REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Batna 2

Faculté des Mathématiques et de l'Informatique

Département d'Informatique

THESE

En vue de l'obtention du diplôme de

Doctorat en Sciences en Informatique

Présentée par

Sonia-Sabrina BENDIB

Méthodologie de Conception de Systèmes Embarqués Temps Réel

Soutenue publiquement le 10/01/2019 devant le jury formé de :

Dr. Souhila BOUAM	**Présidente**	Université de Batna2
Pr. Hamoudi KALLA	**Rapporteur**	Université de Batna2
Pr.Abdelkamel TARI	**Examinateur**	Université de Bejaia
Pr. Ramdane MAAMRI	**Examinateur**	Université de Constantine2
Dr. Elkamel MERAH	**Examinateur**	Université de Khenchela
Dr. Hamouma MOUMEN	**Examinateur**	Université de Batna2

⋆ *Science sans conscience n'est que ruine de l'âme* ⋆

F. Rabelais

A la mémoire de mes parents adorés
A ta mémoire ma si douce Fahima
A Neddine et à mes trois perles
A toute personne sincère
où qu'elle soit

Je dédie ce travail

الحمد لله العليم بذات الصدور

Mes sincères remerciements vont aux personnes ayant contribué à l'aboutissement de cette thèse, tant sur le plan professionnel que sur le plan personnel:

En premier lieu, je tiens à remercier vivement mon directeur de thèse Mr Hamoudi Kalla, Professeur à l'université Batna 2 de Batna de m'avoir proposé un sujet passionnant et d'avoir dirigé cette thèse avec une compétence remarquable et une disponibilité sans faille. Ses conseils avisés et ses encouragements continus m'ont permis de mener cette thèse à son terme;

Reconnaissante également envers Mme Souhila Bouam Maitre de conférences à l'université Batna 2 de Batna d'avoir manifesté un intérêt particulier à notre travail et d'avoir accepté de présider le jury;

C'est un véritable honneur pour moi que Mr Abdelkamel Tari Professeur à l'université Abderrahmane Mira de Bejaia, Mr Ramdane Maamri Professeur à l'université Abdelhamid Mehri de Constantine, Mr Elkamel Merah Docteur à l'université de Khenchela et Mr Hamouma Moumen Docteur à l'université Batna 2 de Batna aient accepté d'examiner ce travail de recherche. Je les en remercie vivement tout en espérant être à la hauteur de leurs questionnements;

Il m'était parfois difficile de persévérer mais les encouragements incessants m'ont donné la force de continuer. Des collègues et ami(es) que je tiens à remercier sincèrement pour leur soutien indéfectible. De tout mon coeur, un prompt rétablissement à Mr Lahlouhi Amar à l'égard duquel j'ai une immense estime. Un merci tout particulier à Houda pour sa générosité et à Mr Betta Mohamed qui ne se lasse point de répondre présent lorsque son aide est sollicitée ... Merci beaucoup;

Mon travail de thèse a quelques fois été une source d'angoisse et de doute, ce sont ma famille et ma belle famille qui étaient et qui sont là pour me soutenir, je sais que je ne les en remercierai jamais assez;

De ce parcours, je retiens des discussions fructueuses, des rires et parfois aussi de grandes colères .... Une chose demeure, cependant, essentielle à mes yeux : Peu importe tes états d'âme, ta sincérité fera toute la différence ...

Il me tarde de remercier ma petite famille : Merci pour les mots doux et réconfortants, pardonnez-moi pour le temps que je vous ai pris et les mauvaises humeurs que je vous ai fait subir bien des fois. Je sais ! je sais ! Maman est parfois insupportable ! Mais n'oubliez jamais qu'elle vous adooooooore !!!

Et toi Neddine, merci pour ta présence, merci pour ton soutien, merci pour ta patience, merci beaucoup pour la fameuse réplique si encourageante :'Je sais que tu y arriveras Inch'Allah !', merci infiniment;

Une pensée toute particulière pour toi ma douce Fahima ... Tout au long de ce travail, je t'ai sentie là, bien présente ... Tu tenais tellement à soutenir une thèse, je sais que tu en étais largement capable sauf que le bon Dieu en a décidé autrement ... C'est à ta mémoire que ce travail sera présenté ...

