas d'éclairage supplémentaire.

### 6.2.1 Modélisation d'un système OSEK/VDX

Nous commençons par modéliser un système se basant sur OSEK/VDX (voir partie 4 pour plus de détails sur ce standard). La figure 6.4 présente une description en Pola d'un système

pour lequel une tâche `T1` (ligne 4-10) est déclarée comme étant non-préemtible et les tâches `T2` (ligne 11-18) et `T3` (lignes 19-32) appartiennent à un même groupe de tâches, c'est-à-dire qu'elles ne sont pas préemptibles par une tâche appartenant au même groupe. La tâche `T1` doit pouvoir préempter les autres tâches suivant la politique d'ordonnancement choisie. Pour représenter cela, une ressource preemptible `proc` (ligne 3) est définie et chaque tâche l'utilise. De plus, `T1` est déclarée **not preemptable** (ligne 4).

Les tâches `T2` et `T3` se comportent de la même manière : tant qu'une tâche du même groupe est exécutée, aucune tâche du groupe ne peut l'interrompre. L'utilisation du paramètre **not preemptable** n'est pas souhaitable si l'on veut conserver la préemption par les tâches plus prioritaire qui n'appartiennent pas au groupe. Une nouvelle ressource `vproc` est spécifiée avec pour paramètre **not preemptable** (ligne 2) et se comporte comme une ressource virtuelle qui ne prend effet que pour les tâches qui l'utilisent.

Les allocations `alloc1` et `alloc2` (lignes 34 à 39) spécifient quelles tâches utilisent quelles ressources et quel ordonnanceur doit être considéré.

Remarquons un comportement particulier qui a été inspiré d'une application de contrôle moteur. Dans cette application des points de préeemption ont été définis, c'est-à-dire des instants où une tâche non préemptible est interrompue par une autre tâche du même groupe. Cette notion est modélisable en Pola en utilisant des actions. La tâche `T3` présente un tel comportement. Elle est divisée en deux actions successives (ligne 20-21). La libération de la ressource est donnée par le mot-clé **giveback** (ligne 20) à la fin de la première action. La mot-clé **endoftask** (ligne 21) quand à lui indique la terminaison de la tâche. Cependant, par défaut, les deux actions s'exécutent en parallèle quand la tâche est activée. Pour éviter ce comportement, il faut ajouter une description en réseau de Petri dans la partie **behavior** (ligne 26 à 31) spécifiant clairement l'ordre d'exécution.

Ici l'état initial est donné par le mot-clé **pl** (ligne 27) suivant du nom de la place et du nombre de jetons. Les transitions sont spécifiées à l'aide du mot-clé **tr**, par exemple la transition de $s1$ à $s2$ se nome $a1end$ (ligne 28). Les transitions sont liées aux actions correspondant par le mot-clé **lb**, par exemple, $act1$ est lié à $a1end$ (ligne 30). Cela signifie que la condition de tirage de $a1end$ est ajoutée à $act1$ et que donc l'action $act1$ ne peut être activée que si un jeton est présent dans $s1$. Quand l'action est terminée, le jeton est enlevé de $s1$ et un nouveau est créé dans $s2$, ce qui active $act2$.

**Analyse du cas d'étude OSEK.** Plusieurs expérimentations ont été conduites à partir du modèle OSEK, les différentes variantes des modèles utilisés sont :

- **Osek A** : le modèle originalement proposé,
- **Osek B** : la même version, mais avec une période de 60 pour la tâche T1,
- **Osek C** : la même version, mais avec une période de 30 pour la tâche T1 et une échéance de 6. L'échéance de la tâche T2 est elle aussi réduite à 17, ce qui signifie que seulement T1 et la première action de T3 peuvent se produire avec l'exécution de T2. L'échéance de la tâche T3 est de 25 et une quatrième tâche est ajoutée au second groupe avec une durée d'exécution de 1 et une période et une échéance égale à 20,
- **Osek D** : une cinquième tâche est ajoutée au cas Osek C. Cette tâche a la plus haute priorité, une période de 15 et une durée d'exécution ainsi qu'une échéance de 1.

Dans la version B, le modèle a été modifié de manière à ce que la méta-période soit doublée. Nous entendons par méta-période le temps qu'il faut pour que le système retrouve sont état initial. La version B et C différent par le nombre de tâches et par des échéances plus

```
system osek is
 res vproc is not preemptable
 res proc is preemptable
 not preemptable task T1 is
  action act1 in [5,5] with alloc1
  period [31,31]
  deadline 21
  level 7
  policy FP
 end
 task T2 is
  action act1 in [4,4] with alloc2
  period [73,73]
  offset [7,7]
  deadline 25
  level 10
  policy FP
 end
 task T3 is
  action act1 in [7,7] with alloc2 giveback
  action act2 in [8,8] with alloc2 endoftask
  period [97,97]
  deadline 45
  level 5
  policy FP
  behavior is
   pl s1 (1)
   tr a1end s1 -> s2
   tr a2end s2 -> s1
   lb indus.T3.act1 a1end
   lb indus.T3.act2 a2end
 end
 policy FP is max L
 allocation alloc1 is
  resources proc
  tasks T1
 allocation alloc2 is
  resources proc, vproc
  tasks T2, T3
end
```

FIGURE 6.4 – Modèle Pola d'un système OSEK/VDX

strictes pour la version C. La version D a pour but d'étudier l'influence du nombre de tâches sans changer la méta-période.

Le tableau 6.2 présente les données collectées en analysant les différents systèmes. Nous trouvons de gauche à droite : le nombre de places (#pl.) et le nombre de transitions (#tr.) générées par le réseau de Petri, le nombre de d'états (#états) et d'arcs (#arcs) du graph de comportement utilisé par Tina, le temps total pour analyser le graph sur un Xeon E5520 (temps) et la méta-période du système étudié (méta-période).

| | #pl. | #tr. | #états | #arcs | temps | méta-période |
|---|---|---|---|---|---|---|
| OSEK A | 28 | 25 | 61,105 | 62,384 | 20 s | 219,511 |
| OSEK B | 29 | 26 | 84,934 | 86,368 | 28 s | 424,860 |
| OSEK C | 36 | 37 | 211,426 | 225,486 | 1 m. 50s. | 424,860 |
| OSEK D | 42 | 45 | 365,513 | 428,463 | 5 m. 25s. | 424,860 |

TABLE 6.2 – Résultats obtenus pour les différentes versions du cas d'étude OSEK

Les expérimentations montrent que la méta-période est un facteur important de complexité ainsi que le nombre de tâches. Les analyses ont cependant abouties dans tous les cas. Il n'est pas raisonnable de comparer les temps d'analyse avec des méthodes analytiques classique d'ordonnancement. Remarquons cependant que ces analyses dépassent la simple vérification de l'ordonnançabilité puisque le comportement des tâches peut aussi être vérifié et que Pola offre une très grandes expressivité pour représenter des applications basées sur OSEK avec des comportements complexes.

### 6.2.2 Modélisation d'une réseau industriel time-trigged

Cet exemple est inspiré de (Kandasamy *et al.*, 2003) et illustre l'utilisation de Pola pour modéliser un réseau industriel dirigé par le temps. Le système est une application de contrôle commande dans laquelle des messages sont échangés entre les capteurs, les traitements et les actionneurs. Le réseau est composé de trois bus dirigés par le temps (Time-Triggered) de type FlexRay. La figure 6.5 fournit un aperçu du système. Trois types de tâches sont considérées : les tâches de capteur, les tâches d'actionneur et les tâches de traitement. Les tâches pour les capteurs et les actionneurs ont leur propre ressource d'exécution alors que celles de traitement sont mutualisées sur des processeurs. Toutes les tâches communiquent à travers le réseau.

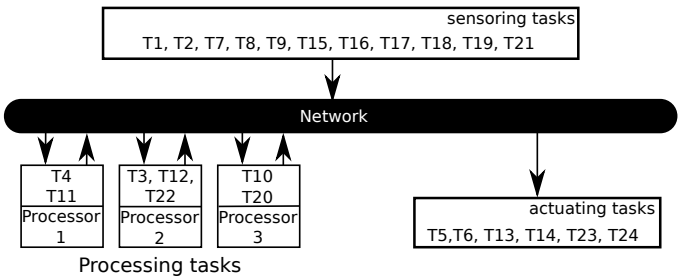



FIGURE 6.5 – Schéma du cas d'étude de contrôle commande avec un réseau industriel

Le modèle complet en Pola n'est pas fourni, seuls les patterns pour chaque composant sont présentés. Les tâches pour les capteurs et les actionneurs sont toutes spécifiées de la même manière. La seul différence est que les tâches pour les capteurs n'ont pas d'offset contrairement à celles des actionneurs. Ces offsets sont des artefacts dus à la méthode utilisée dans (Kandasamy *et al.*, 2003) : les tâches liées aux capteurs envoient périodiquement des

données à travers le réseau à une tâche de traitement, alors que les activations des tâches pour les actionneurs sont décalées à l'aide d'un offset pour attendre la transmission des données après traitement. Un tâche suit le pattern décrit dans la figure 6.6.

```
task spTask is
 action a1 in [B,B] with alloc
 (offset [C,C])
 deadline D
 period [E,E]
 policy DM
 behavior is
  tr send -> PrepareTransmission
  lb sys.Messagei.released PrepareTransmission
  lb sys.soTask.a1 send
end
```

FIGURE 6.6 – Pattern d'une tâche en Pola pour le modèle de (Kandasamy *et al.*, 2003)

Le comportement modélise les interactions sur le réseau : quand une tâche termine son action (`system.soTask.a1`) un message (`Messagei`) est signalé à l'aide d'un drapeau comme disponible pour transmission, puis lorsque le début de la transmission survient (`sys.Messagei.released`) le drapeau est remis à zéro.

L'allocation des actions est différente si la tâche est dédiée aux capteurs ou au traitement. L'allocation d'un traitement est définie normalement (la figure 6.5 montre les processeurs sur lesquels sont allouées les tâches). Par contre, une tâche de capteur à sa propre unité d'exécution. En Pola cela peut se modéliser de plusieurs manières :

— soit avec un seul ensemble de ressources tel que le nombre total de ressources soit suffisant pour toutes les tâches, dans ce cas une seule allocation est nécessaire,

— soit avec un ressource par tâche, dans ce cas il faut définir une allocation par tâche.

Les deux méthodes sont sémantiquement identiques, la première étant plus concise.

Les tâche pour les actionneurs sont modélisées en Pola par le pattern décrit dans la figure 6.7 où `allacActuating` définit l'allocation de la tâche de la même manière que pour les capteurs.

```
task actuatingTask is
 action a1 in [B,B] with allocActuating
 offset [C,C]
 deadline D
 period [E,E]
 policy DM
end
```

FIGURE 6.7 – Pattern d'une tâche d'actionneur en Pola pour le modèle de (Kandasamy *et al.*, 2003)

L'accès au réseau se fait suivant un protocole time-triggered, chaque nœud du système dispose d'un intervalle de temps pendant lequel il peut transmettre ses messages. Dans (Kandasamy *et al.*, 2003), un seul message est envoyé par intervalle, ainsi une trame n'est composée que d'un unique message. Un cycle complet de communication est une séquence récurrente d'intervalles. La modélisation de la communication en Pola est fait en ajoutant un comportement spécifique pour décrire les intervalles pendant lesquels un message est autorisé à être envoyé. Le modèle ainsi obtenu suit le pattern décrit dans la figuer 6.8 où $Bus$ est le nom du bus, $Ai$ la durée de l'intervalle $i$, $Mi$ le nom du message associé

```
1  behavior is
2    pl s1 (1)
3    tr Bus_slot1_slot2 [A1,A1] s1 -> s2
4    ...
5    tr Bus_sloti_slot1 [Ai,Ai] si -> s1
6    ...
7    tr Bus_slotn_slotn [An,An] si -> s1
8
9    lb sys.M1.Bus s1
10   ...
11   lb sys.Mi.Bus si
12   ...
13   lb sys.Mn.Bus sn
14 end
```

FIGURE 6.8 – Pattern en Pola d'un cycle de communication time-triggered

à l'intervalle et `sys.Mi.Bus` l'état de la ressource *Bus* quand elle est utilisée par `Mi`. L'initialisation est faite par l'activation de l'intervalle 1 (ligne 2).

Un message a un unique émetteur et peut avoir plusieurs récepteurs. Il est caractérisé par un temps de transmission $B$ et une échéance $D$. La figure 6.9 présente le pattern retenu pour un message. Il met en avant deux faiblesses du langage Pola. Tout d'abord, le réseau impose un slot spécifique pour chaque message, ce qui signifie que la tâche qui modélise le message doit garder la ressource. Cependant, par défaut l'action d'une tâche rend toujours sa ressource. C'est pourquoi une action « fantôme » (`ghost`) est ajoutée pour permettre à `a1` de garder la ressource. Deuxièmement, le mot-clé **endoftask** indique normalement quelle action termine la tâche. Malheureusement, **endoftask** implique aussi le mot-clé **giveback**. Il est donc nécessaire d'écrire un comportement spécifique pour obtenir celui souhaité.

```
1  noinit task message is
2   action a1 in [B,B] with Bus
3   action ghost in [0,0] with ghostalloc
4   deadline D
5   policy DM
6   behavior is
7    tr unrelease -> r
8    lb sys.message.released r
9    lb sys.message.a1 unrelease
10 end
```

FIGURE 6.9 – Pattern en Pola d'un message pour le modèle de (Kandasamy *et al.*, 2003)

**Analyse du cas d'étude time-trigged.** Ce cas d'étude montre que Pola est capable de modéliser un système temp réel distribué. En modélisant et analysant le modèle obtenu, une erreur a été constatée dans l'article (Kandasamy *et al.*, 2003) : le slot pour le message $m3$ n'est pas assez grand pour transmettre les données.

L'analyse avec le non respect de l'échéance de $m3$ prend moins de 2 secondes. Avec un temps de transmission plus faible pour le message $m3$, l'analyse complète a nécessité 3 min. 40 s sur un Xeon E5520 dont 2 min. 12 s pour construire le comportement. L'espace d'état comprend 70665 états et 488774 transitions. Dans ce cas toutes les échéances étaient respectées.

## 6.3 Analyse d'un modèle AADL à l'aide de Pola

Pour poursuivre dans l'automatisation de la démarche de vérification, nous nous sommes intéresser à l'utilisation de Pola comme langage pivot dans un processus de vérification en se basant sur un langage de haut niveau pour la modélisation d'un système temps réel.

Dans le cadre de la modélisation des systèmes embarqués critiques, deux langages sont souvent employés : UML avec le profil MARTE (Gérard *et al.*, 2007) et AADL (Architecture Analysis and Design Language) (`http://www.aadl.info`). Afin de vérifier les modèles issus de ces langages, des chaînes d'outils spécifiques sont développées. En 2009, le projet européen SPICES (`https://distrinet.cs.kuleuven.be/projects/spices`) recensait pas moins d'une douzaine d'outils et de méthodes autour du langage AADL permettant de vérifier des propriétés sur l'occupation mémoire, l'ordonnancement, la consommation d'énergie, la génération de code, etc..

L'étude menée avec Florent Peres et Xiaomu Shi a consisté à introduire Pola au sein d'une chaîne de transformation et de vérification prenant en entrée un modèle en AADL. Pour cela, nous nous plaçons dans le cadre de l'ingénierie des modèles pour définir les transformations permettant de passer d'un modèle AADL à un modèle Pola. Cependant, la principale difficulté rencontrée a été de garantir que cette transformation préserve la sémantique du modèle AADL dans celui en Pola. La travail s'est basé sur la version 2.1 de AADL.

### 6.3.1 Introduction (rapide) à AADL

AADL (Architecture Analysis and Design Language) est un langage de description d'architecture conçu pour les systèmes temps réel embarqués dans les domaines avionique, spatial, robotique, santé, etc. Il a été développé à partir des retours d'expérience de MethaH et a été porté comme base de standardisation sous l'autorité du SAE (International Society for Automotive Engineers) et de son ASD (Avionics System Division).

En novembre 2004, le SAE a produit la première version du standard AADL (SAE AS5506) et l'a fait évoluer depuis. Le standard SAE AADL est composé de : une spécification du langage avec une syntaxe textuelle, une sémantique et une représentation graphique ; un profil UML ; une spécification XML/XMI comme format de modèle ; et plusieurs annexes spécifiant des points particuliers tels que la formalisation du comportement, la conformité des interfaces avec C et Ada, l'extension du modèle d'erreur, etc.

Le langage AADL étant extrêmement riche, il est impossible d'en donner ici une vue exhaustive. Les lecteurs peuvent se référer aux notes techniques disponibles sur le site de la SAE pour avoir plus de détails ou bien directement se référer au standard pour disposer de la syntaxe et de la sémantique.

Le standard fournit les concepts de modélisation par composant d'une architecture logicielle et matérielle pour les systèmes temps réel critiques. Il permet de décrire une application en terme de tâches concurrentes ainsi que leurs interactions et leur allocation sur le matériel. Une architecture est décrite par une hiérarchie de composants. Nous ne présentons ici que les composants manipulés dans cet article.

Un composant logiciel est soit un *thread* pour représenter le flot séquentiel d'exécution d'une fonction et modéliser une unité ordonnançable ; soit un *process* pour représenter un composant structuré pour les groupes logiques de *thread*, *data* et *thread group* ; soit un *data*

pour modéliser les données manipulées dans le code source et les protocoles de partage en cas d'accès concurrents ; soit un *subprogram* (qui peut être appelé à partir de *thread* ou d'autres *subprogram*) pour représenter le point d'entrée d'une exécution dans le code source.

La plate-forme d'exécution est modélisée à l'aide du composant *processor* qui est une abstraction du matériel et du logiciel responsable de l'ordonnancement et de l'exécution des *threads*.

Le composant composite *system* qui fait le lien entre les applications logicielles et la plate-forme d'exécution.

Chaque composant est décrit en AADL par son type et son interface fonctionnelle, visible par les autres composants. L'interface comprend des propriétés (*properties*) avec les valeurs des attributs et des caractéristiques du composant, ainsi que des *feature* qui spécifient comment le composant est interfacé avec les autres. Plusieurs catégories de *features* sont distinguées : les *port* qui sont les interfaces de communication pour les échanges de données ou d'événements ; les *subprogam* qui sont les interfaces d'appel entrant ou sortant des sous-programmes ; et les *subcomponent access* qui représentent une donnée interne accessible par un composant externe.

L'implémentation d'un composant définit sa structure interne en terme de sous-composants et de connexions. Elle autorise la spécification des valeurs des propriétés non-fonctionnelles du composant, des états opérationnels ainsi que des instants de l'exécution permettant de passer d'un état à l'autre lors d'un changement de *mode*.

AADL peut être étendu pour introduire des spécificités propres à chaque application en plus des propriétés pré-définies. Ces extensions facilitent les modélisations spécifiques et permettent des analyses dédiées. Elles se présentent sous la forme d'annexes dans lesquelles sont déclarées les extensions du langage sous forme conceptuelle et syntaxique. Dans le travail présenté ici, il a été décidé de ne pas introduire de nouvelles propriétés pour procéder à l'analyse du comportement, mais uniquement d'utiliser celle qui sont fournies par le standard.

### 6.3.2 Transformation AADL vers Pola

Une chaîne de vérification basée sur des transformation de modèles n'est possible et pertinente que si la transformation garantit que le comportement exprimé par le modèle d'origine est identique à celui de la cible. Pour cela, il est nécessaire d'étudier les sémantiques de chaque langage et s'il est possible de traduire chaque élément d'un langage à l'autre. Cette partie met en évidence les similarités entre AADL et Pola au niveau comportemental ainsi que les points bloquants pour définir une transformation entre ces deux modèles.

#### Les principaux composants AADL et leur pendant en Pola

Le comportement d'une application est principalement capturée en AADL par les *threads*, qui se traduisent en Pola par la notion de tâche (**task**). La partie suivante présente les relations et les similitudes entre ces deux concepts. Le comportement temporel des *threads* est aussi lié à l'ordonnancement spécifié par les propriétés `Scheduling_Protocol` et

`Preemptive_Scheduler` des composants *processor*. Au niveau Pola, cela va se traduire par une politique d'ordonnancement (**policy**) et des allocations (**alloc**) (voir partie 6.3.2).

Un *subprogam* en AADL décrit une partie de code qui peut être appelée par des *threads*. Il a donc une influence sur l'exécution et le comportement de ces derniers. Cette notion se traduit en Pola sous la forme d'actions (**action**). Pour décrire les interactions entre les *subprograms* et les *threads*, il est nécessaire de disposer d'un modèle comportemental des *threads*, ce qui n'existe pas dans le langage standard AADL. L'annexe comportementale (Franca *et al.*, 2007) comble ce manque, mais, pour le travail présent, nous limitons l'étude aux comportements spécifiés dans le standard, c'est pourquoi nous n'évoquerons plus les *subprogram* par la suite.

Les données partagées entre les *thread* ont aussi une influence sur le comportement des applications. En AADL, les données sont modélisées à l'aide du composant *data* et la propriété associée `Concurrency_Control_Protocol` qui spécifie leur politique d'accès. En Pola, cela se traduit par une ressource (**res**). Les politiques d'accès nécessite quant à elle l'écriture de patterns dédiés dans la partie comportementale de Pola (**behavior**). De la même manière que pour les *subprogram*, n'ayant pas accès au comportement d'exécution d'un *thread*, nous supposerons que les ressources sont prises par un *thread* pendant toute son exécution. La partie 6.3.2 aborde ce point.

Le tableau 6.3 présente une synthèse des différentes transformations de AADL vers Pola.

| **System** | |
|---|---|
| AADL | Pola |
| **system** *name ...* **end** *name ;* | **System** *name* **is**... **end** |
| *actual_processor_binding => ... applies to ...* | spéciale |
| **Processor** | |
| AADL | Pola |
| **processor implementation** *name ...* **end** *name ;* | **res** *name* **is preemptable** |
| *Preemptive_Scheduler => false*<br>*Scheduling_Protocol => DMS*<br>*Scheduling_Protocol => EDF* | **res** *...* **is not preemptable**<br>**policy** *DM* **is** *min D* + **task policy** *DM*<br>**policy** *EDF* **is** *min d* + **task policy** *EDF* |
| **Thread** | |
| AADL | Pola |
| **thread implementation** `name ... end name ;` | **task** *name* **is**... **end** |
| *Dispatch_Protocol => periodic + Period => X*<br>*Dispatch_Protocol => sporadic + Period => X*<br>*Dispatch_Protocol => aperiodic*<br>*Compute_Execution_Time => X ..Y*<br>*Compute_Deadline => X*<br>*Priority => X* | **period** *[X,X]*<br>**period** *[X,w]*<br>**behavior** avec réseau de Petri spécifique<br>**action** *...* **in** *[X,Y]*<br>**deadline** *X*<br>**level** *X* |
| **Data** | |
| AADL | Pola |
| **data** *name ...* **end** *name ;* | **res** *name* **is not preemptable** |
| *Concurrency_Control_Protocol* | **behavior** avec réseau de Petri spécifique |

Table 6.3 – Synthèse des transformations de AADL vers Pola

### Comparaison entre les états d'un thread et d'une tâche

En AADL, le contrôle de l'exécution d'une application est modélisé par les *threads*. Un *thread* exécute une séquence de code quand il est activé et élu par l'ordonnanceur. Les

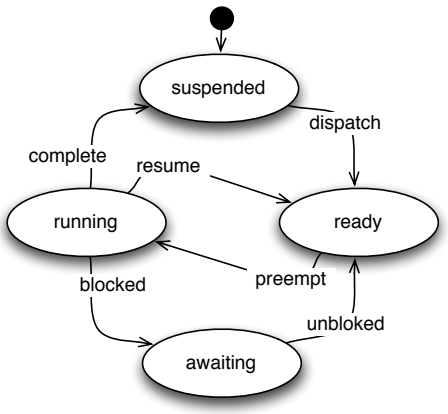



FIGURE 6.10 – Les états d'un *thread* AADL

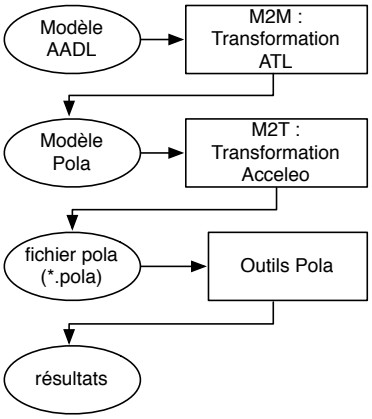



FIGURE 6.11 – Processus de vérification d'un modèle AADL avec Pola

différents états qu'il peut prendre — les états d'intialisation, d'erreur et de changement de mode ne sont pas considérés ici — sont (voir figure 6.10) :

— *suspended* : Une fois initialisé, le *thread* attend d'être activé pour commencer son exécution. Il retourne dans cet état à la fin de chacune de ses exécutions.

— *ready* : Une fois activé, le *thread* attend d'être choisi par l'ordonnanceur pour s'exécuter. Cet état est atteint plusieurs fois pendant une exécution s'il est préempté ou bloqué.

— *running* : Le *thread* exécute son code. Il reste dans cet état jusqu'à avoir terminé son exécution ou suite à une préemption, ou l'attente d'une ressource partagée.

— *awaiting* : Le *thread* attend la libération d'une ressource — ou le résultat d'un appel à un sous-programme, mais nous ne considérons pas ce cas.

Dans Pola, les éléments qui modélisent l'exécution de l'application sont les tâches (**task**) et les actions (**action**), ces dernières offrant une modélisation plus fine du comportement. Les états d'une tâche sont identiques à ceux d'un *thread*, mais ne sont pas explicitement représentés dans la modélisation Pola. Ils sont uniquement accesibles et identifiés à l'aide d'accesseurs (voir partie 6.1.1) utilisés comme propositions atomiques :

— *suspended* correspond à $\neg released$, avec *released* l'accesseur d'état dénotant le caractère actif d'une tâche, c'est-à-dire qu'une nouvelle instance de la tâche a été crée qui peut soit être en attente de son exécution, soit être en train de s'exécuter.

— *ready* correspond à $released \wedge (\exists r \in RES(task))(\neg task.r)$, avec $task.r$ est l'accesseur d'état dénotant la possession de le ressource $r$ par $task$, et $RES(task)$ est l'ensemble des ressources nécessaire à l'exécution de *task*.

— *running* correspond à l'état $released \wedge (\forall r \in RES(task))(task.r)$. Une tâche *task* s'exécute quand elle est activée et que toutes les ressources nécessaires à son exécution sont libres.

— *awaiting* correspond à l'état *ready* et $(\forall a \in ALLOC(task))(\neg a.active)$, où $a.task.active$ est l'accesseur d'état notant l'activité de l'allocation $a$ de la tâche *task*, et *ALLOC(task)* est l'ensemble des allocations permettant d'allouer les ressources necessaires à l'exécution de *task*. Dans cet état (comme l'état *ready*), la tâche est donc activée, mais ne possède pas toutes les ressources nécessaire à son exécution et (contrairement à *ready*) aucune allocation ne peut les lui donner car toutes sont désactivées.

### Comparaison entre les transitions d'un thread et d'une tâche

Les transitions entre les états d'un *thread* (voir figure 6.10) sont :

- *dispatch* : le *thread* est activé suite à l'arrivée d'un événement. Celui-ci peut être implicitement défini avec des patrons de type périodique (propriété `Dispatch_Protocol`) ou bien être directement connecté au port *Dispatch*. Il est ainsi possible de spécifier les événements déclencheurs du réveil en les connectant directement à ce port.
- *complete* : le *thread* termine son exécution. Lors de la terminaison d'un *thread*, un événement est produit sur le port `Complete`.
- *preempt* et *resume* : le *thread* est préempté ou reprend son exécution. Ces transitions sont contrôlées par l'ordonnanceur (voir la partie suivante).
- *blocked* et *unblocked* : le *thread* est bloqué suite à la demande d'accès à une ressource indisponible. Il est débloqué lorsque la ressource devient libre. Dans cet article, la prise de ressource partagée est toujours considérée comme survenant au début de l'exécution du *thread* et la libération à sa fin.

Au niveau Pola, les transitions associées aux états d'une tâche sont identiques à celles en AADL et peuvent être contrôlées soit directement à partir de la partie comportementale, à l'aide des accesseurs, soit de manière implicite.

### Propriétés des composant

Outre les informations architecturales apportées par les différents composants — *threads*, *process*, *processor*, etc. — les propriétés normatives de l'annexe A du standard apportent aussi des informations au niveau temporel et de l'ordonnancement. Chaque propriété d'un composant permet d'enrichir sa description et doit aussi être traduit en Pola.

**Les propriétés du composant *system*.** La propriété *Actual_Processor_Binding* spécifie l'allocation des *process*, des *threads* et des *data* sur les ressources de traitement. Dans le cas des *process*, tous les *thread* qui le composent, sont alors liés au même *processor*.

Si plusieurs *processors* sont disponibles et qu'aucune spécification n'est faite sur l'allocation d'un *thread*, alors celui-ci est considéré comme pouvant s'exécuter sur n'importe lequel suite à une décision d'un ordonnanceur globale, c'est-à-dire que l'ordonnanceur choisit globalement sur quelle ressource il alloue l'exécution d'un *thread*. Le standard AADL n'est pas suffisamment précis concernant la possibilité de migration des *thread* pendant leur exécution, c'est-à-dire la possibilité de changer de processeur suite à un ré-ordonnancement.

En Pola, l'allocation est faite à l'aide du mot **alloc** suivi d'une combinaison de tâches et de ressources.

**Les propriétés du composant *processor*.** La propriété `Scheduling_Protocol` spécifie par une chaîne de caractères le nom de la politique d'ordonnancement qui gère le partage de la ressource *processor* entre les *threads* qui lui sont liés.

Dans l'annexe A du standard, une liste de politiques est fournie en exemple, comme `RMS` pour la politique *Rate Monotonic* ou `EDF` pour la politique *Earliest Deadline First*. Cependant, les politique d'ordonnancement ne sont pas formalisées ce qui rend impossible

une traduction automatique pour la vérification. De plus, les exemples fournis pour illustrer l'énumération `Supported_Scheduling_Protocols` dans le standard AADL sont ambigus et parfois contradictoires (ex. page 276 de la version 2.1 de 2012).

Au niveau Pola, cela se traduit directement par l'ajout d'une politique d'ordonnancement avec le mot **policy**, décrite par une combinaison linéaire des caractéristiques des tâches (voir partie 6.1.1).

La propriété `Preemptive_Scheduler` définit si l'ordonnanceur est préemptif ou non, ce qui se traduit en Pola par la spécification **preemptable** ou **not preemptable** sur la ressource qui modélise le *processor*.

Les propriétés `Process_Swap_Execution_Time` et `Thread_Swap_Execution_Time` qui définissent le temps nécessaire à un changement de contexte, soit au niveau *process*, soit au niveau *thread*, ne sont pas pris en considération dans la traduction. En effet, les traduire consisterait à intercaler un délai entre le choix de la prochaine tâche et le démarrage effectif de la tâche. Le manque de précision sémantique sur les possibilités de perturbations au cours de ce délai nous ont conduit à ne pas considérer ce cas.

**Les propriétés du composant *thread*.** La propriété `Dispatch_Protocol` spécifie le comportement sur le réveil d'un *thread*. Elle peut prendre différentes valeurs :

- `Periodic` : Le *thread* est activé périodiquement. La période est spécifiée par la valeur de la propriété `Period`. Cette propriété est exprimée en Pola par la caractéristique **period** [`Period`,`Period`] associée à une tâche.
- `Sporadic` : Deux activations successives du *thread* sont toujours séparées par une quantité de temps minimale. Cette quantité est donnée par la valeur de la propriété `Period`. Cette propriété est traduite en Pola par **period** [`Period`,w[.
- `Aperiodic` : Le *thread* est activé par un événement externe qui est connecté au port `Dispatch`. En Pola, une tâche est apériodique si **period** [0,w[, mais la plupart du temps il est préférable ne pas spécifier de période et lui associer un comportement spécifique décrit en réseau de Petri dans la partie **behavior**. Ne rien spécifier comme période ou en déclarer une, décrit des comportements différents : dans le premier cas, la tâche ne pourra pas se réveiller spontanément (un évènement extérieur est donc requis) ; tandis que dans le second, la tâche peut être réveillée n'importe quand.

Pour compléter les propriétés temporelles sur l'activation d'un *thread*, `First_Dispatch_Time` donne la date de la première requête d'activation ce qui se traduit directement en Pola par un **offset [X,X]** avec `X` la valeur du décalage.

Les valeurs `Timed`, `Hybrid` et `Background` de la propriété `Dispatch_Protocol` n'ont pas été abordée dans cette étude.

La propriété `Compute_Execution_Time` est l'intervalle de temps pendant lequel le *thread* est dans l'état *compute*. Cela se traduit au niveau des tâches par une unique action associée à un intervalle temporel **action ... in** `[X,Y]`. De même, `Compute_Deadline` est la durée maximale pouvant séparer le *dispatch* d'un *thread* et sa terminaison, ce qui se traduit directement par **deadline** en Pola.

Pour terminer, la propriété `Priority` qui permet de définir des politiques d'ordonnancement à priorités fixes quelconques se traduit en Pola par l'ajout dans une tâche de **level** `X`, où `X` est la valeur entière définissant le niveau de priorité.

**Les propriétés du composant *data*.** Un composant *data* peut être accessible par plusieurs *threads* et nécessite donc la spécification d'une politique d'accès. En AADL, ces protocoles sont spécifiés à l'aide de la propriété `Concurrency_Control_Protocols` qui peut prendre diverses valeurs : `Semaphore`, `Interrupt_Masking`, `Priority_Ceiling`...

En Pola, la majorité de ces protocoles doivent être spécifiés de manière ad-hoc à l'aide d'un comportement (**behavior**). Pour l'instant, ce travail n'est pas fait et est conséquent, en particulier pour les protocoles dynamiques comme le protocole à priorité plafond.

Dans un premier temps, nous avons exprimé en Pola les données sous la forme d'une ressource non préemptible qui est associée à toutes les allocations des tâches qui ont accès à cette ressource. L'utilisation d'une ressource par un *thread* est spécifié en AADL à l'aide d'un `features` de type **requires data access** sur une donnée.

### 6.3.3 Implémentation de la transformation

Du point de vue de la mise en œuvre, nous nous appuyons sur les méthodes de l'ingénierie dirigée par les modèles (IDM) pour réaliser la transformation de AADL vers Pola. Pour cela, nous avons utilisé le langage de transformation de modèle ATL (Jouault et Kurtev, 2006) qui permet de spécifier les règles pour produire un modèle cible à partir d'un modèle source. Ce langage est inspiré du standard QVT de l'OMG et est disponible en tant que module dans le projet Eclipse.

Le métamodèle Ecore de AADL utilisé est celui proposé dans l'atelier TOPCASED (`https://www.polarsys.org/topcased`) en conformité avec les spécifications du SAE. Pour le langage Pola, un métamodèle ad-hoc a été produit en notation Ecore. Ce métamodèle comprend la partie propre au langage Pola ainsi que la partie permettant de décrire le comportement en réseau de Petri .

De plus, les modèles Pola devant être réécrits sous une forme textuelle pour servir d'entrée à la chaîne de traitements Pola, nous avons employé Acceleo (`https://eclipse.org/acceleo/`) de la société Obeo pour faire du *model-to-text* (M2T). Cet outil permet de produire simplement du code à partir de modèles. L'intégration de ces méthodes et outils, nous a permis de développer une chaîne complète de transformation dans cet environnement (voir figure 6.11).

Une première implémentation de la transformation a été réalisée sur un sous-ensemble de AADL. Cette implémentation comprend :

- la réécriture des composants de contrôle de comportement d'un thread,
- la prise en compte des politiques d'ordonnancement *Rate Monotonic* et à priorité fixe ;
- l'activation périodique, sporadique et apériodique sur le port *Dispatch*,
- la prise en compte des données partagées avec une politique d'accès de type sémaphore.

## 6.4 Bilan et analyse retrospective

Ce travail a permis d'étudier l'expressivité d'un langage formel pour modéliser des systèmes temps réel ordonnancés. Pola s'est révélé suffisamment complet pour représenter facilement

un grand nombre de systèmes et sa flexibilité permet de l'étendre facilement pour capturer des comportements moins communs.

La définition d'un processus complet automatisant la phase de vérification montre qu'il est possible pour un concepteur d'utiliser une démarche formelle sans pour autant être un expert en model-checking. La complexité de l'approche est ainsi cachée pour que le concepteur n'ait qu'à se focalise sur sa tâche de conception et non sur la difficulté de modélisation.

Par contre, les expérimentations menées montrent une limitation commune à ce genre d'approche à savoir une explosion de l'espace d'état dès que le système commence à être d'une taille significative. Cela reste un problème pour utiliser ces approches dans un cadre réaliste.

Ce travail a été une première étape vers le projet exposé dans la section 2.4.2 qui a pour but de mettre en place une chaîne automatique de conception et de génération d'architecture vérifiée à l'aide de méthodes formelles. La compréhension de Pola et des problèmes liés aux méthodes formelles permettent ainsi d'éviter certains pièges liés à la vérification de l'ordonnancement en particulier sur les aspects préemptifs.

CHAPITRE 7

# Simulation et évaluation d'ordonnanceurs multiprocesseurs

Depuis une vingtaine d'année de très nombreux algorithmes d'ordonnancement pour les architectures multicœurs ont été proposés. Ces politiques sont classiquement classées suivant la possibilité pour les tâches de migrer ou non d'un processeur à un autre au cours de leur exécution. On distingue ainsi les politiques partitionnées pour lesquelles la migration est interdite, de celles dites globales qui autorisent la migration des travaux. Des politiques plus récentes, qualifiées de semi-partitionnées, proposent des compromis entre ces deux approches afin d'en accroître leurs performances.

Être capable de comparer ces politiques de manière théorique ou pratique est un problème en soi. En effet, les architectures multicœurs introduisent de nouvelles complexités d'implémentation comme le partage des mémoires caches, la communications inter-cœur, les ressources partagées entre cœur, la distribution des prises de décision de l'ordonnanceur, etc., qui sont souvent difficile à mesurer et à évaluer.

C'est à ce problème que la thèse de Maxime Chéramy (2014) s'est attachée. Pour cela, un outil de simulation a été développé pour mesurer les performances des ordonnanceurs multicœurs, puis, des expérimentations ont été conduites afin d'évaluer et de comparer les nombreuses politiques existantes. Ce chapitre traite de ce travail. L'architecture du simulateur est abordé dans une première partie et montre en quoi elle répond bien au besoin que nous avions. Une second partie illustrent les résultats obtenus à partir des expérimentations. Un lecteur souhaitant obtenir plus de détails pourra se référer pour la première partie aux articles (Chéramy *et al.*, 2013, 2014a) et à (Chéramy, 2014) pour les évaluations. Ce travail a aussi donnée lieu à une publication faisant la synthèse des différentes politiques existantes dans le domaine de l'ordonnancement temps réel multiprocesseur (Chéramy *et al.*, 2015a).

Remarquons que depuis sa diffusion, le simulateur est régulièrement cité dans des publications en ordonnancement et que de nombreux étudiants d'établissement de recherche et d'enseignement sollicitent les développeurs pour les aider à l'utiliser.

## 7.1 Méthodes d'évaluation des politiques d'ordonnancement

Un très grand nombre d'algorithmes pour l'ordonnancement multiprocesseur ont été proposés depuis le milieu des années 1990. À titre d'exemple, le tableau 7.1, extrait de (Chéramy *et al.*, 2015a), liste une quarantaine d'approches en ordonnancement global et semi-partionné qui ont été majoritairement publiées sur ces dix dernières années. Bien que de nombreux travaux pour les évaluer aient été réalisés, il est difficile de se faire une idée des performances de ces politiques. En effet, les expérimentations sont conduites avec des hypothèses, des entrées et des implémentations différentes, ce qui rend les résultats difficilement comparables et reproductibles.

Remarquons aussi que les critères pour évaluer les algorithmes d'ordonnancement sont très variés. Ainsi, dans le cas de systèmes temps réel durs, la principale métrique est le nombre de tâches ordonnançables. Cependant, des critères comme le nombre de préemptions ou le nombre d'appels à l'ordonnanceur, ont aussi une influence sur les performances du système. Il est donc important de fixer ces critères et de les mesurer de manière homogène pour pouvoir comparer les ordonnanceurs.

Plusieurs moyens existent pour évaluer les performances des politiques d'ordonnancement. Le premier est d'utiliser des plate-formes réelles sur lesquelles sont faites des mesures. Les nombreux travaux sur Litmus-RT (Calandrino *et al.*, 2006), une extension du noyau Linux développée à l'Université de Caroline du Nord, adoptent cette approche. De même, Lelli *et al.* (2012) suivent cette démarche, mais avec une autre implémentation de Linux pour RM et EDF global. Les expérimentations sont souvent réalisées sur une architecture données, mais aussi parfois à l'aide de simulateurs *cycle-accurate* afin de faire varier les paramètres de la plate-forme matérielle, comme cela a été pratiquée dans (Zhu *et al.*, 2003). Ces approches expérimentales permettent d'obtenir des résultats très précis, mais nécessite de développer complètement les ordonnanceurs et de les intégrer dans un système d'exploitation, ce qui requière une grande expertise et peut être difficile à réaliser. De plus, les résultats mesurés dépendent fortement de l'architecture cible et des tâches utilisées.