مـلـخـص

أنظمة الوقت الحقيقي المدمجة عبارة عن أنظمة تمثل متطلباتها تحديًا للمطورين. يجب على هذه الأنظمة، بالإضافة إلى توفير الوظائف المطلوبة، أن تحترم بشكل حتمي قيود الوقت الحقيقي المفروضة عليها. يخص عملنا تطوير أنظمة الوقت الحقيقي المدمجة حيث يجدر الاهتمام بمرحلة جد مهمة وهي مرحلة الجدولة. نأخذ بعين الاعتبار خاصة الجدولة الثابتة ثنائية الهدف مع الموثوقية وطول الجدولة كهدفين يجب تحسينهما. يتم اقتراح نهجين؛ أوّلهما هرمي يحسّن الهدفين بالتناوب، في حين أن الثاني نهج تكييفي. يتمثل التكييف في توسيع مساحة البحث، وبالتالي ضمان تنوع أفضل للحلول. تم دمج النهجين على التوالي في منهجية (AAA) "المواءمة (Adéquation) الخوارزمية (Algorithme) البنية (Architecture)" لإعطاء نسختين سمينا أولاهما AAA-ثنائية الهرمية والثانية AAA-ثنائية التكييف.

الكلمات المفتاحية: أنظمة الوقت الحقيقي المدمجة، منهجية AAA ، الجدولة ثنائية الهدف ، معدل فشل النظام الإجمالي (GSFR) ، طول الجدولة ، التكييف ، التحسين متعدد الأهداف، واجهة باريتو.

Abstract

Real-time embedded systems are systems whose requirements are a challenge for developers. In addition to providing the required functionalities, such systems must satisfy real-time constraints imposed on them. Our work concerns the development of real-time embedded systems where interest is focused on a relevant stage of development namely scheduling step. More particularly, We consider bi-objective static scheduling with reliability and schedule length as objectives to optimize.

Two approaches are proposed, the first one is hierarchical optimizing alternately both objectives while the second one is adaptive. The adaptation consists of extending the search space in order to ensure a better solution diversiy. The two approaches are respectively integrated into the AAA (Adequation Algorithm Architecture) methodology to produce two versions which we called AAA-Bi-Hierarchical for the first approach and AAA-Bi-Adaptive for the second one.

Keywords : Real-time embedded systems, AAA methodology, Bi-objective scheduling, GSFR (Global System Failure Rate), Schedule length, Adaptation, Multi-objective optimization.

Résumé

Les systèmes embarqués temps réel sont des systèmes dont les exigences constituent un défi pour les développeurs. En plus d'offrir les fonctionnalités requises, ces systèmes doivent impérativement respecter les contraintes temps réel qui leur sont imposées. Notre travail concerne le développement des systèmes embarqués temps réel où l'intérêt est porté sur une étape non des moindres du développement à savoir l'étape d'ordonnancement. Plus particulièrement, nous considérons l'ordonnancement statique bi-objectifs avec la fiabilité et la longueur d'ordonnancement comme objectifs à optimiser. Deux approches sont proposées, la première est hiérarchique optimisant alternativement les deux objectifs tandis que la seconde est une approche adaptative. L'adaptation consiste à étendre l'espace de recherche assurant de ce fait une meilleure diversité des solutions. Les deux approches sont respectivement intégrées dans la méthodologie 'Adéquation Algorithme Architecture' (AAA) pour en donner deux versions que nous avons appelées AAA-Bi-Hiérarchique pour la première et AAA-Bi-Adaptative pour la seconde.

Mots-clés : Systèmes embarqués temps réel, Méthodologie AAA, Ordonnancement bi-objectifs, GSFR (Global System Failure Rate), Longueur d'ordonnancement, Adaptation, Optimisation multi-objectifs.