Un second moyen pour évaluer les politiques consiste à utiliser un simulateur pour lequel les comportements du logiciel et du matériel sont fortement abstraits. De tels simulateurs permettent de prototyper rapidement des ordonnanceurs sans réaliser une réelle implémentation. De plus, les campagnes d'expérimentation peuvent être facilement conduites sur des machines usuelles. L'inconvénient majeur de cette approche est de masquer les détails d'implémentation d'un ordonnanceur ainsi que le comportement fin de la plate-forme, tel que les accès à la mémoire, alors qu'ils peuvent avoir des conséquences importantes sur les performances. Plusieurs simulateurs d'ordonnancement appartenant à cette catégroie existent et ont été utilisés pour faire des évaluations :

— MAST (Harbour *et al.*, 2001) propose un ensemble d'outils pour modéliser et analyser les systèmes temps réel distribués. Il est accompagné par des implémentations de tests de faisabilité et de sensibilité ainsi que d'un simulateur JSimMAST. À notre connaissance, ce simulateur ne propose aucune politique d'ordonnancement multi-cœur [1],

— Cheddar (Singhoff *et al.*, 2004) est lui aussi un simulateur couplé à des tests d'ordonnançabilité pour différentes architectures. Son écriture ayant été faite en ADA, il nécessite une recompilation complète à chaque ajout de politique d'ordonnancement, ce qui contraint son extensibilité,

1. Depuis la réalisation de ses travaux de nouveaux outils ont été développés pour MAST et les politiques multiprocesseurs sont intégrées, voir par exemple `http://mast.unican.es/gen4mast/`.

| Nom | Références |
|---|---|
| Ordonnanceurs globaux généralisant les politiques monoprocesseurs | |
| RM-US (Rate Monotonic with Utilization Separation) | Andersson *et al.* (2001) |
| EDF-US (Earliest Deadline First with Utilization Separation) | Srinivasan et Baruah (2002) |
| SM-US (Slack Monotonic with Utilization Separation) | Andersson (2008) |
| DM-DS (Deadline Monotonic with Density Separation) | Bertogna *et al.* (2005) |
| PriD/EDF$^{(k)}$ (Priority-Driven / Earliest Deadline First$^{(k)}$) | Goossens *et al.* (2003) |
| fpEDF (fixed-priority Earliest Deadline First) | Baruah (2004) |
| Tp-TkC (Fixed Priority with adaptiveTkC) | Andersson et Jonsson (2000) |
| GFL (Global-Fair Lateness) | Erickson *et al.* (2014) |
| EDZL (Earliest Deadline Zero Laxity) | Lee (1994) |
| EDCL (Earliest Deadline Critical Laxity) | Kato et Yamasaki (2008a) |
| FPZL, FPCL, FPSL (Fixed Priority Zero Laxity) | Davis et Kato (2012) |
| GLLF (Global Least Laxity First) | Mok (1983) |
| GMLLF (Global Modified Least Laxity First) | Oh et Yang (1998) |
| U-EDF (Unfair-EDF) | Nelissen *et al.* (2012) |
| Ordonnanceurs globaux de type PFair et ERFair | |
| EPDF (Earliest Pseudo-Deadline First) | Anderson et Srinivasan (2000b) |
| PF (Proportionate Fair) | Baruah *et al.* (1996) |
| PD (Pseudo-Deadline) | Baruah *et al.* (1995) |
| PD$^2$ (Pseudo-Deadline$^2$) | Anderson et Srinivasan (1999) |
| ER-PD$^2$ (Early-Release Fair Pseudo Deadline$^2$) | Anderson et Srinivasan (2000a) |
| PL (Pseudo-Laxity) | Kim et Cho (2011) |
| Ordonnanceurs globaux de type DPFair et BFair | |
| LLREF (Largest Local Remaining execution time First) | Cho et Ravindran (2006) |
| LRE-TL (Local Remaining Execution-Time and Local plane) | Funk et Nanadur (2009) |
| DP-WRAP (Deadline Partitioning-Wrap) | Levin *et al.* (2010) |
| BF (Boundary Fair) | Zhu *et al.* (2003) |
| BF$^2$ (Boundary Fair$^2$) | Nelissen *et al.* (2014) |
| NVNLF (No Virtual Nodal Laxity First) | Funaoka *et al.* (2008) |
| SA (Scheduling Algorithm) | Khemka et Shyamasundar (1997) |
| Ordonnanceurs semi-partitionnés | |
| EDF-fm (Earliest Deadline First-fixed or migrating) | Anderson *et al.* (2005) |
| EKG (EDF with task splitting and k processors in a Group) | Andersson et Tovar (2006) |
| EDHS (Earliest Deadline and Highest-priority Split) | Kato et Yamasaki (2008d) |
| EDF-WM (EDF Window constrained Migration) | Kato *et al.* (2009) |
| EDF-C=D (Earliest Deadline First with C=D) | Burns *et al.* (2012) |
| EDF-RRJM (EDF-Round Robin Job Migration) | George *et al.* (2011) |
| Ehd2-SIP ou EDDHP | Kato et Yamasaki (2007) |
| EDDP (Earliest Deadline Deferrable Portion) | Kato et Yamasaki (2008b) |
| RMDP (Rate Monotonic Deferrable Portion) | Kato et Yamasaki (2008c) |
| DMPM (Deadline Monotonic with Priority Migration) | Kato et Yamasaki (2009) |
| PDMS_HPTS_DS | Lakshmanan *et al.* (2009) |
| SPA2 (Semi-Partitioned scheduling Algorithm 2) | Guan *et al.* (2010) |
| HSP (Harmonic Semi-Partitioned) | Fan et Quan (2012) |
| EDF-BR (EDF with Bandwith Reservation) | Massa et Lima (2010) |
| EDF-os (EDF-based optimal semi-partitioned scheduling) | Anderson *et al.* (2014) |
| NPS-F (Notional Processor Scheduling-First-Fit bin-packing) | Bletsas et Andersson (2009) |
| Autres | |
| RUN (Reduction to UNiprocessor) | Regnier *et al.* (2011) |
| SPRINT (SPoradic Run for INdependent Tasks) | Baldovin *et al.* (2014) |
| Carousel-EDF | Sousa *et al.* (2013) |
| QPS (Quasi-Partitioning Scheduler) | Massa *et al.* (2014) |

TABLE 7.1 – Liste non exhaustive des ordonnanceurs multiprocesseurs globaux

— RTSIM (Casile *et al.*, 1998) est un simulateur d'ordonnancement temps réel développé depuis 1998, mais dont la dernière version disponible date de 2007 et pose des problèmes de compilation sur les systèmes récents,

— STORM (Urunuela *et al.*, 2010) et YARTISS (Chandarli *et al.*, 2012) sont deux simulateurs dédiés à l'évaluation des politiques d'ordonnancement. Leur architecture permet d'écrire facilement des nouvelles politiques d'ordonnancement et de les évaluer pour différents jeux de tâches. Cependant, à cause de son moteur de simulation basé sur un pas de temps, STORM ne permet pas de manipuler des échelles de temps variables, ce qui pose problème si l'on veut introduire, par exemple, les surcoûts systèmes. Quant à YARTISS, il est certainement l'outil le plus adapté pour mener à bien des expérimentations, mais son architecture autour de la consommation d'énergie et surtout le manque de souplesse de son interface apporte certaine limitations.

## 7.2 SimSo

Afin de mener une étude sur les performances des ordonnanceurs multiprocesseurs, un nouveau simulateur, baptisé SimSo (pour *Simulation of Multiprocessor Scheduling with Overheads*), a été développé. Ce simulateur est librement disponible sur `http://projects.laas.fr/simso/` avec des sources ouvertes. Une interface graphique, principalement à destination de l'enseignement, est aussi disponible. Enfin, une version web (s'exécutant dans un navigateur) est proposée.

Afin de manipuler des échelles de temps variées sans ralentir la simulation, l'architecture de SimSo se base sur une simulation à événement discret. Le moteur d'exécution utilisé est celui de SimPy [2]. Sa structure sous forme de processus permet de modéliser facilement le comportement des composants des système temps réel ce qui a facilité le développement de SimSo.

La structure de SimSo suit au plus proche les architectures d'ordonnanceur dans les systèmes d'exploitation, ainsi l'API s'inspire de celle de Linux et les timers sont clairement identifiés pour implémenter les mécanismes d'horloge. Les composants représentant les processeurs, les tâches, les travaux, etc., sont aussi clairement identifiés. Chacun de ces objets simule le comportement d'une partie du système : les tâches réveillent les travaux, les travaux émulent l'exécution du code d'une tâche, les timers lancent des traitements sur un processeur à un instant donné, etc.

Toujours dans une volonté d'être le plus réaliste possible, les surcoûts temporels du système sont simulés. Ceux-ci incluent les surcoûts directs comme les changements de contexte ou les appels à l'ordonnanceur, mais aussi les surcoûts indirects comme les temps de blocage des processeurs lors d'une prise de décision d'un ordonnanceur. Ces surcoûts concernent uniquement les aspects du système d'exploitation et ne prennent pas en considération ceux induits par l'exécution des tâches comme les surcoûts liés à la mémoire cache (cet aspect est abordé dans le chapitre 8).

### 7.2.1 Architecture du simulateur

Les principaux éléments qui composent l'architecture de SimSo sont représentés sur la figure 7.1. Ces éléments sont les classes : *Model*, *Processor*, *Task*, *Job*, *Timer* et *Scheduler*.

2. SimPy : http ://simpy.readthedocs.org/

Les objets instanciés à partir des classes *Processor*, *Task*, *Job* et *Timer* sont des processus au sens de SimPy, c'est-à-dire des entités actives qui peuvent attendre des événements ou exécuter du code.

La classe *Model* est le point d'entrée de la simulation. C'est elle qui est responsable d'instancier et de lancer les différents processus du système.

La classe *Scheduler* implémente une politique d'ordonnancement et est fournie par l'utilisateur. Ses méthodes sont appelées par les objets *Processor*. Bien entendu, un large ensemble d'ordonnanceurs est déjà mis à disposition avec le simulateur.

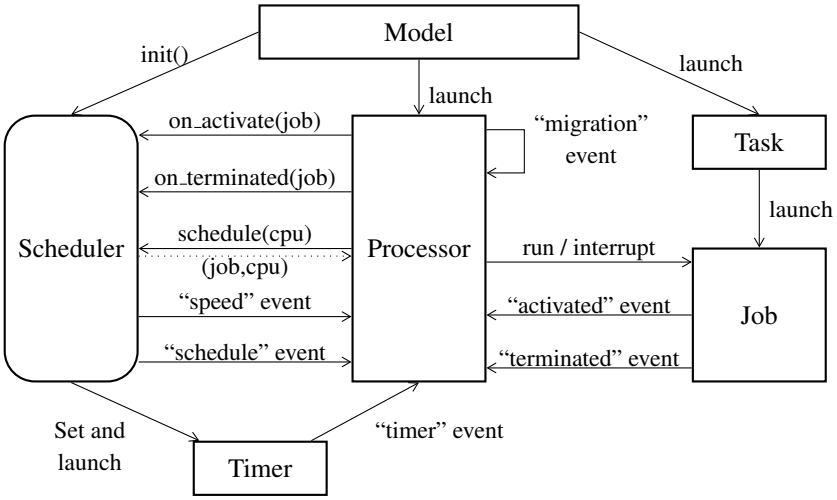



FIGURE 7.1 – Interactions entre les principales instances de classe. *Processor*, *Task*, *Job* et *Timer* sont des processus au sens de SimPy et peuvent avoir plusieurs instances.

La classe *Processor* a un rôle central dans la simulation du système car elle se charge de simuler le comportement du système d'exploitation et du processeur. Chaque processeur physique est représenté par une instance de la classe *Processor* et contrôle l'état des travaux (en exécution ou en attente) en fonction des décisions de l'ordonnanceur. Le processeur s'occupe aussi d'appeler les méthodes de l'ordonnanceur pour l'informer des événements liés à l'ordonnancement et pour obtenir des décisions d'ordonnancement.

Cette centralisation au niveau de la classe *Processor*, qui est aussi un processus SimPy, permet de simuler les surcoûts système tels que les changements de contexte ou les appels à l'ordonnanceur. Ainsi, tout code simulé s'exécute virtuellement sur un processeur, que ce soit un travail ou un service du système d'exploitation comme une fonction de l'ordonnanceur.

Le comportement d'un objet *Processor* est représenté de façon simplifiée par la figure 7.2. Si aucun évènement d'ordonnancement n'est à traiter, alors le processeur s'exécute normalement (branches 2 et 3). Dans ce cas, si un travail actif est assigné au processeur, alors celui-ci s'exécute. Sinon le processeur est simplement mis en attente. Lorsqu'un signal est reçu par le processeur, alors celui-ci interrompt l'exécution éventuelle du travail en cours pour traiter cet évènement. Dans ce cas (branche 1), le processeur simule l'exécution du code du système d'exploitation sur le processeur. Il est alors tout à fait possible d'associer des délais aux appels des méthodes de l'ordonnanceur et même de créer des zones critiques si on veut éviter que deux processeurs puissent appeler l'ordonnanceur en parallèle.

La figure 7.1 montre les différents types d'évènements qui sont susceptibles d'interrompre l'exécution d'un travail. Certains proviennent des travaux lorsqu'ils sont activés et lorsque

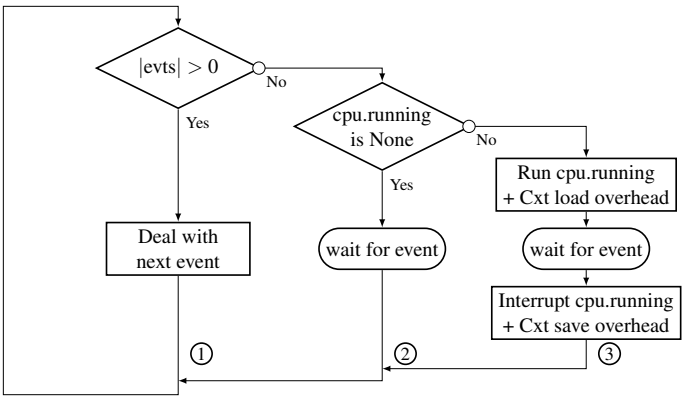



FIGURE 7.2 – Schéma d'exécution simplifié d'un *Processor*.

leur exécution se termine. D'autres proviennent de l'ordonnanceur qui peut demander à un processeur d'exécuter sa méthode *schedule*, ou encore de changer de vitesse d'exécution. Un *Timer* peut également se déclencher et interrompre un processeur. Enfin, un processeur peut interrompre un autre processeur pour appliquer une décision d'ordonnancement (ce qui l'équivalent d'un *inter-processor interrupt*, IPI).

### 7.2.2 Modèle de temps d'exécution d'un travail

La plupart des expérimentations avec simulation qui évaluent les politiques d'ordonnancement sont conduites en considérant les durées d'exécution des travaux égales à leur pire temps d'exécution (wcet). Cette hypothèse est particulièrement pessimiste pour différentes raisons : le wcet est une borne supérieure qui est rarement, voire jamais, atteinte par un travail et il est d'autant plus rare que tous les travaux atteignent leur wcet en même temps. L'utilisation des wcet introduit aussi un biais dans la comparaison des politiques en donnant un avantage aux politiques qui se basent sur les pires temps d'exécution pour prendre des décisions au détriment de celles qui naturellement s'adaptent au variation de charge processeur. L'article (Chéramy *et al.*, 2014b) montre quelques résultats mettant en avant ces problèmes.

Ces remarques justifient l'étude des politiques d'ordonnancement pour plusieurs modèles de temps d'exécution. Dans SimSo, ceci est pris en compte avec l'*Execution Time Model* (ETM) qui contrôle les durées d'exécution des travaux. Il n'existe qu'un seul objet ETM pour tout le système et est créé par la classe *Model* avant d'être passé en argument aux objets *Job* au moment de leur création. L'objet ETM est informé par les travaux de tous les évènements relatifs à leur exécution (voir figure 7.3). À partir de ces informations, cet objet est capable de fournir à tout travail, une borne inférieure de sa durée d'exécution restante. Notons que l'objet ETM ne communique qu'avec les travaux ce qui permet d'ajouter de nouveaux modèles de temps sans avoir à modifier les classes qui forment le cœur de la simulation. Les modèles disponibles actuellement dans SimSo sont listés dans le tableau 7.2.

Certains ETM ne permettent pas de déterminer la durée d'exécution d'un travail avant sa terminaison. C'est par exemple le cas du modèle prenant en compte l'impact du partage de cache. En effet, au moment où un travail s'exécute, il n'est pas possible de connaître les travaux qui seront exécutés en parallèle dans le futur. La solution retenue pour simuler ce comportement consiste à exécuter les travaux avec une durée inférieure à la durée réelle, puis de recommencer l'évaluation jusqu'à ce que l'exécution soit terminée. L'inconvénient

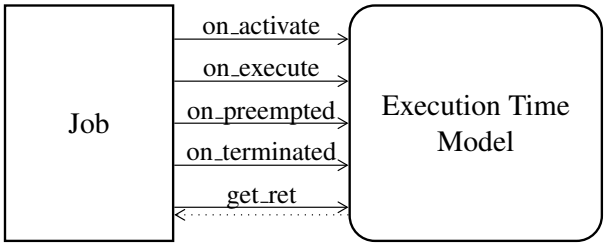



FIGURE 7.3 – Interface d'un *Execution Time Model*.

| Nom | Description |
|---|---|
| WCET | La durée d'exécution des travaux est toujours égale au WCET si la vitesse du processeur est de 100%. La durée d'exécution s'adapte proportionnellement à la vitesse du processeur. |
| ACET | Similaire au model WCET sauf le WCET est substitué par une valeur tirée aléatoirement selon une distribution normale centrée autour du ACET puis borné par le WCET. |
| Fixed Penalty | Similaire au modèle WCET sauf que la durée d'exécution est prolongée par une pénalité fixe après chaque préemption. |
| Cache Model | Utilisation de modèles statistiques pour l'évaluation de la durée d'exécution en tenant compte des caches (partagés ou non). |

TABLE 7.2 – Liste des *Execution Time Models* disponibles.

de cette solution est que si la borne inférieure renvoyée est trop éloignée de la réalité, de nombreux appels sont nécessaires. La convergence est malgré tout garantie puisque l'unité de base de la simulation est entière, égale au cycle processeur.

Ainsi, chaque travail simulant son exécution fait un appel à la méthode *get_ret* de l'ETM afin d'obtenir une borne inférieure de son temps restant d'exécution. Lorsque ce temps est écoulé, le travail appelle à nouveau cette méthode et ainsi de suite jusqu'à ce qu'elle retourne zéro.

Ces modèles de temps d'exécution permettent aussi d'implémenter des politiques d'ordonnancement utilisant le *Dynamic Voltage and Frequency Scaling* (DVFS). Le temps d'exécution d'un travail est simplement réévalué à chaque changement de vitesse des processeurs. Pour montrer la faisabilité de cette approche, une politique d'ordonnancement modifiant un simple rapport proportionnel entre la vitesse du processeur et l'exécution des travaux a été implémentée, mais SimSo eput être étendu à des modèles plus complexes sans difficulté.

### 7.2.3 Écriture d'un ordonnanceur avec SimSo

SimSo permet d'écrire des ordonnanceurs variés. Ces algorithmes s'appuient sur une interface définie pour être la plus proche possibles des mécanismes réellement disponibles dans les systèmes d'exploitation. Par exemple, pour les politiques qui provoquent des appels réguliers à l'ordonnanceur, la précision du pas de temps des timers doit être considéré comme une contrainte d'implémentation, contrairement aux études théoriques qui souvent supposent le temps comme une valeur réelle.

L'intérêt d'utiliser un simulateur est de simplifier les expérimentations en facilitant l'écriture des ordonnanceurs. Dans SimSo, les ordonnanceurs sont décrits dans des modules en

Python permettant ainsi d'avoir des abstractions de haut niveau. Remarquons que pour un vrai système, ce langage ne serait pas employé, par contre, les algorithmes restent les mêmes ainsi que leur logique. Ce qui est important pour rester proche d'un ordonnanceur réel sont les interfaces entre cette ordonnanceur et le reste du système. Dans notre cas, les méthodes servant d'interface sont :

— **init :** cette méthode sert à initialiser l'ordonnanceur au début de la simulation,

— **on_activate :** cette méthode est appelée à chaque fois qu'un travail est activé,

— **on_terminated :** cette méthode est appelée quand l'exécution d'un travail se termine ou quand un travail est avorté,

— **schedule :** cette méthode retourne la décision d'ordonnancement. Elle est appelée quand le processeur demande une décision d'ordonnancement. Cette requête fait habituellement suite à l'activation d'un travail, à sa terminaison ou à l'expiration d'un timer.

La figure 7.4 montre un exemple de code d'un ordonnanceur multiprocesseur de type Global Earliest Deadline First.

```
1  from simso.core import Scheduler
2
3  class G_EDF(Scheduler):
4      def init(self):
5          self.ready_list = []
6
7      def on_activate(self, job):
8          self.ready_list.append(job)
9          # Envoi d'un \'ev\'enement "schedule" au processeur.
10         job.cpu.resched()
11
12     def on_terminated(self, job):
13         # Envoi d'un \'ev\'enement "schedule" au processeur.
14         job.cpu.resched()
15
16     def schedule(self, cpu):
17         decision = None # No change.
18
19         if self.ready_list:
20             # recherche un processeur libre ou le processeur ex\'ecutant
21             # le travail le moins prioritaire.
22             key = lambda x: (1 if not x.running else 0,
23                              x.running.absolute_deadline if x.running else 0)
24             cpu_min = max(self.processors, key=key)
25
26             # Choisi le travail avec la plus grande des priorit\'es dans la liste des t\^aches pr\^etes.
27             job = min(self.ready_list, key=lambda x: x.absolute_deadline)
28
29             # Si le travail a une plus haute priorit\'e que celui s'ex\'ecutant sur le processeur :
30             if (cpu_min.running is None or
31                     cpu_min.running.absolute_deadline > job.absolute_deadline):
32                 self.ready_list.remove(job)
33                 if cpu_min.running:
34                     self.ready_list.append(cpu_min.running)
35                 # Ordonnance le travail sur le processeur.
36                 decision = (job, cpu_min)
37
38     return decision
```

FIGURE 7.4 – Code d'un ordonnanceur global Earliest Deadline First scheduler.

Une bibliothèque d'ordonnanceurs pour SimSo a été développée. Elle comprend actuellement plus de vingt-cinq algorithmes d'ordonnancement. Toutes les catégories d'ordonnanceurs sont représentées : politiques monoprocesseurs, des politiques multiprocesseurs partitionnées, globales et hybrides. Le tableau 7.3 offre un aperçu des politiques disponibles. À cela s'ajoute un ensemble d'algorithmes d'allocation des tâches aux processeurs pour les ordonnancements partitionnés.

### 7.2.4 Mener une expérimentation avec SimSo

L'utilisation de SimSo pour évaluer les politiques d'ordonnancement nécessite la génération d'ensemble de tâches. Pour cela, le nombre de tâches, les périodes, les échéances et

| Nom | Abréviation |
|---|---|
| **Monoprocesseur** | |
| Earliest Deadline First | EDF |
| Rate Monotonic | RM |
| Fixed Priority | FP |
| **Monoprocesseur avec DVFS** | |
| Static-EDF Pillai et Shin (2001) | Static-EDF |
| Cycle-Conserving EDF Pillai et Shin (2001) | CC-EDF |
| **Global multiprocesseurs par généralisation des algorithmes monoprocesseur** | |
| Global-EDF | G-EDF |
| Global-RM | G-RM |
| Earliest Deadline Zero Laxity Lee (1994) | EDZL |
| Global-Least Laxity First Mok (1983) | G-LLF |
| Modified Least Laxity First Oh et Yang (1998) | G-MLLF |
| Priority-Driven / EDF$^{(k)}$ Goossens *et al.* (2002) | PriD |
| EDF-US Srinivasan et Baruah (2002) | EDF-US |
| Global-Fair Lateness Erickson *et al.* (2014) | G-FL |
| U-EDF Nelissen *et al.* (2012) | U-EDF |
| **Global de type PFair** | |
| Earliest Pseudo-Deadline First Anderson et Srinivasan (2000b) | EPDF |
| PD$^2$ Anderson et Srinivasan (1999) | PD$^2$ |
| ER-PD$^2$ Anderson et Srinivasan (2000a) | ER-PD$^2$ |
| **Global de type DPFair** | |
| LLREF (ou LNREF) Cho et Ravindran (2006) | LLREF |
| LRE-TL Funk et Nanadur (2009) | LRE-TL |
| DP-WRAP Levin *et al.* (2010) | DP-WRAP |
| BF Zhu *et al.* (2003) | BF |
| NVNLF Funaoka *et al.* (2008) | NVNLF |
| **Multiprocesseur par approche semi-partitionnée ou hybride** | |
| EKG Andersson et Tovar (2006) | EKG |
| EDHS Kato et Yamasaki (2008d) | EDHS |
| RUN Regnier *et al.* (2011) | RUN |

Table 7.3 – Liste des ordonnanceurs implémentés dans SimSo

l'utilisation globale du système doivent être fixés. Bini et Buttazzo (2005) ont montré que la génération de ces ensembles peut introduire un biais dans les évaluations dans le cas monoprocesseur et par extension pour le multiprocesseur.

Pour répondre à ce problème, plusieurs méthodes de génération d'ensembles de tâches ont été implémentées dans SimSo. La première est celle proposée par Kato et Yamasaki (2008b) qui consiste à ajouter des tâches jusqu'à atteindre la charge processeur demandée. Le nombre de tâches est ainsi variable. Les autres se basent sur les algorithmes *UUniFast-Discard* et *RandFixedSum* et permettent de générer un nombre de tâches fixe pour un facteur d'utilisation donné. D'après les études menées dans (Emberson *et al.*, 2010), ces méthodes introduisent un faible biais dans les évaluations des politiques d'ordonnancement.

Ces algorithmes génèrent un ensemble de tâches et doivent être combinés à un générateur de période. Pour cela, plusieurs algorithmes sont aussi mis à disposition dans SimSo :

— *Distribution uniforme pour des intervalles de valeurs :* La plupart des évaluations utilisent cette approche pour étudier l'influence des périodes même si ce tirage n'est pas représentatif des systèmes réels ;

— *log-uniforme (Davis et Burns, 2009) :* Pour une période sur un intervalle 1-1000 ms, la génération log-uniforme des périodes produit un nombre égal de tâches dans chaque intervalle 1-10, 10-100 et 100-1000. D'après Davis et Burns (2009), ces tirages sont représentatif des applications réelles ;

— *Réduction de l'hyper-période (Goossens et Macq, 2001) :* Les périodes sont tirées comme le produit d'un ensemble limité de nombres premiers afin de réduire l'hyper-période du système. Ce tirage présente aussi, d'après ces auteurs, l'avantage de produite des périodes réalistes ;

— *Tirage aléatoire parmi un ensemble de valeurs :* Les périodes des tâches sont simplement tirées parmi un ensemble de valeurs fixées. Ce tirage ce veut proche d'une application réelle où les valeurs des périodes dérivent des spécifications.

Une interface graphique est disponible avec SimSo ce qui permet de générer rapidement des systèmes de tâches et de visualiser leur trace en fonction d'un ordonnanceur. Cependant, son utilisation est à réserver principalement lors de l'écriture d'une nouvelle politique d'ordonnancement pour son débogage. Pour conduire une campagne d'évaluation sur un ensemble de politiques et un ensemble de systèmes de tâches, il est préférable d'utiliser SimSo comme un module Python et d'y faire appel à partir d'un script pour automatiser la création des systèmes, leur simulation et la collecte des données.

Pendant la simulation d'un système, tous les événements sont enregistrés pour ensuite post-traiter ces données et calculer les métriques d'évaluation. Ainsi, il n'est pas nécessaire de modifier le code du simulateur pour ajouter de nouvelles métriques. Par contre, pour faciliter l'utilisation de SimSo, des calculs de métriques classiques sont fournis, parmi lesquels :

— *Taux de succès :* le ratio entre le nombre de travaux qui n'ont pas respecté leur échéance par rapport au nombre de travaux total. Cette valeur donne un indicateur de performance sur l'ordonnaçabilité du système de tâches.

— *Nombre de préemptions et migrations :* une distinction est faite entre migration d'un travail et la migration d'une tâche, car elles peuvent avoir des impacts différents sur les durées d'exécution.

— *Nombre d'appels à l'ordonnanceur :* certaines politiques d'ordonnancement sont connues pour prendre beaucoup de décisions d'ordonnancement, alors que d'autres demandent

beaucoup de temps pour décider. Il est donc intéressant de regarder le nombre d'appel à l'ordonnanceur.

— *Laxité normalisée :* Lelli et al. Lelli *et al.* (2012) proposent de mesurer la performance d'un ordonnanceur en étudiant les laxités des travaux. Cette laxité est calculée comme étant la différence entre l'échéance et le temps de réponse d'un travail et diviser par sa période. Une plus grande laxité est synonyme d'une plus grande réactivité du système.

## 7.3 Étude de performance des politiques d'ordonnancement multicœur

Un grand nombre d'évaluations ont été conduites dans le cadre la thèse de Maxime Chéramy. Ces évaluations ont montré qu'il était possible d'utiliser SimSo pour conduire une large campagne d'évaluation des politiques d'ordonnancement. De plus, toutes ces expérimentations ont été réalisées dans des conditions identiques ce qui autorise les comparaisons entre les algorithmes d'ordonnancement. Nous ne détaillerons pas ici les résultats et laisserons le lecture se référer à (Chéramy, 2014) pour plus de détails. Nous ne retiendrons que les principaux résultats.

**Algorithmes de type G-EDF.** Bien que de nombreux systèmes ne peuvent pas être prouvés ordonnançables avec les tests d'ordonnançabilité de l'algorithme G-EDF, il se trouve qu'en pratique une majorité le sont. Les variantes de cet algorihtme améliorent le taux de systèmes ordonnançables sans pour autant nuire aux performances de l'ordonnancement en termes de préemptions, migrations et temps de réponse. Notons par exemple les bons résultats de G-FL qui ne nécessite qu'un changement trivial de G-EDF au niveau du calcul de la priorité des travaux. Ou encore G-MLLF qui est capable d'ordonnancer un grand nombre de systèmes. G-MLLF provoque cependant une augmentation du nombre de préemptions et migrations, mais qui reste raisonnable pour un nombre faible de tâches ou un nombre important de processeurs.

**Ordonnancement partitionné.** L'algorithme d'affectation *Decreasing First-Fit* permet de partitionner plus d'ensembles de tâches que *Decreasing Worst-Fit*, cependant, ce dernier provoque moins de préemptions pour des systèmes modérément chargés. Par ailleurs, le nombre de systèmes ordonnançables par des approche partitionnées est important, en particulier si le nombre de tâches est grand. Par exemple, il est (très) rare qu'un système de plus de 50 tâches dont le taux d'utilisation est inférieur à 95% ne soit pas correctement ordonnancés. Les algorithmes P-EDF et G-EDF n'ont pas pu être clairement départagés du point de vue du nombre de préemptions et migrations et des temps de réponse, mais P-EDF semble être en mesure d'ordonnancer un plus grand nombre de systèmes, en particulier si la charge est faible ou lorsqu'il y a de nombreux processeurs. Cependant, l'algorithme G-FL présente toujours moins de dépassement d'échéance que P-EDF si le taux d'utilisation est important.

**Politiques DP-Fair.** LLREF engendre un nombre de préemptions plus important que nécessaire en effectuant un tri inutile des travaux à exécuter. Ce problème a été corrigé dans les politiques LRE-TL et DP-WRAP ce qui leur permettent de réduire le nombre de préemptions. Les auteurs de LRE-TL annoncent une réduction du nombre de préemptions et migrations par rapport à LLREF de l'ordre du nombre de tâches. Cependant, nos

résultats indiquent que la réduction est proche du nombre de processeurs et que le nombre de tâches n'influe pas sur les résultats. Les politiques LRE-TL et DP-WRAP utilisent des algorithmes pour la distribution temporelle des dotations très différentes et pourtant elles produisent des nombres de préemptions et migrations très proches. Ces politiques génèrent cependant toujour un nombre important de préemptions et migrations en comparaison des politiques de type G-EDF ou P-EDF.

**Comparaison U-EDF, RUN et EKG.** Les résultats obtenus par Nelissen *et al.* (2012) pour illustrer les bonnes performances de leur algorithme U-EDF en le comparant à RUN et EKG ont été analysés. Les résultats obtenus ne contredisent pas ceux présentés, mais les paramètres expérimentaux dans la publication initiale correspondent à des configurations où EKG est défavorisé ce qui fausse les conclusions.

De manière plus globale, les politiques U-EDF et RUN engendrent peu de préemptions et migrations et donnent de bons temps de réponse, parmi les politiques théoriquement optimales. Les politiques de type G-EDF ainsi que P-EDF sont cependant meilleures sur ces critères, mais ne permettent pas d'ordonnancer des systèmes aussi chargés.

**Utilisation du wcet.** L'utilisation systématique du wcet dans les évaluations concernant le nombre de préemptions, migrations et temps de réponse est discutable. En effet, des politiques comme G-EDF et U-EDF sont capables de tirer profit de durées d'exécution plus faibles que prévues contrairement à des politiques qui se basent beaucoup sur le wcet comme RUN. Les expérimentations ont montré que G-EDF et U-EDF sont capables d'ordonnancer des systèmes pour lesquels la charge totale est estimée au delà des capacités du système, mais avec des charges moyennes autour de 90%. En pratique, ces configurations correspondent à des système réels pour lesquels les wcet sont sur-estimés et ont des exécutions moyennes bien inférieures.

**Politiques conservatives.** Les politiques *work-conserving* ont montré qu'elles permettent de réduire de façon significative le nombre de préemptions et migrations pour des systèmes qui ne sont pas chargés à 100%. De plus, dans la pratique, ces politiques sont capables de tirer profit de durées d'exécution inférieures au wcet ce qui augmente d'autant plus leur attrait. Un travail pour utiliser au mieux les temps d'inactivité des processeurs est nécessaire pour améliorer les performances des politiques RUN et U-EDF.

## 7.4 Bilan et analyse retrospective

Le travail réalisé au cours de la thèse de maxime Chéramy a permis le développement de SimSo, un simulateur d'ordonnancement temps réel pour architecture multiprocesseur. Il a été conçu de manière modulaire pour faciliter ses extensions à de nouveaux modèles de tâches et d'architectures. Une attention toute particulière a été portée au contrôle de la durée d'exécution des travaux afin d'étudier la variété des comportement des systèmes ordonnancés.

L'interface de programmation pour la mise en œuvre d'un ordonnanceur a été pensée pour être réaliste en se basant sur des éléments disponibles dans un vrai système, mais tout en restant simple. Plus de vingt-cinq algorithmes d'ordonnancement ont été mis en œuvre

pour SimSo, montrant que l'interface est assez souple pour permettre l'implémentation d'algorithmes aux approches variées (global, partitionné, semi-partitionné, etc).

Les expérimentations menées ont montré l'utilisabilité de SimSo et permis une meilleure compréhension des politiques existantes. Les résultats obtenus confortent ceux existants dans la littérature tout en relativisant certains. Les politiques algorithmiquement simples et basées sur EDF présentent de très bonnes performances.

Ce travail a été l'occasion de comprendre en profondeur les politiques d'ordonnancement multiprocesseur et de se confronter à la reproductibilité de leur évaluation. Cette problématique dépasse le cadre des politiques d'ordonnancement et devrait être abordée plus sérieusement pour fournir à la communauté temps réel des moyens d'évaluation et de comparaison qui soient facilement reproductibles et outillés.

Maintenir une activité autour de SimSo semble nécessaire pour l'imposer comme un outil de référence dans la communauté temps réel et ainsi le faire vivre. À l'heure actuelle, SimSo est employé par des chercheurs et des étudiants extérieur au projet initial. La facilité de prise en main de l'outil, son aspect modulaire lui permettant de s'adapter à de nouveaux besoins et la bibliothèque d'ordonnanceurs disponibles sont certainement les aspects qui ont été les plus appréciés.

SimSo a aussi été employé à l'INSA de Toulouse et à l'École Centrale de Nantes dans la cadre d'enseignements sur les systèmes temps réel. L'interface graphique et la facilité pour paramétrer les systèmes de tâches permet aux étudiants de rapidement comprendre le comportement d'une politique d'ordonnancement et l'influence des différents paramètres. La version web de son interface graphique a aussi été perçu comme un plus.

CHAPITRE 8

# Prise en compte des incertitudes dans l'analyses d'ordonnançabilité

Ce chapitre présente deux études qui traitent de l'impact sur l'ordonnancement des incertitudes sur les durées d'exécution des tâches. Le premier travail aborde l'analyse d'ordonnançabilité de systèmes avec des modèles stochastiques pour les durées d'exécution. L'étude réalisée a conduit au développement d'une méthode d'échantillonnage pour accélérer le calcul du pire temps de réponse. Cela s'est traduit par une publication (Refaat et Hladik, 2010) suite au travail de Khaled Refaat et par l'encadrement de deux stages de Master, celui de Vikas Shukla sur l'outil Stochan (López *et al.*, 2008) et celui de Larissa Calsavara sur la modélisation et la vérification de réseaux de Petri stochastiques à l'aide de l'outil Oris (Sassoli et Vicario, 2006).

Le second travail présenté dans ce chapitre consiste en l'introduction dans un simulateur d'ordonnancement temps réel d'un modèle de temps qui prend en considération les perturbations induites par les mémoires caches qui sont partagées. Pour cela, le modèle de comportement des tâches a été étendu pour qualifier et quantifier les accès aux caches. Différentes méthodes d'estimation des taux de succès d'accès aux caches ont été ensuite évaluées avant d'être intégrées dans le simulateur SimSo (voir chapitre 7). Cette étude a été réalisée dans le cadre de la thèse de Maxime Chéramy au cours du projet ANR RESPECTED. Deux publications y sont rattachées (Chéramy *et al.*, 2013; Chémary *et al.*, 2013).

## 8.1 Ordonnancement temps réel et variabilité des durées d'exécution des tâches

Estimer la durée d'exécution du travail d'une tâche présente de nombreuses difficultés et est sujet à de multiples sources d'incertitudes. Par exemple, les données en entrée et l'état de l'architecture d'exécution au démarrage d'un travail ont une influence importante sur son exécution. Or l'état du système est généralement impossible à connaître et la durée d'exécution est difficilement caractérisable en tant que fonction des données en entrée.

Dans le cadre de la théorie de l'ordonnancement temps réel, la variabilité des durées d'exécution des tâches est souvent modélisée par l'estimation d'une borne inférieure (*best-case execution time*, bcet) et une supérieure (*worst-case execution time*, wcet). Cependant,

l'utilisation de ces bornes dans les analyses d'ordonnançabilité introduit du pessimisme et indirectement une surestimation des ressources nécessaires pour ordonnancer le système. Or, pour certaines applications, une prédiction stochastique du comportement temporel des tâches peut être suffisante pour garantir les exigences du système. Une manière de répondre à ce besoin est de modéliser les temps d'exécution par des variables aléatoires discrètes et de calculer par une analyse d'ordonnançabilité adpatée la probabilité de respect des échéances.

Ces analyses s'inspirent des méthodes classiques, par exemple en estimant le pire temps de réponse. Cependant, le coût calculatoire des méthodes avec un modèle stochastique est souvent élevé. Pour contourner ce problème, plusieurs solutions ont été proposées, telles que des algorithmes d'ordonnancement spécifiques pour limiter les interactions entre les tâches (Atlas et Bestavros, 1998; Abeni et Buttazzo, 2001), ou l'introduction d'hypothèses pour simplifier l'analyse au détriment de la précisions des résultats (Manolache *et al.*, 2007). Le travail présenté dans la partie 8.2 de ce chapitre s'inscrit dans cette démarche et propose une méthode pour réduire la complexité calculatoire en introduisant un échantillonnage sur les durées d'exécution.

Dans le cas d'une architecture multiprocesseur, les durées d'exécution sont aussi sensibles aux contentions sur les ressources partagées dues aux exécutions parallèles. Une des sources de ces contentions est l'utilisation des mémoires caches partagées entre les cœurs. La seconde partie de ce chapitre présente un ensemble de modèles stochastiques pour évaluer la durée d'exécution des tâches en tenant compte des caches. Les méthodes retenues ne sont pas issues du domaine temps réel, mais des travaux sur l'estimation des performances des programmes et des architectures matérielles. Nous verrons que ces modèles peuvent être utilisés pour simuler un système ordonnancé et ainsi étudier l'influence des mémoires caches sur l'ordonnancement.