Table des matières

Introduction générale **1**

1 Systèmes embarqués temps réel 5

1.1 Introduction **7**

1.2 Systèmes temps réel **7**

1.2.1 Système 7

1.2.2 Types de systèmes 8

1.2.3 Du réactif au temps réel 9

1.2.4 Fonctionnement d'un système temps réel 10

1.3 Systèmes embarqués temps réel **11**

1.3.1 Systèmes embarqués 11

1.3.2 Systèmes embarqués temps réel et leur classification 12

1.4 Architectures embarquées temps réel **13**

1.4.1 Architecture matérielle 13

1.4.2 Architecture logicielle 15

1.5 Systèmes distribués embarqués temps réel **17**

1.5.1 Tâche temps réel 17

1.5.2 Ressource 22

1.5.3 Modes d'exécution 22

1.6 Ordonnancement temps réel 22

1.6.1 Problématique de l'ordonnancement 22

1.6.2 Algorithmes d'ordonnancement 23

1.6.3 Exactitude du système 25

1.6.4 Ordonnançabilité/Faisabilité 25

1.6.5 Analyse d'ordonnançabilité 26

1.7 Conclusion 27

2 Conception des systèmes embarqués temps réel 28

2.1 Introduction 30

2.2 Exigences posées par les systèmes temps réel 30

2.2.1 Exigences fonctionnelles 30

2.2.2 Exigences non fonctionnelles 31

2.3 Approches de développement 31

2.3.1 Génie logiciel classique 31

2.3.2 Co-design 34

2.3.3 Méthodologies orientées UML 36

2.3.4 Approches à base de composants 38

2.3.5 Ingénierie dirigée par les modèles 39

2.3.6 Techniques formelles 44

2.4 Méthodolgie AAA (Adéquation Algorithme Architecture) 46

2.5 Langages de développement 48

2.6 Quelques critères de comparaison 49

2.7 Conclusion 50

3 Optimisation multi-objectifs 51

3.1 Introduction 53

3.2 Aide à la décision 53

3.2.1 Problème de décision multicritère 54

3.2.2 Processus décisionnel 55

3.3 Optimisation combinatoire 56

3.3.1 Problème d'optimisation 57

3.3.2 Processus d'optimisation 58

3.4 Optimisation Multi-Objectifs : La solution aux problèmes réels d'optimisation 60

3.4.1 Optimisation multi-objectifs 61

3.4.2 Notion de dominance et d'optimalité 61

3.4.3 Équilibre Intensification/Diversification 64

3.5 Méthodes de résolution 65

3.5.1 Classification 65

3.5.2 Quelques métaheuristiques 68

3.6 Ordonnancement multi-objectifs : Une application de l'OMO 73

3.7 Conclusion 75

4 Ordonnancement bi-objectifs dans les systèmes embarqués temps réel 76

4.1 Introduction 78

4.2 Agrégation des deux objectifs en un seul 79

4.2.1 Principe de l'approche 79

4.2.2 Quelques travaux d'agrégation des deux objectifs en un seul 80

4.3 Transformation d'un objectif en contrainte 81

4.3.1 Principe de l'approche 81

4.3.2 Quelques travaux de transformation d'un objectif en contrainte 82

4.4 Approche par hiérarchisation 83

4.4.1 Principe de l'approche 83

4.4.2 Quelques travaux adoptant une approche hiérarchique 83

4.5 Approche génétique 84

4.5.1 Principe de l'approche 84

4.5.2 Quelques travaux adoptant une approche génétique 85

4.6 Comparaison d'algorithmes et Métriques 86

4.7 Conclusion 86

5 Approches proposées 87

5.1 Méthodologie Adéquation Algorithme Architecture (AAA) 89

5.2 Modèles d'entrée des approches proposées 90

5.2.1 Modèle d'application 90

5.2.2 Modèle de plateforme 91

5.2.3 Contraintes d'exécution et de communication 91

5.2.4 Notations 92

5.2.5 Longueur d'ordonnancement (Makespan) 93

5.2.6 Modèle de fiabilité 94

5.3 AAA-Bi-Hiérarchique 96

5.3.1 Principe de l'approche 96

5.3.2 Algorithme RBB (Reliability-Based Bi-objective scheduling heuristic) 98

5.3.3 Fonctionnement de RBB 99

5.3.4 Algorithme MBB (Makespan-Based Bi-objective scheduling heuristic) 101

5.3.5 Fonctionnement de MBB 102

5.3.6 Expérimentation de l'approche AAA-Bi-Hiérarchique 104

5.4 AAA-Bi-Adaptative 108

5.4.1 Principe de l'approche 108

5.4.2 Algorithme ARBB (Adaptive Reliability-Based Bi-objective scheduling heuristic) 111

5.4.3 Fonctionnement de ARBB 112
5.4.4 Procédure d'adaptation 112
5.4.5 Fonctionnement de la procédure d'adaptation 113
5.4.6 Algorithme AMBB (Adaptive Makespan-Based Bi-objective scheduling heuristic) 114
5.4.7 Fonctionnement de AMBB 115
5.4.8 Flot d'implantation 116
5.4.9 Expérimentation de l'approche AAA-Bi-Adaptative 117

5.5 Conclusion 121

Conclusion générale et perspectives 123

Bibliographie 126

Bibliographie 126

Table des figures

1.1 Système et Environnement. 7
1.2 Classification des systèmes [Harel and Pnueli, 1985]. 9
1.3 Principe général d'un système temps réel. 10
1.4 Architectures centralisée et multi-calculateurs. 14
1.5 Architecture faiblement distribuée. 14
1.6 Architecture fortement distribuée. 15
1.7 Architecture logicielle d'un système temps réel [Dorin, 2010]. 16
1.8 Etats et transitions d'une tâche [Meumeu Yomsi, 2009]. 18
1.9 Modèle d'une tâche temps réel périodique [Meumeu Yomsi, 2009]. 20
1.10 Exemple de graphe de précédence. 21