## 8.2 Analyse d'ordonnançabilité échantillonnée pour des durées d'exécution stochastiques

Cette première partie repose sur les travaux de Diaz *et al.* (2004) pour calculer la distribution du temps de réponse d'un système de tâches pour des ordonnanceurs à priorités fixes avec des durées d'exécution modélisées par des variables aléatoires discrètes. Des probabilités sur les paramètres d'ordonnancement, telle que le respect des échéances, sont ainsi évaluées. Cependant, ces calculs restent coûteux en temps et en mémoire et ne permettent pas de traiter de grands systèmes de tâches. Le travail exposé ci-après et publié dans (Refaat et Hladik, 2010) permet de simplifier l'analyse de Diaz *et al.* (2004) grace à un échantillonnage de la distribution des durées d'exécution tout en garantissant le pessimisme de l'analyse.

### 8.2.1 Modèle de tâches et calcul du temps de réponse stochastique

Le modèle utilisé dans (Diaz *et al.*, 2004) est composé d'un ensemble $\mathcal{T} = \{1...N\}$ de $N$ tâches périodiques. Chaque tâche $i$ est caractérisée par sa période $p_i$, son offset initial $o_i$, son temps d'exécution $e_i$, son échéance relative $d_i$ et sa probabilité maximum de dépassement d'échéance $m_i$. Seul le temps d'exécution $e_i$ est une variable aléatoire discrète. Sa distribution est supposée connue.

La tâche étant périodique, un nombre infini de travaux en résulte. Le travail $j$ de la tâche $i$ est activé à la date $a_{i,j}$ avec $a_{i,j} = o_i + (j-1)p_i$ et son temps de réponse est une variable aléatoire discrète notée $r_{i,j}$.

Le temps de réponse de la tâche $i$ est calculé comme étant le temps moyen des temps de réponse de ses travaux (Diaz *et al.*, 2004) :

$$f_{r_i}(x) = \frac{1}{m_i} \sum_{j=1}^{m_i} f_{r_{i,j}}(x) \tag{8.1}$$

avec $m_i = \frac{p}{p_i}$ le nombre de travaux de la tâche $i$ activés sur une hyper-période de taille $p = \text{ppcm}\{p_i\}$ et $f_X(x) = P\{X = x\}$, la fonction de masse de la variable aléatoire $X$.

Dans le cas de l'analyse stochastique, une tâche $i$ est dite ordonnançable si la probabilité que son temps de réponse soit supérieur à son échéance est plus petite que sa probabilité maximum de dépassement d'échéance, soit $P\{r_i > d_i\} < m_i$.

Le temps de réponse d'un travail $j$ de la tâche $i$ s'obtient à l'aide de l'équation suivante :

$$r_{i,j} = W(a_{i,j}) * e_i * J_{i,j} \tag{8.2}$$

avec $W(a_{i,j})$ le travail en retard à l'instant $a_{i,j}$ et $J_{i,j}$ l'interférence sur le travail $j$ causée par les travaux des tâches plus prioritaires après la date $a_{i,j}$. Le symbole $*$ est utilisé pour le produit de convolution entre deux fonctions, soit dans le cas discret $(f * g)(n) = \sum_{m=-\infty}^{\infty} f(n-m)g(m)$.

Le calcul de l'équation (8.2) peut être réalisé de manière itérative. Pour cela, notons $\Lambda_{i,j} = \{\lambda_0, \lambda_1, ...\}$ l'ensemble des dates d'activations des travaux plus prioritaires que la tâche $i$ et des travaux $n$ de la tâche $i$ tels que $n \leq j$. L'ensemble $\Lambda_{i,j}$ est ordonné de manière croissante. Nous identifions par l'indice $k$ la valeur $a_{i,j}$ dans l'ensemble $\Lambda_{i,j}$, soit $\lambda_k = a_{i,j}$.

Le travail restant à traiter lors de l'activation du travail $j$, c'est-à-dire à l'instant $\lambda_k = a_{i,j}$, est calculé itérativement pour $n < k$ à l'aide des relations suivantes :

$$f_{W_0}(0) = 1 \tag{8.3}$$

$$W_{n+1} = shrink\left(W_n * e_n, \lambda_{n+1} - \lambda_n\right) \tag{8.4}$$

avec la fonction *shrink* définie par :

$$f_{shrink(W,\Delta)}(x) = \begin{cases} 0 & \text{si } x < 0, \\ \sum_{z=-\infty}^{0} f_W(z+\Delta) & \text{si } x = 0, \\ f_W(x+\Delta) & \text{if } x > 0. \end{cases} \tag{8.5}$$

La fonction *shrink* estime la loi de probabilité d'une nouvelle variable aléatoire à partir d'une autre variable aléatoire $W$ telle que la probabilité de la valeur 0 soit égale à la somme des probabilités des valeurs de $W$ qui sont inférieures à $\Delta$ et en décalant les valeurs de $\Delta$ tout en conservant la loi de probabilité de $W$. Autrement dit, la fonction *shrink* « coupe » la fonction de répartition de $W$ pour les valeurs inférieures à $\Delta$ et se « recale » sur la valeur 0.

Le calcul itératif décrit par l'équation (8.4) permet de calculer l'interférence due aux tâches activées avant $a_{i,j}$. En convoluant cette valeur avec la distribution du temps de réponse de la tâche $i$, soit $W(a_{i,j}) * e_i$, il est possible de calculer la distribution du temps de réponse

du travail $j$ sans considérer les activations des travaux plus prioritaires qui sont activés après $a_{i,j}$.

Pour évaluer les interférences dues aux tâches plus prioritaires activées après $a_{i,j}$, le calcul itératif suivant est appliqué pour $n \geq k$ et tant que $\lambda_n \leq \lambda_k + d_i$ :

$$r(\lambda_{n+1}) \quad = \quad AF\left(R(\lambda_n), \lambda_n - \lambda_k, e_n\right) \tag{8.6}$$

avec $e_n$ la durée d'exécution de la tâche associée à la date $\lambda_n$ et $AF$ définie par

$$f_{AF(r,\Delta,e)}(x) = \begin{cases} f_r(x) & \text{if x} \leq \Delta \\ \sum_{i=\Delta+1}^{\infty} f_r(i).f_e(x-i) & \text{if } x > \Delta \end{cases} \tag{8.7}$$

La fonction $AF$ conserve les probabilités de la variable $r$ pour les valeurs comprise entre 0 et $\Delta$ et procède ensuite à la convolution entre $r$ et la fonction de masse de $e$ en décalant les valeur de $\Delta$. Autrement dit, elle intègre dans la fonction de masse $r$, le temps d'exécution $e$ d'un travail en prenant en considération le décalage induit par sa date d'activation.

L'équation (8.6) permet de calculer la fonction de masse du temps de réponse du travail $j$ de la tâche $i$. Ensuite l'équation (8.1) est appliquée pour évaluer celle de la tâche $i$. La probabilité de dépasser l'échéance est alors simplement évaluer en sommant les probabilités des valeurs du temps de réponse plus petites que l'échéance et en soustrayant le résultat à 1, soit

$$P\{r_i > d_i\} = 1 - \sum_{k=0}^{k=d_i} P\{r_i = k\} \tag{8.8}$$

### 8.2.2 Méthode d'échantillonnage

Le calcul présenté ci-dessus peut être extrêmement couteux en terme de temps et de mémoire. Sa complexité dépend en grande partie de la taille des ensembles des valeurs décrivant les durées d'exécution. Un moyen pour éviter ce problème est de procéder à un échantillonnage des valeurs qui préserve le pessimisme des calculs.

En statistique, l'échantillonnage consiste à sélectionner des individus afin d'avoir une connaissance sur l'ensemble de la population. L'échantillonnage probabiliste est un processus d'échantillonnage dans lequel la probabilité d'occurrence d'un individu est pris en considération, c'est-à-dire qu'un individu avec une haute probabilité aura une plus grande chance d'être sélectionné au cours de l'échantillonnage qu'un individu avec une faible probabilité.

Pour l'analyse stochastique d'ordonnançabilité, chaque durée d'exécution d'une tâche sera échantillonnée en sélectionnant $k$ valeurs parmi les $N$ valeurs de la distribution puis en ajoutanr la somme des probabilités restantes à la pire des valeurs de temps d'exécution (voir figure 8.1).

La nouvelle distribution $e'$ après échantillonnage de $k$ éléments dans $e$ a pour forme

$$f_{e'}(x) = \sum_{i=1}^{k} f_e(x_i)\delta(x - x_i) + \left(1 - \sum_{i=1}^{k} f_e(x_i)\right)\delta(x - x_w) \tag{8.9}$$

avec $x_i$ la valeur du temps d'exécution pour le $i$ème échantillon, $x_w$ la valeur du pire temps d'exécution, $f_e$ la fonction de masse originale et $\delta$ définie par :

$$\delta(x) = \begin{cases} 1 & \text{si } x = 0 \\ 0 & \text{si } x \neq 0 \end{cases} \tag{8.10}$$

,

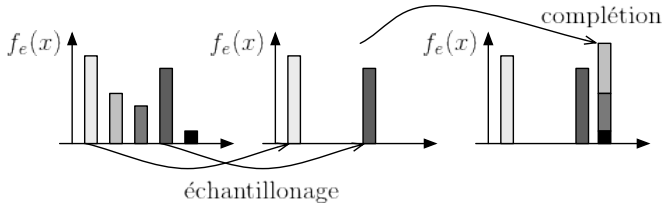



FIGURE 8.1 – Exemple d'échantillonnage avec $k = 2$. Deux échantillons sont sélectionnés, les autres étant cumulés à la fin de la distribution.

Le premier terme de l'équation (8.9) représente la contribution des $k$ échantillons alors que le second est ajouté pour assurer le pessimisme de la méthode.

Pour une variable discrète $X$ pouvant prendre les valeurs $\{x_0, ..., x_n\}$, une manière d'implémenter le choix des échantillons est de tirer une valeur aléatoire dans l'intervalle $(0, 1)$ puis de sélectionner la valeur $x_0$ si $U < f_X(x_0)$ ou $x_i$ si $\sum_{j=0}^{i-1} f_X(x_j) \leq U < \sum_{j=0}^{i} f_X(x_j)$. On procède ainsi $k$ fois en excluant la possibilité de sélectionner deux fois la même valeur.

Le nombre d'échantillons $k$ est un paramètre qui peut être fixé pour régler le temps de calcul et limiter l'utilisation de la mémoire consommée. Par contre, cette valeur a aussi un impact sur le pessimisme introduit dans l'évaluation des pires temps de réponse. Trouver un bon compromis entre la performance des calculs et le pessimisme est donc à faire.

### 8.2.3 Propriétés de l'échantillonnage

Pour prouver que la nouvelle distribution garantit le pessimisme, la relation d'ordre entre deux distributions introduite par Diaz *et al.* (2004) est utilisée :

**Définition 8.1.** *(Diaz* et al.*, 2004) Soit deux variables aléatoires discrètes $X$ et $Y$, « $X$ est dite pire que $Y$ » et notée $X \succeq Y$ ssi $F_X(x) \leq F_Y(x)$ pour tous $x$, avec $F_X(x) = P\{X \leq x\}$, la fonction de distribution cumulative.*

Cette définition permet ensuite d'introduire une relation d'ordre entre des systèmes de tâches :

**Théorème 8.1.** *(Diaz* et al.*, 2004) Soit deux systèmes de tâches $S$ et $S'$ avec des paramètres identiques sauf pour le travail $j$ de la tâche $i$ dont le temps d'exécution est $e$ dans le système $S$ et $e'$ dans le système $S'$. Si $e' \succeq e$ alors les temps de réponse de tous les travaux calculés par l'équation* (8.2) *ont pour relation $r' \succeq r$.*

Or, par définition de $e'$ par l'équation (8.9) nous avons $e' \succeq e$ et donc :

**Théorème 8.2.** *Soit $S'$ un système de tâches obtenu après échantillonnage du système $S$, alors les temps de réponse de tous les travaux calculés par l'équation* (8.2) *ont pour relation $r' \succeq r$.*

### 8.2.4 Expérimentations

Les expérimentations publiées dans (Refaat et Hladik, 2010) présentent de nombreuses erreurs détectées a posterio. Les présenter ici n'aurait aucun intérêt. Il est cependant à remarquer que les travaux de Maxim *et al.* (2012) étendent le concept introduit ici et

montrent clairement l'intérêt de cette approche. Dans ce papier trois méthodes d'échantillonnage sont comparées et les expérimentations montrent la réduction effective du coût calculatoire et que le pessimisme est bien conservé.

## 8.3 Étude de l'impact du cache sur l'ordonnancement

Dans le cadre de la thèse de Maxime Chéramy (2014), une étude a été réalisée pour évaluer l'influence des mémoires caches sur un système de tâches en fonction des politiques d'ordonnancement. Pour cela il a fallu enrichir le modèle classique d'exécution des tâches avec des modèles capables de prendre en considération le cache, puis de les intégrer dans SimSo (voir la présentation de ce simulateur dans la partie 7).

Plusieurs modèles statistiques ont été sélectionnés pour évaluer la durée d'exécution des tâches en tenant compte des mémoires caches. Les méthodes retenues proviennent des recherches sur l'évaluation des performances des architectures matérielles. Elles ne sont donc pas issues du domaine temps réel et leur but est d'approximer le nombre de défauts de cache et non pas d'en prédire le comportement au pire.

Seuls les modèles, les conclusions des différentes expérimentations et leur couplage avec SimSo sont présentés dans ce chapitre. Un lecteur intéressé par une description complète des travaux peut consulter (Chéramy, 2014).

### 8.3.1 Modèles de tâche pour représenter les accès au cache

Les travaux ont été limités aux caches associatifs avec l'algorithme de remplacement LRU. Seuls les caches de données ont été considérés et le coût lié au chargement des instructions est pris en compte dans le CPI de base (voir ci-après). Il est supposé qu'il existe un cache d'instructions dédié. L'architecture des caches est hiérarchique et inclusive. Dans le cas de caches inclusifs, les données présentes dans le cache de premier niveau sont également présentes dans le cache situé au niveau supérieur.

Les coûts liés aux protocoles de cohérence ont été négligés. Il est cependant important de souligner que ce coût devient de moins en moins négligeable avec l'augmentation du nombre de cœurs partageant un unique cache. Ainsi les travaux réalisés se limitent à des architectures dotées de quatre cœurs au maximum.

**Modèle de cache.** Les caches sont organisés de manière hiérarchique ce qui est représenté sous la forme d'une liste ordonnée associée à chaque processeur. Les caches peuvent être partagés entre plusieurs processeurs à condition de respecter la propriété d'inclusion des caches.

Un cache est défini par une taille $C$ (en lignes), son associativité $A$ et la pénalité temporelle associée à un défaut de cache. La pénalité $pm_{Lx}$ pour un cache $Lx$ correspond au temps nécessaire pour accéder au niveau de cache supérieur. Le dernier niveau est la mémoire du système. Le temps nécessaire pour accéder au premier niveau de cache est noté $pm_0$.

**Modèle de tâche.** Le modèle classique de tâche de Liu et Layland (1973) est étendu par des informations pour caractériser les accès aux mémoires cache. Ainsi, une tâche est définie par sa période et son échéance relative et par :

— son nombre d'instructions ($n$) : cette valeur représente le nombre d'instructions exécutées par un travail. Ici, ce nombre est fixe, mais pourrait être remplacé par un tirage aléatoire, selon une distribution donnée, pour caractériser la variabilité des exécutions.

— son nombre moyen de cycles par instuctions (*cpi* et *base_cpi*) : Le nombre moyen de cycles par instruction (CPI) d'une programme est un indicateur de performance. Plus le CPI est faible et plus le programme s'exécute rapidement (Mogul et Borg, 1991). Le nombre moyen de cycles nécessaires pour exécuter une instruction sans considérer les pénalités liées aux accès mémoire est noté *base_cpi*.

— son taux d'accès mémoire ($API$) : ce taux représente la proportion des instructions d'un programme qui font un accès mémoire en lecture ou en écriture.

— son profil d'accès mémoire (*sdp*) : le moyen retenu pour décrire les accès au cache est le *Stack Distance Profile* (SDP) (Mattson *et al.*, 1970) d'un programme. Cette valeur est une distribution discrète (valeur-probabilité) dont la valeur représente le nombre de lignes de cache différentes accédées entre deux accès consécutifs à une même ligne et dont la probabilité est la moyenne des accès au cache ayant cette distance par rapport à tous les accès du programme. La figure 8.2 illustre cette notion.

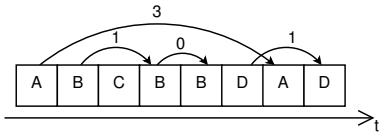


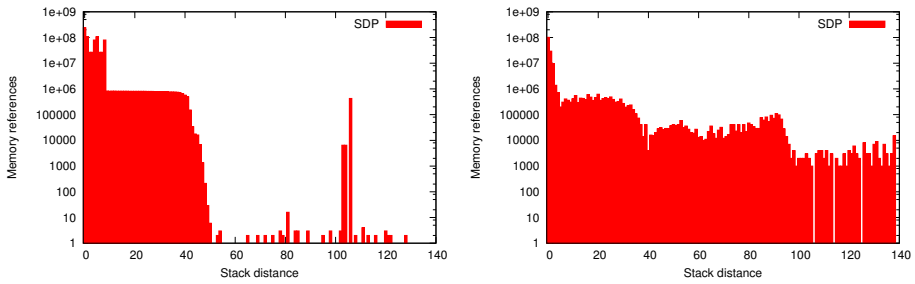

FIGURE 8.2 – Séquence d'accès mémoire. A, B, C et D sont des lignes de cache et les nombres indiquent les distances. La probabilité associée à chaque distance est dans cet exemple de 1/4 pour la distance 0, 1/2 pour la distance 1 et 1/4 pour la distance 3.

### 8.3.2 Collecte des données d'entrée des modèles

Afin d'obtenir des données réalistes pour caractériser le comportement des tâches par rapport au cache, des mesures ont été réalisées sur divers programmes choisis parmi les suites de benchmarks MiBench (Guthaus *et al.*, 2001) et Mälardalen (Gustafsson *et al.*, 2010).

Le nombre d'instructions ($n$), le taux d'accès mémoire ($API$), le CPI (*base_cpi*) et le profil d'accès mémoire (*sdp*) ont été a été collectés en utilisant le simulateur gem5 (Binkert *et al.*, 2011) puis en post-traitant la trace des accès mémoire. La figure 8.3 donne un aperçu des mesures de deux *sdp* obtenus à partir de la suite MiBench.

Au total, une trentaine de programmes ont été sélectionnés et analysés pour servir de bibliothèque de référence aux études sur les modèles d'accès au cache. Les premières analyses ont montré que les SDP étaient extrêmement variés et qu'ils ne suivent pas une une distribution classique (contrairement aux hypothèses faites dans certains travaux). Sur certaines distributions, il a été observé des plateaux et des pics ce qui est synonyme de variations brusques du nombre de défauts de cache lors de variations de la taille du cache (taille physique ou effective à cause d'un partage) et donc une sensibilité importante des programmes à l'architecture. Par exemple, le SDP du programme CRC représenté sur la figure 8.3(a) montre une brusque rupture autour de 50 lignes, et donc un cache avec moins de lignes provoquera peut d'échec.




(a) CRC (b) UD

FIGURE 8.3 – SDP calculée sur les application (a) CRC et (b) UD de MiBench.

### 8.3.3 Modèles de taux de défauts d'accès au cache

L'estimation de la durée d'exécution d'un travail peut se faire en calculant le CPI en fonction du taux de défauts de cache pour chaque niveau de cache. Pour évaluer cette valeur plusieurs modèles ont été proposés par la communauté scientifique travaillant sur l'évaluation de performances des architecture. Nous utilisons ensuite ce CPI dans la simulation pour estimer, pour un intervalle de temps, le nombre d'instructions exécutées par une tâche. Les sections suivantes présentent des modèles permettant d'évaluer le taux de défauts de cache pour une tâche seule, un cache partagé, et suite à une préemption ou migration.

#### Estimation du nombre de cycles par instruction

Le modèle de CPI utilisé est celui étudié par Hennessy et Patterson (2011) et Eklov *et al.* (2011). Pour rappel, le *base_cpi* d'une tâche est le CPI sans les pénalités liées aux accès mémoires et $pm_{Lx}$ la pénalité moyenne pour un défaut de cache au $L_x$. Le CPI pour une tâche $\tau$ est défini par :

$$cpi_\tau = base\_cpi_\tau + API_\tau \cdot P_\tau \tag{8.11}$$

avec $P_\tau$ la pénalité moyenne associée à un accès mémoire :

$$P_\tau = pm_0 + \sum_{x=1}^{X} mr_{\tau,Lx} \cdot pm_{Lx} \tag{8.12}$$

où $mr_{\tau,Lx}$ est le taux de défaut du cache $Lx$ pour la tâche $\tau$.