2.1 Le modèle de développement en V. 32
2.2 Processus simplifié de codesign. 34
2.3 Flots de conception. 35
2.4 Les différentes couches de la méta-modélisation de l'OMG. 41
2.5 Transformation de modèles. 42
2.6 Vue globale du processus de développement *ACCORD/UML* [Taha, 2008]. 42
2.7 Flot de co-conception basé *MDA* dans l'environnement *Gaspard2* [Cherif, 2014]. . . 43
2.8 Techniques formelles pour la vérification. 44
2.9 Techniques formelles lors d'une transformation. 45

2.10 Illustration de la méthodologie *AAA* [Lecomte, 2011]. 47

3.1 Le processus de décision selon le modèle IDCR (de Simon). 55
3.2 Le processus d'optimisation. 59
3.3 Cycle Décision/Optimisation. 60
3.4 Espaces des solutions réalisables et des fonctions-objectifs. 62
3.5 Exemple de dominance (cas de deux objectifs à minimiser). 62
3.6 Types de solutions dans l'espace des solutions. 63
3.7 Convergence et Diversité. 65
3.8 Structure d'un chromosome. 71
3.9 Exemple de croisement. 72
3.10 Exemple de mutation. 72
3.11 Espace des objectifs à deux dimensions. 74

4.1 Méthode de la moyenne arithmétique pondérée. 79
4.2 Transformation d'un objectif en contrainte. 82
4.3 Approche par hiérarchisation des objectifs. 83

5.1 Flot d'implantation. 89
5.2 Exemple de graphe d'algorithme. 91
5.3 Exemple de graphe d'architecture. 91
5.4 Problème initial. 97
5.5 Principe de l'approche hiérarchique. 98
5.6 Restriction itérative de l'espace des objectifs. 103
5.7 Flot partiel d'implantation. 104
5.8 Impact de N sur le Makespan pour CCR=1 et P=4. 105
5.9 Impact de N sur la fiabilité pour CCR=1 et P=4. 105
5.10 Impact du CCR sur le Makespan pour N=50 et P=4. 106
5.11 Impact du CCR sur la fiabilité pour N=50 et P=4. 106

5.12 Impact de P sur le Makespan pour N=50 et CCR=1. 107
5.13 Impact de P sur la fiabilité pour N=50 et CCR=1. 107
5.14 Principe de l'approche adaptative. 109
5.15 Principe du module d'adaptation. 109
5.16 Génération de solutions voisines. 110
5.17 Flot partiel d'implantation. 116
5.18 Flot d'implantation. 117
5.19 Impact de N sur le Makespan pour CCR=1 et P=4. 118
5.20 Impact de N sur la fiabilité pour CCR=1 et P=4. 118
5.21 Impact du CCR sur le Makespan pour N=50 et P=4. 119
5.22 Impact du CCR sur la fiabilité pour N=50 et P=4. 119
5.23 Impact de P sur le Makespan pour N=50 et CCR=1. 120
5.24 Impact de P sur la fiabilité pour N=50 et CCR=1. 120

Liste des tableaux

2.1 Tableau récapitulatif des approches de développement. 50

5.1 Exemple de temps d'exécution. 92

5.2 Exemple de temps de communication. 92

5.3 Sémantique des notations. 93

5.4 Exemple de taux de panne de processeurs/liens. 96

5.5 Résultats du Makespan et de la fiabilité pour N=60. 120

Introduction générale

Le développement d'un système quel qu'il soit est un processus fastidieux, il l'est, néanmoins, davantage lorsque le système est complexe. Tel est le cas des systèmes embarqués temps réel dont les exigences sont fortes et les enjeux délicats. Nombreuses sont, aujourd'hui, les applications concernées par le temps réel. La robotique, le contrôle de processus industriels ou encore les télécommunications en sont des exemples.

Le développement de tels systèmes nécessite la prise en compte des fonctionnalités devant être offertes d'une part et des contraintes temps réel à respecter d'autre part. Ces dernières constituent, en effet, un aspect essentiel notamment pour les systèmes temps réels durs où le manquement d'une échéance peut s'avérer catastrophique. L'aspect critique de ces systèmes constitue un élément clé dans leur développement et les approches traitant cette problématique sont nombreuses. En effet, que ce soit via le génie logiciel classique ou l'ingénierie dirigée par les modèles ou encore des méthodes dédiées, l'objectif demeure la conception de systèmes répondant aux besoins des utilisateurs et respectant pertinemment l'ensemble des contraintes temps réel imposées.