À partir du nombre d'instructions, $n_\tau$, et du CPI, il est possible d'en déduire une estimation de la durée d'exécution de la tâche $\tau$ :

$$C_\tau = cpi_\tau \cdot n_\tau \tag{8.13}$$

À l'inverse pour une durée $\delta$, le nombre d'instructions réalisées par une tâche peut être estimée à $n_\tau = cpi_\tau / \delta$.

#### Défauts de cache pour une tâche seule

Pour un cache LRU, tous les accès mémoire se faisant à une *stack distance* supérieure à la taille du cache entraîne un défaut de cache. Par conséquent, le taux de défaut de cache

pour une tâche $\tau$ peut s'exprimer à partir du SDP par :

$$mr_{\tau,L_x} = 1 - \sum_{i=0}^{A_{L_x}-1} sdp_\tau(i) \tag{8.14}$$

avec $A_{L_x}$ l'associativité du cache $L_x$.

Cette équation suppose l'utilisation d'un SDP calculé avec un nombre d'ensembles identique à celui du cache utilisé. Malheureusement, ceci implique de disposer d'un SDP pour tous les ensembles possibles ce qui pose des difficultés dans le contexte de la simulation. Nous pouvons donc, soit calculer un nouveau SDP pour chaque nombre d'ensembles, soit nous limiter au calcul pour des caches entièrement associatifs (SDP calculé pour un seul ensemble) et procéder ensuite à une estimation en fonction du nombre d'ensembles. Dans les deux cas, ceci engendre une erreur et l'équation (8.14) devient :

$$mr_{\tau,Lx} = 1 - \sum_{i=0}^{C_{Lx}-1} sdp(i) \tag{8.15}$$

avec $C_{Lx}$ la taille du cache $L_x$ en nombre de lignes.

En utilisant cette dernière équation avec les équations (8.12), (8.11) et (8.13) pour le calcul du CPI, nous pouvons en déduire le nombre d'instructions exécutées sur un intervalle de temps.

Ces modèles ont été testés expérimentalement et ont montré que l'utilisation d'un SDP différent pour chaque taille de cache permet de déterminer précisément le nombre de défauts de cache. Cependant, le calcul d'un SDP nécessite la trace des accès mémoire d'un programme (jusqu'à plusieurs giga par trace pour des programmes pourtant petits) et il faut du temps pour calculer le SDP (entre quelques secondes et plusieurs heures en fonction de la distance moyenne des accès et de leur nombre). Or, l'un des objectifs des expérimentations avec le simulateur est de pouvoir modifier facilement les caractéristiques des caches ce qui rend impossible l'utilisation de SDP différenciés.

Les expérimentations ont aussi montré qu'utiliser un unique SDP pour toutes les tailles de cache induit une erreur dans l'estimation, mais qu'elle reste relativement faible. Cela ne pose donc pas de problème pour la simulation d'ordonnancement dont l'objectif n'est pas de reproduire fidèlement le comportement d'un programme, mais simplement les tendances. De plus, ce qui importe réellement est l'estimation des durées d'exécution des tâches et non le nombre de défauts de cache. Les expérimentations ont montré que les erreurs dans l'estimation du nombre de défauts de cache ont un impact réduit dans le calcul de la durée d'exécution des programmes.

#### Défauts de cache pour une tâche avec un cache partagé

Lorsque plusieurs tâches s'exécutent en parallèle sur différents processeurs qui partagent des caches, ceux-ci ne sont généralement pas partagés de manière équitable. En effet, une tâche qui fait beaucoup d'accès mémoire aura tendance à accaparer une grande partie du cache en évitant à ses lignes d'en être exclues.

De nombreux modèles existent pour traiter ce cas. Les études que nous avons menées ont essentiellement portée sur trois modèles : FOA (Chandra *et al.*, 2005), SDC (Chandra *et al.*, 2005) et Babka (Babka *et al.*, 2012). Le choix de ces modèles a été fait en considérant leur complexité calculatoire et les résultats affichés dans les publications.

Nous n'exposerons pas ici les calculs propre aux modèles, ni les expérimentations qui ont été menées pour les comparer. Celles-ci ont montré que les erreurs induites par le modèle SDC sont très importantes et que les modèles FOA et Babka ont des performances similaires, mais que le calcul de Babka est beaucoup plus couteux. Ainsi, pour la simulation, c'est le modèle FOA qui a été retenu. Pour les deux modèles, même s'il y a des erreurs dans l'estimation, nous constatons que les effets des caches sur le CPI sont reproduits de manière fidèle.

#### Défauts de cache suite à une préemption

Pour évaluer la perturbation dans le cache suite à une préemption, il est nécessaire d'estimer le nombre de lignes appartenant à un programme qui sont présentes dans le cache avant l'interruption et le nombre de lignes présentes au moment de la reprise. Cette différence quantifie la perturbation que le cache subit et permet ainsi d'estimer le nombre de défauts de cache supplémentaires causés par la préemption et qui n'auraient pas eu lieu sinon.

Afin d'estimer le nombre de lignes présentes dans le cache avant et après la préemption, il est nécessaire d'évaluer le nombre de lignes utilisées par un programme en fonction du nombre d'accès mémoire (que nous connaissons à l'aide de API, CPI et de la durée d'exécution). Pour cela, plusieurs modèles ont été proposé. Le premier représente le chargement du cache par une loi exponentielle, mais s'est révélé en pratique inutilisable car la distribution dans le temps des *cold misses* (c'est-à-dire des lignes qui n'ont jamais encore été chargées) varie beaucoup d'une tâche à une autre. Deux modèles ont ensuite été étudiés, un premier basé sur les travaux de Babka *et al.* (2012) et le second directement issu de (Liu *et al.*, 2008). Nous ne détaillerons pas les modèles car lors des expérimentations plusieurs problèmes sont apparus :

— les modèles permettent d'évaluer avec une bonne précision le chargement moyen pour la plupart des programmes, mais le point de la préemption dans le programme a une influence importante sur le temps de chargement du cache, introduisant une très grande variabilité qu'il n'est pas possible de reproduire avec les seules informations du modèle de tâche utilisé,

— évaluer le nombre de défauts de cache moyen suite à une préemption en fonction de la durée de la perturbation n'a pas été possible.

L'ensemble de ces problèmes n'a pas permis de mettre en place une méthode pour évaluer le coût des préemptions pendant la simulation. Il a donc été décidé d'utiliser une pénalité fixe pour chaque programme. La valeur moyenne de cette pénalité pouvant être estimée en fonction de la taille des caches et du SDP. Il faut cependant remarquer que la perturbation induite par une préemption est généralement très faible au regard de la durée d'exécution d'un programme.

### 8.3.4 Intégration des modèles dans SimSo

Le simulateur d'ordonnancement SimSo a été présenté dans le chapitre précédent (chapitre 7). Comme nous l'avons vu, il a été développé pour permettre l'ajout de pénalités temporelles lors de certains évènements liés à l'ordonnancement. Ces pénalités sont prises en compte au niveau des objets *Processor* qui simulent l'exécution des travaux et le fonctionnement du système d'exploitation. Par exemple, si un processeur appelle une méthode de l'ordonnanceur, une attente sera ajoutée pour simuler le délai introduit par la décision d'ordonnancement et donc empêcher ce processeur de continuer à traiter des travaux.

Le modèle FOA a été utilisé pour simuler le partage de cache entre plusieurs programmes qui s'exécutent simultanément et des pénalités fixes ont été ajoutées pour les préemptions et les migrations. Pour cela il a été nécessaire d'enrichir le modèle des tâches avec le nombre moyen de cycles par instruction sans prendre en compte les pénalités liés aux accès mémoire (*base_cpi*) ; le nombre d'instructions exécutées par les travaux de la tâche ($n$) ; la proportion d'accès mémoire par instruction ($API$) ; le profil des accès mémoire pour un cache entièrement associatif (*sdp*) ; et le coût temporel d'une préemption ou migration. Ces caractéristiques sont fixées par l'expérimentateur et peuvent être issues de vrais programmes ou être générées de manière artificielle.

La prise en compte des caches se fait en calculant à partir du CPI la durée d'exécution des travaux simulés. L'introduction de ces durées dans SimSo passe par l'utilisation de l'*Execution Time Models* (ETM) qui estime une borne inférieure de la durée d'exécution restante d'un travail en fonction des événements survenants lors de la simulation (voir partie 7.2.2 pour plus de détails).

**Prise en compte des préemptions et des migrations.** Dans l'ETM, un premier dictionnaire *running* a été ajouté qui contient pour chaque processeur le travail en cours d'exécution, et un second dictionnaire *was_running_on* qui contient pour chaque travail le processeur sur lequel il s'est exécuté pour la dernière fois. Un troisième dictionnaire, *penalty*, permet de comptabiliser pour chaque travail la pénalité cumulée engendrée par les préemptions et les migrations.

Lors de l'appel à la méthode *on_execute*, le dictionnaire *was_running_on* est consulté pour savoir si le travail s'est déjà exécuté (on ignore la première exécution). Si c'est le cas le simulateur regarde si le travail est resté sur le même processeur ou s'il a migré. Dans le cas d'une simple préemption, le simulateur examine si un autre travail s'est exécuté sur ce processeur entre-temps. S'il s'agit d'une migration ou s'il s'agit d'une interruption avec exécution d'un autre travail sur le même processeur, alors la pénalité de migration est ajouté au travail.

La pénalité calculée est additionnée au temps d'exécution du travail dans la fonction *get_ret* (voir ci-après). Remarquons que ces pénalités sont considérées comme fixes, mais pourraient être variables en intégrant de nouveaux modèles.

**Prise en compte des caches partagés.** Lorsque les caches sont pris en compte, chaque travail dans SimSo dispose d'un certain nombre d'instructions à exécuter. La durée nécessaire pour l'exécution de ces instructions dépend du CPI du travail. Ce CPI est calculé à l'aide du modèle FOA et à partir des caractéristiques du travail ($API$, *sdp*, *base_cpi*) ainsi que des caractéristiques des caches (taille, pénalité, hiérarchie).

À chaque changement d'état dans le système (exécution ou interruption d'un travail), le CPI de chaque travail est calculé pour l'intervalle défini par le précédent évènement et la date courante. En divisant la durée de l'intervalle par le CPI des travaux, on obtient pour chaque travail le nombre d'instructions exécutées sur cet intervalle. Le nombre d'instructions exécutées depuis le début pour chaque travail est ainsi mis à jour. Pour le calcul de FOA, la liste des travaux en cours d'exécution est nécessaire pour évaluer le partage des caches.

La méthode *on_execute* de l'ETM est complétée par une mise à jour du nombre d'instructions et de la liste des travaux en cours d'exécution. De même, lorsqu'un travail est

préempté (*on_preempted*) ou se termine (*on_terminated*), le nombre d'instructions est mis à jour, puis le travail est retiré de la liste des travaux en cours d'exécution.

La fonction *get_ret* retourne une borne inférieure du temps restant d'exécution. Cette durée est calculée à partir du nombre restant d'instructions à exécuter et du CPI calculé sans partage de cache entre tâches. La pénalité des préemptions et migrations est ensuite ajoutée à cette durée.

## 8.4 Bilan et analyse retrospective

Ces deux travaux ont permis d'étudier l'ordonnancement temps réel avec des hypothèses stochastiques. La première étude propose une optimisation calculatoire pour l'analyse d'ordonnançabilité d'un système avec des durées d'exécution modélisées par des distributions discrètes. Alors que la seconde étude s'est attachée à introduire des modèles stochastiques d'exécution de tâches capables de considérer les caches partagés dans un simulateur d'ordonnancement.

Autant la première étude est un simple approfondissement d'une méthode déjà existante, autant la seconde propose une approche originale. Ce dernier travail a été réalisé en s'appuyant sur les travaux d'une communauté scientifique différentes de celle du temps réel (ici principalement la communauté travaillant sur l'évaluation des performances des architectures matérielles), montrant de nouveau (voir parties 3 et 4) l'intérêt de mettre en place des travaux transverses.

La poursuite du travail avec SimSo est une voie offrant de nombreuses perspectives. La plus simple consiste à exploiter SimSo pour étudier le comportement des ordonnanceurs en considérant les mémoires cache partagées. Une seconde voie est d'exploiter les modèles dans une démarche de conception pour configurer l'ordonnancement en optimisant l'utilisation des mémoires caches. Cette étude permettrait de faire le lien entre le niveau système (ordonnancement) et le niveau de l'architecture d'exécution.

# Bibliographie


Luca Abeni et Giorgio C. Buttazzo : Stochastic analysis of a reservation based system. *In Proc. the 9th workshop on parallel and distributed real-time systems*, 2001.

Tilak Agerwala et Mike Flynn : Comments on capabilities, limitations and "correctness" of Petri nets. *In Proc. of the 1st annual Symposium on Computer architecture*, SCA, 1973.

Ahmad Al Sheikh : *Resource allocation in hard real-time avionic systems : scheduling and routing problems.* Thèse de doctorat, Université de Toulouse, 2011.

Ahmad Al Sheikh, Olivier Brun, Maxime Chéramy et Pierre-Emmanuel Hladik : Optimal design of virtual links in AFDX networks. *Real-Time Systems*, 49(3), 2013.

Ahmad Al Sheikh, Olivier Brun et Pierre-Emmanuel Hladik : Decision support for task mapping on IMA architecture. *In Proc. of the 3rd Junior Researcher Workshop on Real-Time Computing*, JRWRTC, 2009.

Ahmad Al Sheikh, Olivier Brun et Pierre-Emmanuel Hladik : Partition scheduling on an IMA platform with strict periodicity and communication delays. *In Proc. of the 18th International Conference on Real-Time and Network Systems*, RTNS, 2010.

Ahmad Al Sheikh, Olivier Brun et Pierre-Emmanuel Hladik : Ordonnacement de tâches sous contrainte de périodicité stricte. *In Proc. of the 12e Conférence de la société Française de Recherche Opérationnelle et Aide à la Décision*, ROADEF, 2011a.

Ahmad Al Sheikh, Olivier Brun, Pierre-Emmanuel Hladik et Balakrishna J. Prabhu : A best-response algorithm for periodic scheduling. *In Proc. of the 23rd Euromicro Conference on Real-Time Systems*, ECRTS, 2011b.

Ahmad Al Sheikh, Olivier Brun, Pierre-Emmanuel Hladik et Balakrishna J. Prabhu : Strictly periodic scheduling on an IMA-based avionic platform. *In Proc. of the 15th Austrian-French-German conference on Optimisation*, AFG, 2011c.

Ahmad Al Sheikh, Olivier Brun, Pierre-Emmanuel Hladik et Balakrishna J. Prabhu : Strictly periodic scheduling in IMA-based architectures. *Real-Time Systems*, 48(4), 2012.

Sebastian Altmeyer, Robert I. Davis et Claire Maiza : Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems*, 48(5), 2012.

James Anderson, Vasile Bud et Uma Maheswari Devi : An EDF-based scheduling algorithm for multiprocessor soft real-time systems. *In Proc. of the 17th Euromicro Conference on Real-Time Systems*, ECRTS, 2005.

James Anderson, Jeremy P. Erickson, UmaMaheswari Devi et Benjamin Casses : Optimal semi-partitioned scheduling in soft real-time systems. *In Proc. of the 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA, 2014.

James H. Anderson et Anand Srinivasan : A new look at Pfair priorities. Rapport technique, TR00-023, University of North Carolina at Chapel Hil, 1999.

James H. Anderson et Anand Srinivasan : Early-release fair scheduling. *In Proc. of the 12th Euromicro Conference on Real-Time Systems*, ECRTS, 2000a.

James H. Anderson et Anand Srinivasan : Pfair scheduling : beyond periodic task systems. *In Proc. of the 7th International Conference on Real-Time Computing Systems and Applications*, RTCSA, 2000b.

Björn Andersson : Global static-priority preemptive multiprocessor scheduling with utilization bound. *In Principles of Distributed Systems*, volume 5401 de *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008.

Björn Andersson, Sanjoy Baruah et Jan Jonsson : Static-priority scheduling on multiprocessors. *In Proc. of the 22nd IEEE Real-Time Systems Symposium*, RTSS, 2001.

Björn Andersson et Jan Jonsson : Fixed-priority preemptive multiprocessor scheduling : to partition or not to partition. *In Proc. of the 7th International Conference on Real-Time Computing Systems and Applications*, RTCSA, 2000.

Björn Andersson et Eduardo Tovar : Multiprocessor scheduling with few preemptions. *In Proc. of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA, 2006.

Patrick Andrianiaina, Alexandre Seuret et Daniel Simon : Robust control under weakened real-time constraints. *In 50th IEEE Conference on Decision and Control and European Control Conference*, 2011.

Alia Atlas et Azer Bestavros : Statistical rate monotonic scheduling. *In Proc. of the 19th IEEE Real-Time Systems Symposium*, RTSS, 1998.

Neil C. Audsley : Deadline-monotonic scheduling. Rapport technique YCS 146, Departement of Computer Science, University of York, 1990.

Neil C. Audsley : On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1), 2001.

Guillaume Auriol, Vincent Mahout, Thierry Rocacher, Pascal Acco et Pierre-Emmanuel Hladik : La conception orientée objet au secours de la programmation de microcontrôleur ou inversement... *Journal sur l'enseignement des sciences et technologies de l'information et des systèmes*, 12, 2013.

Philip Axer, Rolf Ernst, Heiko Falk, Alain Girault, Daniel Grund, Nan Guan, Bengt Jonsson, Peter Marwedel, Jan Reineke, Christine Rochange, Maurice Sebastian, Reinhard Von Hanxleden, Reinhard Wilhelm et Wang Yi : Building timing predictable embedded systems. *ACM Trans. Embed. Comput. Syst.*, 13(4), 2014.

Vlastimil Babka, Peter Libič, Tomáš Martinec et Petr Tůma : On the accuracy of cache sharing models. *In Proc. of the 3rd joint WOSP/SIPEW international conference on Performance Engineering*, ICPE, 2012.

Theodor P. Baker : A stack-based resource allocation policy for realtime processes. *In Proc. of the IEEE Real-Time Systems Symposium*, RTSS, 1990.

Andrea Baldovin, Geoffrey Nelissen, Tullio Vardanega et Eduardo Tovar : SPRINT : Extending RUN to Schedule Sporadic Tasks. *In Proceedings of the 22nd RTNS*, 2014.

James Barhorst, Todd Belote, Pam Binns, Jon Hoffman, James Paunicka, Prakash Sarathy, John Scoredos, Peter. Stanfill, Douglas Stuart et Russell Urzi : A research agenda for mixed-criticality systems. White paper at Cyber-Physical Systems Week, 2009.

Sanjoy Baruah : Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24:93–128, 2003.

Sanjoy Baruah : Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*, 53(6):781–784, 2004.

Sanjoy Baruah, Deji Chen, Sergey Gorinsky et Aloysius Mok : Generalized multiframe tasks. *Real-Time Systems*, 17:5–22, 1999.

Sanjoy Baruah, Neil Cohen, Greg Plaxton et Donald Varvel : Proportionate progress : A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.

Sanjoy K. Baruah, Johannes E Gehrke et C. Greg Plaxton : Fast scheduling of periodic tasks on multiple resources. *In Proc. of the 9th International Parallel Processing Symposium*, pages 280–280, 1995.

Jean-Luc Bechennec, Michael Briday, Sébastien Faucou et Yvon Trinquet : Trampoline an open source implementation of the OSEK/VDX RTOS. *In Proc. of the 11th International Conference on Emerging Technologies and Factory Automation*, ETFA, 2006.

Sofien Beji, Abdelouahed Gherbi, John Mullins et Pierre-Emmanuel Hladik : Model-driven approach to the optimal configuration of time-triggered flow within a ttethernet network. *In Proc. of the 9th System Analysis and Modelling Conference*, SAM, 2016.

Thierry Benoist, Etienne Gaudin et Benoît Rottembourg : Constraint programming contribution to benders decomposition : A case study. *In Proc. of the 8th International Conference on Principles and Practice of Constraint Programming*, CP, 2002.

Guillem Bernat, Antoine Colin et Stefan Petters : pWCET a Toolset for automatic Worst-Case Execution Time Analysis of Real-Time Embedded Programs. *In 3rd Int. Workshop on WCET Analysis, at the Euromicro conference on Real-Time Systems*, 2003.

Bernard Berthomieu, Jean-Paul Bodeveix, Christelle Chaudet, Silvano Zilio, Mamoun Filali et François Vernadat : Formal verification of AADL specifications in the topcased environment. *In Proc. of the Ada-Europe*, 2009.

Bernard Berthomieu, Jean-Paul Bodeveix, Patrick Farail, Mamoun Filali, Hubert Garavel, Pierre Gaufillet, Frederic Lang et François Vernadat : Fiacre : an intermediate language for model verification in the topcased environment. *In Proc. of Embedded Real Time Software*, ERTS, 2008.

Bernard Berthomieu, Didier Lime, Olivier H. Roux et François Vernadat : Reachability. problems and abstract state spaces for time Petri nets with stopwatches. *Journal Discrete Event Dynamic Systems*, 2004a.

Bernard Berthomieu, Florent Peres et François Vernadat : Bridging the gap between timed automata and bounded time petri nets. *In Lecture Notes in Computer Science*, 2006.

Bernard Berthomieu, Pierre-Olivier Ribet et François Vernadat : The tool TINA – construction of abstract state spaces for Petri nets and time petri nets. *International Journal of Production Research*, 42(14), 2004b.

Marko Bertogna, Michele Cirinei et Giuseppe Lipari : New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. *In Proc. of the 9th International Conference on Principles of Distributed Systems*, OPODIS, 2005.

Antoine Bertout, Julien Forget et Richard Olejnik : Automated runnable to task mapping. Rapport technique, Laboratoire d'Informatique Fondamentale de Lille, 2013.

Franck Bimbard et Laurent George : FP/FIFO feasibility conditions with kernel overheads for periodic tasks on an event driven osek system. *In Proc. of the International Symposium on Object-Oriented Real-Time Distributed Computing*, ISORC, 2006.

Enrico Bini et Giorgio C. Buttazzo : Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2), 2005.

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill et David A. Wood : The gem5 simulator. *SIGARCH Computer Architecture News*, 2011.

Konstantinos Bletsas et Björn Andersson : Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. *In Proc. of the 30th IEEE Real-Time Systems Symposium*, RTSS, 2009.

Cornelius G. E. Boender et Alexander H. G. Rinnooy Kan : Bayesian stopping rules for multistart global optimization methods. *Mathematical Programming*, 37(1), 1987.

Frédéric Boniol, Pierre-Emmanuel Hladik, Claire Pagetti, Frédéric Aspro et Victor Jégu : A framework for distributing real-time functions. *In Proc. of the 6th International Conference on Formal Modeling and Analysis of Timed Systems*, FORMATS, 2008.

Matthias Brun : *Contribution à la considération explicite des plates-formes d'exécution logicielles lors d'un processus de déploiement d'application.* Thèse de doctorat, Univ. de Nantes, 2010.

Giacomo Bucci, Luigi Sassoli et Enrico Vicario : Oris : A tool for state-space analysis of real-time preemptive systems. *In Proc. of the The Quantitative Evaluation of Systems, First International Conference*, QEST, 2004.

Alan Burns et Sanjoy Baruah : Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 21, 2008.

Alan Burns, Robert Davis, P. Wang et Fengxian Zhang : Partitioned edf scheduling for multiprocessors using a C=D task splitting scheme. *Real-Time Systems*, 28(1), 2012.

Alan Burns, Brian Dobbing et Tullio Vardanega : Guide for the use of the ada ravenscar profile in high integrity systems. *Ada Letter*, XXIV(2), 2004.

Giorgio C. Buttazzo : *Hard Real-time Computing Systems : Predictable Scheduling Algorithms And Applications (Real-Time Systems Series), 3rd ed.* Springer, 2011. ISBN 978-1461406754.

John M. Calandrino, Hennadiy Leontyev, Aaron Block, UmaMaheswari C. Devi et James H. Anderson : Litmus$^{RT}$ : A testbed for empirically comparing real-time multiprocessor schedulers. *In Proc. of the 27th IEEE International Real-Time Systems Symposium*, RTSS, 2006.

Jean-Paul Calvez : *Spécification et Conception des Systèmes : une Méthodologie*. Edition Masson, 1990.

Hadrien Cambazard et Pierre-Emmanuel Hladik : Learning and cooperation for a hard real-time task allocating problem. *In Proc. of the 46th Annual conference of the Canadian Operational Research Society & The Institute for Operations Research and the Management Sciences International*, CORS-INFORMS, 2004.

Hadrien Cambazard, Pierre-Emmanuel Hladik, Anne-Marie Déplanche et Narendra Jussien : Deux approches pour la résolution d'un problëme d'allocation de tâches en temps-réel dur. *In Proc. of the 3e Journées Francophones de Programmation par Contrainte*, JFPC, 2007.

Hadrien Cambazard, Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Narendra Jussien et Yvon Trinquet : Decomposition and learning for a hard real time task allocation problem. *In Proc. of the Principles and Practice of Constraint Programming*, CP, 2004a.

Hadrien Cambazard, Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Narendra Jussien et Yvon Trinquet : Décomposition et apprentissage pour un problème d'allocation de tâches temps réel. *In Proc. des Journées Nationales sur la résolution Pratique de Problèmes NP-Complets*, JNPC, 2004b.

Antonino Casile, Giorgio C. Buttazzo, Gerardo Lamastra et Giuseppe Lipari : A scheduling simulator for real-time distributed systems. *In Proc. of the 15th workshop Distributed Computer Control Systems*, 1998.

Francisco J. Cazorla, Eduardo Quiñones, Tullio Vardanega, Liliana Cucu, Benoit Triquet, Guillem Bernat, Emery Berger, Jaume Abella, Franck Wartel, Michael Houston, Luca Santinelli, Leonidas Kosmidis, Code Lo et Dorin Maxim : Proartis : Probabilistically analyzable real-time systems. *ACM Transactions on Embedded Computing Systems*, 12(2s), 2013.

Anton Cervin : Stability and worst-case performance analysis of sampled-data control systems with input and output jitter. *In Proc. of the American Control Conference*, ACC, 2012.

Younès Chandarli, Frédéric Fauberteau, Damien Masson, Serge Midonnet et Manar Qamhieh : Yartiss : A tool to visualize, test, compare and evaluate real-time scheduling algorithms. *In Proc. of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, WATERS, 2012.

Dhruba Chandra, Fei Guo, Seongbeom Kim et Yan Solihin : Predicting inter-thread cache contention on a chip multi-processor architecture. *In Proc. of the 11th International Symposium on High-Performance Computer Architecture*, HPCA, 2005.

Élodie Chanthery, Gwendoline Le Corre et Pierre-Emmanuel Hladik : De l'illustration du guidage à l'optimisation d'un plan par un robot Lego Mindstorm NXT. *Journal sur l'enseignement des sciences et technologies de l'information et des systèmes*, 2016.

Hussein Charara : *Évaluation des performances temps réel de réseaux embarqués avioniques*. Thèse de doctorat, Institut National Polytechniques de Toulouse, 2007.

Maxime Chémary, Anne-Marie Déplanche et Pierre-Emmanuel Hladik : Simulation of real-time multiprocessor scheduling with overheads. *In Proc. of the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, SIMULTECH, 2013.

Maxime Chéramy : *Évaluation et mise en oeuvre des politiques d'ordonnancement temps réel multicoeur*. Thèse de doctorat, Univ. de Toulouse, Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse), 2014.

Maxime Chéramy, Anne-Marie Déplanche et Pierre-Emmanuel Hladik : Simulation d'ordonnancement temps réel avec prise en compte de l'impact des caches. *In Proc. of the École d'été Temps Réel 2013*, ETR, 2013.

Maxime Chéramy, Pierre-Emmanuel Hladik et Anne-Marie Déplanche : Simso : A simulation tool to evaluate real-time multiprocessor scheduling algorithms. *In Proc. of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, WATERS, 2014a.

Maxime Chéramy, Pierre-Emmanuel Hladik et Anne-Marie Déplanche : Algorithmes pour l'ordonnancement temps réel multiprocesseur. *Journal Européen des Systèmes Automatisés*, 48(7-8), 2015a.

Maxime Chéramy, Pierre-Emmanuel Hladik, Anne-Marie Déplanche et Sébastien Dubé : A flexible simulator for real-time scheduling. *In Proc. of the Work-In-Progess of the 9th IEEE International Symposium on Industrial Embedded Systems*, SIES, 2014b.

Maxime Chéramy, Pierre-Emmanuel Hladik, Anne-Marie Déplanche et Silvano Dal Zilio : Simulation of real-time scheduling algorithms with cache effects. *In Proc. of the 6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems*, WATERS, 2015b.

Hyeonjoong Cho et Binoy Ravindran : An optimal real-time scheduling algorithm for multiprocessors. *In Proc. of the 27th IEEE International Real-Time Systems Symposium*, RTSS, 2006.

Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quinones et Franscisco J. Cazorla : Measurement-based probabilistic timing analysis for multi-path programs. *In Proc. of the 24th Euromicro Conference on Real-Time Systems*, ECRTS, 2012.

Robert Davis et Alan Burns : Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *In Proc. of the 30th IEEE Real-Time Systems Symposium*, RTSS, 2009.

Robert I. Davis et Alan Burns : A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4), 2011.

Robert I. Davis et Shinpei Kato : Fpsl, fpcl and fpzl schedulability analysis. *Real-Time Systems*, 48(6):750–788, 2012.

Jose Luis Diaz, Jose Maria Lopez, Manuel Garcia, Antonio Manuel Campos, Kanghee Kim et Lucia Lo Bello : Pessimism in the stochastic analysis of real-time systems : Concept and applications. *In Proc. of the 25th IEEE International Real-Time Systems Symposium*, RTSS, 2004.

Friedrich Eisenbrand, Karthikeyan Kesavan, Raju S. Mattikalli, Martin Niemeier, Arnold W. Nordsieck, Martin Skutella, José Verschae et Andreas Wiese : Solving an avionics real-time scheduling problem by advanced ip-methods. *In Proc. of the 18th annual European conference on Algorithms : Part I*, ESA, 2010.

Cecilia Ekelin : *An Optimization Framework for Scheduling of Embedded Real-Time Systems*. Thèse de doctorat, Chalmers University of Technology, 2004.

David Eklov, David Black-Schaffer et Erik Hagersten : Fast modeling of shared caches in multicore systems. *In Proc. of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, HiPEAC, 2011.

Glenn A. Elliott, Kecheng Yang et James H. Anderson : Supporting real-time computer vision workloads using openvx on multicore+gpu platforms. *In IEEE Real-Time Systems Symposium*, 2015.

Paul Emberson, Roger Stafford et Robert Davis : Techniques for the synthesis of multiprocessor tasksets. *In Proc. of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, WATERS, 2010.

Jeremy P. Erickson, James H. Anderson et Bryan C. Ward : Fair lateness scheduling : reducing maximum lateness in g-edf-like scheduling. *Real-Time Systems*, 50(1), 2014.

Ming Fan et Gang Quan : Harmonic semi-partitioned scheduling for fixed-priority real-time tasks on multi-core platform. *In Proc. of the Design, Automation & Test in Europ*, pages 503–508, 2012.

Sébastien Faucou, Pierre-Emmanuel Hladik, Anne-Marie Déplanche et Yvon Trinquet : Overview of microkernel standards for real-time in-vehicle embedded systems. *In Proc. of the 4th Taiwanese-French Conference on Information Technology*, TFIT, 2008.

Sara Fleury, Matthieu Herrb et Raja Chatila : Genom : A tool for the specification and the implementation of operating modules in a distributed robot architecture. *In Proc. of the International Conference on Intelligent Robots and Systems*, 1996.

Ricardo Bedin Franca, Jean-Paul Bodeveix, Mamoun Filali, Jean-Francois Rolland, David Chemouil et Dave Thomas : The AADL behaviour annex – experiments and roadmap. *In Proc. of the 12th IEEE International Conference on Engineering Complex Computer Systems*, CECCS, 2007.

Drew Fudenberg et Jean Tirole : *Game Theory*. MIT Press, 1991.

Kenji Funaoka, Shinpei Kato et Nobuyuki Yamasaki : Work-conserving optimal real-time scheduling on multiprocessors. *In Proc. of the 20th Euromicro Conference on Real-Time Systems*, ECRTS, 2008.

Shelby Funk et Vijaykant Nanadur : Lre-tl : An optimal multiprocessor scheduling algorithm for sporadic task sets. *In Proc. of the 17th International Conference on Real-Time and Network Systems*, RTNS, 2009.

Laurent George : Etat de l'art sur la robustesse temporelle des systèmes temps-réel mono- état de l'art sur la robustesse temporelle des systèmes temps-réel monoprocesseur. *Journal Européen des Systèmes Automatisés*, 42(9), 2008.

Laurent George, Pierre Courbin et Yves Sorel : Job vs. portioned partitioning for the earliest deadline first semi-partitioned scheduling. *Journal of Systems Architecture*, 57 (5), 2011.

Laurent George, Paul Muhlethaler et Nicolas Rivierre : Optimality and non-preemptive real-time scheduling revisited. Rapport technique RR-2516, INRIA, 2006.

Sébastien Gérard, Julio Medina et Dorina Petriu : MARTE : A new standard for modeling and analysis of real-time and embedded systems. *In Proc. of the 19th Euromicro Conference on Real-Time Systems*, ECRTS, 2007.

Holger Giese, Bernhard Rumpe, Bernhard andSchätz et Janos Sztipanovits : Science and engineering of cyber-physical systems. Rapport technique, Dagstuhl Seminar, 2012.

Edgar N. Gilbert et Henry O. Pollak : Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1), 1968.

Joël Goossens, Sanjoy Baruah et Shelby Funk : Real-time scheduling on multiprocessors. *In Proc. of the 10th International Conference on Real-Time Systems Embedded System*, RTS, 2002.

Joël Goossens, Shelby Funk et Sanjoy Baruah : Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2-3), 2003.

Joël Goossens, Emmanuel Grolleau et Liliana Cucu-Grosjean : Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. *Real-Time Systems*, pages 1–25, 2016. ISSN 1573-1383. URL `http://dx.doi.org/10.1007/s11241-016-9256-1`.

Joël Goossens et Christophe Macq : Limitation of the hyper-period in real-time periodic task set generation. *In Proc. of the 9th Real-Time Systems Embedded System*, RTS, 2001.

Nan Guan, Martin Stigge, Wang Yi et Ge Yu : Fixed-priority multiprocessor scheduling with liu and layland's utilization bound. *In Proc. of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS, 2010.

Jann Gustafsson, Adam Betts, Andreas Ermedahl et Björn Lisper : The mälardalen wcet benchmarks - past, present and future. *In Proc. of the 10th International Workshop on Worst-Case Execution Time Analysis*, 2010.

Matthew Guthaus, Jeffrey Ringenberg, Dan Ernst, Todd Austin, Trevor Mudge et Richard Brown : Mibench : A free, commercially representative embedded benchmark suite. *In Proc. of the IEEE International Workshop on Workload Characterization (WWC-4)*, 2001.

M. González Harbour, J. J. Gutiérrez García, J. C. Palencia Gutiérrez et J. M. Drake Moyano : MAST : Modeling and analysis suite for real time applications. *In Proc. of the 13th Euromicro Conference on Real-Time Systems*, ECRTS, 2001.

John Hennessy et David Patterson : *Computer architecture : a quantitative approach*. Morgan Kaufmann, 2011.

Thomas Henzinger, Benjamin Horowitz et Christoph Kirsch : Giotto : a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1), 2003.

Jean Hladik, Michel Chrysos, Pierre-Emmanuel Hladik et Lorenzo Ugo Ancarani : *Mécanique quantique : Atomes et noyaux, applications technologiques - cours et exercices corrigés*. Dunod, 2009a. ISBN 978-2100521883.

Jean Hladik et Pierre-Emmanuel Hladik : *Calcul tensoriel en physique, cours et exercices corrigés*. Dunod, 1999. ISBN 978-2100040711.

Jean Hladik et Pierre-Emmanuel Hladik : *Quaternions réels, duaux et complexes*. Ellipse, 2016. ISBN 978-2340010468.

Pierre-Emmanue Hladik, Florent Péres et Xiaomu Shi : Analyse d'un modële AADL à l'aide de pola. *In Proc. of the 10e journée francophone sur les Approches Formelles dans l'Assistance au Développement de Logiciels*, AFADL, 2010.

Pierre-Emmanuel Hladik : *Formulaire de l'étudiant : Physique*. Ellipses Marketing, 1999. ISBN 978-2729899172.

Pierre-Emmanuel Hladik : Fast and tight analysis for autosar schedule tables. Rapport technique, LAAS-CNRS, 2016.

Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche et Narendra Jussien : Dynamic constraint programming for solving hard real-time allocation problems. Rapport technique RI2005 7, IRCCyN, 2005a.

Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche et Narendra Jussien : How solve allocation problems with constraint programming. *In Proc. of the Work-In-Progress Session of the 17th Euromicro Conference on Real-Time Systems*, WiP ECRTS, 2005b.

Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche et Narendra Jussien : Guiding architectural design process of hard real-time systems with constraint programming. *In Proc. of the 3rd Taiwanese-French Conference on Information Technology*, TFIT, 2006a.

Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche et Narendra Jussien : Solving allocation problems of hard real-time systems with dynamic constraint programming. *In Proc. of the 14th International Conference on Real-time and Network Systems*, RTNS, 2006b.

Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche et Narendra Jussien : Solving a real-time allocation problem with constraint programming. *Journal of Systems and Software*, 81(1), 2008.

Pierre-Emmanuel Hladik, Silvano Dal Zilio, Olivier Pasquier, Sébastien Pillement et Bernard Berthomieu : Outillage pour la modélisation, la vérification et la génération d'applications temporisées et embarquées. *In 15èmes journées Approches Formelles dans l'Assistance au Développement de Logiciels*, AFADL, 2016.

Pierre-Emmanuel Hladik et Anne-Marie Déplanche : Best-case response time analysis for precedence relations in hard real-time systems. *In Proc. of the Work-In-Progress Session of the 24th IEEE International Real-Time Systems Symposium*, WiP RTSS, 2003a.

Pierre-Emmanuel Hladik et Anne-Marie Déplanche : An extension of holistic schedulability analysis for precedence relations in multiprocessor systems. *In Proc. of the Work-In-Progress Session of the 15th Euromicro Conference on Real-Time Systems*, WiP ECRTS, 2003b.

Pierre-Emmanuel Hladik et Anne-Marie Déplanche : Ordonnancement préemptif à priorités fixes : Comparaison de méthodes analytiques de calcul de pire temps de réponse. *In Proc. of the 11th Real-Time and Embedded Systems*, RTS, 2003c.

Pierre-Emmanuel Hladik et Anne-Marie Déplanche : Extension au réseau CAN des problëmes de placement. Rapport technique RI2005 4, IRCCyN, 2005.

Pierre-Emmanuel Hladik et Anne-Marie Déplanche : Ordonnancement temps réel multiprocesseur partitionné et programmation par contraintes. *In Proc. of the 10e Conférence de la société FranÁaise de Recherche Opérationnelle et Aide à la Décision*, ROADEF, 2009.

Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Sébastien Faucou et Yvon Trinquet : Adequacy between AUTOSAR OS specification and real-time scheduling theory. *In Proc. of the 2nd International Symposium on Industrial Embedded Systems*, SIES, 2007a.

Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Sébastien Faucou et Yvon Trinquet : Schedulability analysis of OSEK/VDX applications. *In Proc. of the 15th International Conference on Real-Time and Network Systems*, RTNS, 2007b.

Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Sébastien Faucou et Yvon Trinquet : Response times analysis of AUTOSAR OS multitask software. Rapport technique 09059, LAAS-CNRS, 2009b.

Pierre-Emmanuel Hladik et Sébastien Di Mercurio : Plate-forme d'enseignement pour la conception et l'implémentation sur exécutif temps réel. *In Proc. of the Conférence sur l'Enseignement des Technologies et des Sciences de l'Information et des Systèmes*, CETSIS, 2013.

John N. Hooker et G. Ottoson : Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

Frank K. Hwang, Dana S. Richards et Pawel Winter : *The Steiner Tree Problem.* Annals of Discrete Mathematics. Elsevier Science, 1992. ISBN 9780080867939.

ILOG CPLEX : `http://www.ilog.com/products/cplex/`.

Vipul Jain et Ignacio E. Grossmann : Algorithms for hybrid milp/cp models for a class of optimization problems. *INFORMS Journal on Computing*, 13(4), 2001.

Mathai Joseph et Paritosh Pandya : Finding response times in a real-time system. *The Computer Journal*, 29(5), 1986.

Frédéric Jouault et Ivan Kurtev : On the architectural alignment of ATL and QVT. *In Proc. of the ACM Symposium on Applied Computing*, SAC, 2006.

Ulrich Junker : Quickxplain : Conflict detection for arbitrary constraint propagation algorithms. *In Proc. of the 8th International Joint Conference on Artificial Intelligence, Workshop on Modelling and Solving Problems with Constraints*, IJCAI, 2001.

Narendra Jussien : e-constraints : explanation-based constraint programming. *In Proc. of the Workshop on User-Interaction in Constraint Satisfaction*, CP, 2001.

Narendra Jussien : The versatility of using explanations within constraint programming. Habilitation thesis of Université de Nantes, 2003.

Nagarajan Kandasamy, John P. Hayes et Brian T. Murray : Dependable communication synthesis for distributed embedded systems. *In Computer Safety, Reliability, and Security*, volume 2788 de *Lecture Notes in Computer Science*, pages 275–288. Springer Berlin Heidelberg, 2003.

Qinma Kang et Hong He : Task assignment for minimizing application completion time using honeybee mating optimization. *Frontiers of Computer Science*, 7(3):404–415, 2013.

Shinpei Kato et Nobuyuki Yamasaki : Real-time scheduling with task splitting on multiprocessors. *In Proc. of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA, 2007.

Shinpei Kato et Nobuyuki Yamasaki : Global edf-based scheduling with efficient priority promotion. *In Proc. of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2008a.

Shinpei Kato et Nobuyuki Yamasaki : Portioned edf-based scheduling on multiprocessors. *In Proc. of the 8th ACM international conference on Embedded software*, EMSOFT, 2008b.

Shinpei Kato et Nobuyuki Yamasaki : Portioned static-priority scheduling on multiprocessors. *In Proc. of the IEEE International Symposium on Parallel and Distributed Processing*, IPDPS, 2008c.

Shinpei Kato et Nobuyuki Yamasaki : Semi-partitioning technique for multiprocessor real-time scheduling. *In Proc. of the 29th IEEE Real-Time Systems Symposium, Work-in-Progress Session*, WiP RTSS, 2008d.

Shinpei Kato et Nobuyuki Yamasaki : Semi-partitioned fixed-priority scheduling on multiprocessors. *In Proc. of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2009.

Shinpei Kato, Nobuyuki Yamasaki et Yutaka Ishikawa : Semi-partitioned scheduling of sporadic task systems on multiprocessors. *In Proc. of the 21st Euromicro Conference on Real-Time Systems*, ECRTS, 2009.

Omar Kermia et Yves Sorel : A rapid heuristic for scheduling non-preemptive dependent periodic tasks onto multiprocessor. *In Proc. of the 20th international conference on Parallel and Distributed Computing Systems*, PDCS, 2007.

Ashok Khemka et Rudrapatna K. Shyamasundar : An optimal multiprocessor real-time scheduling algorithm. *Journal of Parallel and Distributed Computing*, 43(1), 1997.

Heeheon Kim et Yookun Cho : A new fair scheduling algorithm for periodic tasks on multiprocessors. *Information Processing Letters*, 111(7), 2011.

Michel Kinsy et Srinivas Devadas : Algorithms for scheduling task-based applications onto heterogeneous many-core architectures. *In High Performance Extreme Computing Conference*, HPEC, 2014.

Christoph Kirsch et Raja Sengupta : *Handbook of Real-Time and Embedded Systems*, chapitre The Evolution of Real-Time Programming. Chapman and Hall/CRC, 2007.

Jan Korst : *Periodic multiprocessor scheduling*. Thèse de doctorat, Eindhoven university of technology, Eindhoven, the Netherlands, 1992.

Jan Korst, Emile Aarts et Jan Karel Lenstra : Scheduling periodic tasks. *INFORMS Journal on Computing*, 8, 1996.

Jan Korst, Emile Aarts et Jan Karel Lenstra : Scheduling periodic tasks with slack. *INFORMS Journal on Computing*, 9, 1997.

Leonidas Kosmidis, Charlie Curtsinger, Eduardo Quiñones, Jaume Abella, Emery Berger et Francisco J. Cazorla : Probabilistic timing analysis on conventional cache designs,. *In Design, Automation Test in Europe Conference Exhibition*, DATE, 2013.

Angeliki Kritikakou, Claire Pagetti, Matthieu Roy, Christine Rochange, Madeleine Faugère, Sylvain Girbal et Daniel Gracia Pérez : Distributed run-time wcet controller for concurrent critical tasks in mixed-critical systems. *In 22nd International Conference on Real-Time Networks and Systems*, 2014.

Łukasz Kruk, John P. Lehoczky, Steven Shreve et Shu-Ngai Yeung : Earliest-deadline-first service in heavy-traffic acyclic networks. *The Annals of Applied Probability*, 14(3), 2004.

Hee-Hwan Kwak, Insup Lee, Anna Philippou, Jin-Young Choi et Oleg Sokolsky : Symbolic schedulability analysis of real-time systems. *In Proc. of the 19th IEEE Real-Time Systems Symposium, 1998*, RTSS, 1998.

J. Labetoulle : Some theorems on real time scheduling. *Computer Architecture and Networks*, 1974.

Karthik Lakshmanan, Rangunathan Rajkumar et John P. Lehoczky : Partitioned fixed-priority preemptive scheduling for multi-core processors. *In Proc. of the 21st Euromicro Conference on Real-Time Systems*, ECRTS, 2009.

Gilles Lasnier, Bechir Zalila, Laurent Pautet et Jérome Hugues : Ocarina : An environment for aadl models analysis and automatic code generation for high integrity applications. *In Proc. of the Reliable Software Technologies – Ada-Europe 2009*, 2009.

Michael Lauer, Jérôme Ermont, Frédéric Boniol et Claire Pagetti : Latency and freshness analysis on IMA systems. *In Proc. of the IEEE 16th Conference on Emerging Technologies & Factory Automation*, ETFA, 2011.

Eugene Lawler : Recent results in the theory of machine scheduling. *Mathematical Programming : The State of the Art*, 1983.

Edward A. Lee : The past, present and future of cyber-physical systems : A focus on models. *Sensors*, 15(3):4837, 2015. ISSN 1424-8220. URL `http://www.mdpi.com/1424-8220/15/3/4837`.

Suk Kyoon Lee : On-line multiprocessor scheduling algorithms for real-time tasks. *In Proc. of the 9th IEEE TENCON'94*, 1994.

John P. Lehoczky : Fixed priority scheduling of periodic task sets with arbitrary deadlines. *In Proc. of the 11th Real-Time Systems Symposium*, RTSS, 1990.

John P. Lehoczky : Real-time queueing theory. *In Proc. of the 17th IEEE Real-Time Systems Symposium*, RTSS, 1996.

John P. Lehoczky : Real-time queueing network theory. *In Proc. of the 18th IEEE Real-Time Systems Symposium*, RTSS, 1997a.

John P. Lehoczky : Using real-time queueing theory to control lateness in real-time systems. *In Proc. of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS, 1997b.

Cédrick Lelionnais, Jérôme Delatour, Matthias Brun, Olivier Roux et Charlotte Seidner : Formal synthesis of real-time system models in a MDE approach. *International Journal on Advances in Systems and Measurements*, 7, 2014.

Juri Lelli, Dario Faggioli, Tommaso Cucinotta et Giuseppe Lipari : An experimental comparison of different real-time schedulers on multicore systems. *Journal of Systems and Software*, 85(10), 2012.

Juri Lelli, Claudio Scordino2, Luca Abeni et Dario Faggioli : Deadline scheduling in the linux kernel. *Software Practice and Experience*, 2015.

Matthieu Lemerre, Emmanuel Ohayon, Damien Chabrol, Mathieu Jan et Marie-Benedicte Jacques : Method and tools for mixed-criticality real-time applications within pharos. *In Proc. of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, ISORCW, 2011.

Joseph Leung, Laurie Kelly et James H. Anderson : *Handbook of Scheduling : Algorithms, Models, and Performance Analysis*. CRC Press, Inc., 2004. ISBN 1584883979.

Joseph Leung et Jennifer Whitehead : On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

Joseph Y.-T. Leung et M.L. Merril : A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11, 1980.

Greg Levin, Shelby Funk, Caitlin Sadowski, Ian Pye et Scott Brandt : Dp-fair : A simple model for understanding optimal multiprocessor scheduling. *In Proc. of the 22nd Euromicro Conference on Real-Time Systems (ECRTS), 2010*, ECRTS, 2010.

Chang L. Liu et James Layland : Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.

Fang Liu, Fei Guo, Yan Solihin, Seongbeom Kim et Abdulaziz Eker : Characterizing and modeling the behavior of context switch misses. *In Proc. of the 17th international conference on Parallel architectures and compilation techniques (PACT)*, 2008.

Jane W. S. W. Liu : *Real-Time Systems*. Prentice Hall, 2000.

José María López, José Luis Díaz, Joaquín Entrialgo et Daniel García : Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-Time Systems*, 40(2), 2008.

Stéphane Louise, Matthieu Lemerre, Christophe Aussagues et Vincent David : The oasis kernel : A framework for high dependability real-time systems. *In IEEE 13th International Symposium on High-Assurance Systems Engineering*, HASE, 2011.

Jukka Mäki-Turja et Mikael Nolin : Efficient response-time analysis for tasks with offsets. *In Prooceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS, 2004a.

Jukka Mäki-Turja et Mikael Nolin : Tighter response-times for tasks with offsets. *In Proc. of the 10th International conference on Real-Time Computing Systems and Applications*, RTCSA, 2004b.

Sorin Manolache, Petru Eles et Zebo Peng : *Real-time applications with stochastic task execution times analysis and optimisation.* Springer, 2007.

Mohamed Marouf et Yves Sorel : Schedulability conditions for non-preemptive hard real-time tasks with strict period. *In Proceedings of International Conference on Real-Time and Network Systems*, RTNS, 2010.

Rafael Martí : *Handbook of metaheuristics*, chapitre Multi-start methods. Springer, 2003.

Ernesto Massa et George Lima : A bandwidth reservation strategy for multiprocessor real-time scheduling. *In Proc. of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS, 2010.

Ernesto Massa, George Lima, Paul Regnier, Greg Levin et Scott Brandt : Optimal and adaptive multiprocessor real-time scheduling : The quasi-partitioning approach. *In Proc. of the 26th Euromicro Conference on Real-Time Systems*, ECRTS, 2014.

Richard L. Mattson, Jan Gecsei, Donald R.. Slutz et Irving L. Traiger : Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2), 1970.

Dorin Maxim, Mike Houston, Luca Santinelli, Guillem Bernat, Robert I. Davis et Liliana Cucu-Grosjean : Re-sampling for statistical timing analysis of real-time systems. *In the 20th International Conference on Real-Time and Network Systems*, 2012.

Asma Mehiaoui, Ernest Wozniak, Sara Tucci-Piergiovanni, Chokri Mraidha, Marco Di Natale, Haibo Zeng, Jean-Philippe Babau, Laurent Lemarchand et Sébastien Gerard : A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems. *In Proc. of the 14th ACM SIGPLAN/-SIGBED conference on Languages, compilers and tools for embedded systems*, LCTES, 2013.

Philip M. Merlin et David J. Farber : Recoverability of communication protocols : Implications of a theoretical study. *IEEE Transaction on Communications*, 24(9), 1976.

Patrick Meumeu Yomsi et Yves Sorel : Non-schedulability conditions for off-line scheduling of real-time systems subject to precedence and strict periodicity constraints. *In Proceedings of 11th IEEE International Conference on Emerging technologies and Factory Automation*, WiP ETFA, 2006.

Jörn Migge : *L'ordonnancement sous contraintes temps-réel : un modèle à base de trajectoires*. Thèse de doctorat, Université de Nice, 1999.

Jeffrey C. Mogul et Anita Borg : The effect of context switches on cache performance. *SIGOPS Operating Systems Review*, 25, 1991.

Aloysius Mok : Fundamental design problems of distributed systems for the hard-real-time environment. Rapport technique, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

Aloysius Mok et Deji Chen : A multiframe model for real-time tasks. *In Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 22 –29, 12 1996.

Geoffrey Nelissen, Vandy Berten, Vincent Nélis, Joël Goossens et Dragomir Milojevic : U-EDF : An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks. *In Proc. of the 24th Euromicro Conference on Real-Time Systems*, ECRTS, 2012.

Geoffrey Nelissen, Hang Su, Yifeng Guo, Dakai Zhu, Vincent Nélis et Joël Goossens : An optimal boundary fair scheduling. *Real-Time Systems*, 2014.

Viet Anh Nguyen, Damien Hardy et Isabelle Puaut : Scheduling of parallel applications on many-core architectures with caches : bridging the gap between WCET analysis and schedulability analysis. *In Proc. of the 9th Junior Researcher Workshop on Real-Time Computing*, JRWRTC, 2015.

Sung-Heun Oh et Seung-Min Yang : A modified least-laxity-first scheduling algorithm for real-time tasks. *In Proceeding of the Fifth International Conference on Real-Time Computing Systems and Applications*, RTCSA, 1998.

Yassine Ouhammou, Emmanuel Grolleau, Michael Richard, Pascal Richard et Frédéric Madiot : MoSaRT framework : A collaborative tool for modeling and analyzing embedded real-time systems. *In Complex Systems Design & Management*, 2014.

José C. Palencia et Michael González Harbour : Schedulability analysis for tasks with static and dynamic offsets. *In Proc. of the IEEE Real-time Systems Symposium*, RTSS, 1998.

José C. Palencia et Michael González Harbour : Exploiting precedence relations in the schedulability analysis of distributed real-time systems. *In Proceeding of the 20th Real-time Systems Symposium*, RTSS, 1999.

Florent Péres : *Réseaux de Petri temporels à inhibitions/permissions : Application à la modélisation et vérification de systèmes de tâches temps réel*. Thèse de doctorat, Université de Toulouse, 2010.

Florent Péres, Pierre-Emmanue Hladik et François Vernadat : POLA : un langage dédié au domaine des systëmes temps réel vérifiables par model cheking. *In Proc. of the 10e Conférence de la société FranÁaise de Recherche Opérationnelle et Aide à la Décision*, ROADEF, 2009a.

Florent Péres, Pierre-Emmanue Hladik et François Vernadat : Specification and verification of real-time systems using the POLA tool. *In Proc. of the 3rd International Workshop on Verification and Evaluation of Computer and Communication Systems*, VECoS, 2009b.

Florent Péres, Pierre-Emmanue Hladik et François Vernadat : Specification and verification of real-time systems using POLA. *International Journal of Critical Computer-Based Systems*, 2(3/4):332–351, 2011.

Guillaume Phavorin, Pascal Richard et Claire Maiza : Complexity of scheduling real-time tasks subjected to cache-related preemption delays. *In IEEE 20th Conference on Emerging Technologies & Factory Automation*, ETFA, 2015.

Padmanabhan Pillai et Kang G. Shin : Real-time dynamic voltage scaling for low-power embedded operating systems. *In Proc. of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP, 2001.

Michal Pióro et Deepankar Medhi : *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufmann Publishers Inc., 2004. ISBN 0125571895.

Clément Pira et Christian Artigues : Line search method for solving a non-preemptive strictly periodic scheduling problem. *In Proc. of the 6th Multidisciplinary International Scheduling Conference*, MISTA, 2013.

Victor Pollex, Timo Feld, Frank Slomka, Ulrich Margull, Ralph Mader et Gerhard Wirrer : Sufficient real-time analysis for an engine control unit. *In Prooceedings of the 21st International Conference on Real-Time and Network Systems*, RTNS, 2013.

Petar Radojković, Vladimir **v**Cakarević, Miquel Moretó, Javier Verdú, Alex Pajuelo, Francisco J. Cazorla, Mario Nemirovsky et Mateo Valero : Optimal task assignment in multithreaded processors : a sta- tistical approach. *SIGARCH Comput. Archit. News*, 2012.

Khaled S. Refaat et Pierre-Emmanuel Hladik : Efficient stochastic analysis of real-time systems via random sampling. *In Proc. of the 22nd Euromicro Conference on Real-Time Systems*, ECRTS, 2010.

Philippe Refalo : Impact-based search strategies for constraint programming. *In Proc. of the 10th International Conference on Principles and Practice of Constraint Programming*, CP, 2004.

Paul Regnier, George Lima, Ernesto Massa, Greg Levin et Scott Brandt : Run : Optimal multiprocessor real-time scheduling via reduction to uniprocessor. *In Proc. of the IEEE 32nd Real-Time Systems Symposium*, RTSS, 2011.

Juan Rivas, Javier Gutiérrez, José C. Palencia et Michael González Harbour : Deadline assignment in edf schedulers for real-time distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 26(10), 2015.

Thierry Rocacher, Vincent Mahout et Pierre-Emmanuel Hladik : Un jeu de tir laser ou comment mélanger le traitement de signal, l'électronique et l'assembleur. *Journal sur l'enseignement des sciences et technologies de l'information et des systèmes*, 12, 2013.

Luigi Sassoli et Enrico Vicario : Analysis of real time systems through the oris tool. *In Proc. of the 3rd International Conference on Quantitative Evaluation of Systems*, QUEST, 2006.

Jean-Luc Scharbarg, Frédéric Ridouard et Christian Fraboul : A probabilistic analysis of end-to-end delays on an afdx avionic network. *IEEE Transactions on Industrial Informatics*, 2009.

Oliver Scheickl et Michael Rudorfer : Automotive real time development using a timing-augmented autosar specification. *In Embedded Real Time Software*, 2008.

Klaus Schild et Jörg Würtz : Scheduling of time-triggered real-time systems. *Constraints*, 5(4), 2000.

Yongho Seok, Youngseok Lee, Yanghee Choi et Changhoon Kim : Explicit multicast routing algorithms for constrained traffic engineering. *In Proc. of the 7th International Symposium on Computers and Communications*, ISCC, 2002.

Lui Sha, Rangunathan Rajkumar et John P. Lehoczky : Priority inheritance protocols : An approach to real-time synchronization. *IEEE Transactions on Computer*, 39(9), 1990.

Joseph Sifakis : A framework for component-based construction. *In Third IEEE International Conference on Software Engineering and Formal Methods*, 2005.

Franck Singhoff, Jérôme Legrand, Laurent Nana et Lionel Marcé : Cheddar a flexible real time scheduling framework. *In Proc. of the 2004 annual ACM SIGAda international conference on Ada : The engineering of correct and reliable software for real-time & distributed systems using Ada and related technologies*, SIGAda, 2004.

P. Baltarejo Sousa, P. Souto, E. Tovar et K. Bletsas : The Carousel-EDF scheduling algorithm for multiprocessor systems. *In Proceedings of the 19th RTCSA*, 2013.

Anand Srinivasan et Sanjoy Baruah : Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84(2), 2002.

John Stankovic : Misconceptions about real-time computing. *IEEE Computer*, 21(10): 10–19, 1988.

John Stankovic et Krithi Ramamritham : What is predictability for real-time systems ? *Real-Time Systems*, 2(4), 1990.

Martin Stigge, Pontus Ekberg, Nan Guan et Wang Yi : The digraph real-time task model. *In Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 71 –80, 4 2011.

Radoslaw Szymanek, Flavius Gruian et Krzysztof Kuchcinski : Digital systems design using constraint logic programming. *In Proc. of the Practical Application of Constraint technologies and Logic Programming*, PACLP, 2000.

Erlendur S. Thorsteinsson : Branch-and-check : A hybrid framework integrating mixed integer programming and constraint logic programming. *Lecture notes in Computer Science*, 2239, 2001.

Ken Tindell : Adding time-offsets to scheduling analysis. Rapport technique YCS 221, Department of Computer Science, University of York, 1994.

Richard Urunuela, Anne-Marie Déplanche et Yvon Trinquet : Storm a simulation tool for real-time multiprocessor scheduling evaluation. *In Proc. of the Emerging Technologies and Factory Automation*, ETFA, 2010.

Yun Wang et Manas Saksena : Scheduling fixed-priority tasks with preemption threshold. *In Proc. of the IEEE International Conference on Real-Time Computing Systems and Applications*, RTCSA, 1999.

Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat et Per Stenström : The worst-case execution time problem – overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 2008.

Ernest Wozniak, Marco Di Natale, Haibo Zeng, Chokri Mraidha, Sara Tucci-Piergiovanni et Sebastien Gerard : Assigning time budgets to component functions in the design of time-critical automotive systems. *In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE' 14, 2014.

Haibo Zeng et Marco Di Natale : Using max-plus algebra to improve the analysis of non-cyclic task models. *In Proc. of the 25th Euromicro Conference on Real-Time Systems*, ECRTS, 2013.

Dakai Zhu, Daniel Mossé et Rami Melhem : Multiple-resource periodic scheduling problem : how much fairness is necessary ? *In Proc. of the 24th IEEE Real-Time Systems Symposium*, RTSS, 2003.