Appréhender le développement de tels systèmes peut concerner toute leur prise en charge, de la définition des besoins jusqu'à la réalisation finale, ce qui aboutit en général à l'élaboration d'une méthodologie de développement. D'autre part, l'intérêt peut être porté sur une étape particulière du processus de développement où des modifications et des améliorations sont opérées, ce qui permet de produire des versions améliorées de méthodologies existantes. Tel est le cadre de notre

travail où la méthodologie considérée est 'Adéquation Algorithme Architecture' (AAA) [Grandpierre et al., 1999] et l'étape du processus de développement à laquelle nous nous intéressons est celle de l'ordonnancement .

Partitionner les différentes tâches d'un système sur les ressources disponibles d'une manière optimale est l'objectif principal de l'étape d'ordonnancement. Le résultat de cette étape conditionne fortement la suite du processus de développement. Par ailleurs, la qualité d'un ordonnancement est multidimensionnelle, ce qui nécessite la prise en considération d'un ensemble de critères. En effet, l'ordonnancement multicritère prend en considération non plus un seul critère avec comme résultat une solution unique, mais plusieurs critères qui sont souvent conflictuels. La résolution d'un tel problème est à la jointure de deux domaines à savoir l'aide multicritère à la décision et l'optimisation combinatoire. Des fonctions objectifs sont associés aux critères considérés permettant ainsi leur évaluation. Ce qui en résulte est un ensemble de solutions faisant des compromis entre les différents objectifs et permettant au décideur de choisir les solutions en adéquation avec les besoins courants.

Le travail présenté dans cette thèse concerne le développement de systèmes embarqués temps réel. Le point de départ est la méthodologie AAA dont l'étape d'ordonnancement constitue l'objet de notre travail. Nous considérons l'ordonnancement statique bi-objectifs avec deux critères pertinents pour un système temps réel en général à savoir la fiabilité et la longueur d'ordonnancement. En revanche, notre modèle de fiabilité [Girault and Kalla, 2009] considère le système comme une seule opération sur un processeur afin d'éviter la dépendance entre le taux de panne et la durée de la tâche. L'ordonnancement étant bi-objectif, notre but est de produire des solutions faisant les meilleurs compromis possibles entre les deux objectifs considérés. Cependant, à défaut de produire exactement les solutions optimales constituant ce qui est communément appelé *solutions Pareto*, notre objectif est de s'en approcher le plus possible.

Deux approches sont proposées, la première est une approche hiérarchique supportée par deux heuristiques qui coopèrent en optimisant alternativement les deux objectifs. Quant à la seconde approche, elle utilise une stratégie d'adaptation en étendant l'espace de recherche puis en ajustant

les solutions, ce qui permet d'éviter une convergence prématurée des solutions. Les intégrations respectives des deux approches bi-objectives dans la méthodologie AAA permettent d'en dégager deux versions. La première version *AAA-Bi-Hiérarchique* est une méthodologie basée sur une politique d'ordonnancement hiérarchique. Pour ce qui est de la seconde version *AAA-Bi-Adaptative*, elle utilise une stratégie d'adaptation afin de mieux exploiter l'espace de recherche et d'assurer une meilleure diversification des solutions.

Le reste du document est structuré en adéquation à notre démarche, il est constitué des parties suivantes :

Chapitre1 : Ce chapitre décrit le contexte scientifique de notre travail. Ainsi, il concerne les systèmes embarqués temps réel. Ces derniers y sont décrits à travers leurs caractéristiques, leur fonctionnement ainsi que leurs exigences.

Chapitre2 : Dans ce chapitre, nous nous focalisons sur l'aspect développement. Pour ce faire, nous abordons les approches de développement des systèmes embarqués temps réel et nous en présentons une classification.

Chapitre3 : L'ordonnancement bi-objectif, étape du développement à laquelle nous nous intéressons, est un champ d'investigation de l'optimisation multi-objectifs, aussi ce chapitre est-il un survol des approches de résolution dans le domaine de l'optimisation multi-objectifs.

Chapitre4 : Notre travail étant axé sur l'étape d'ordonnancement et plus particulièrement l'ordonnancement bi-objectifs, les approches traitant cette problématique sont présentées dans ce chapitre. Ceci permettra un positionnement ainsi qu'une évaluation de notre travail.

Chapitre5 : Ce chapitre concerne notre contribution, elle consiste en la proposition de deux versions modifiées de la méthodologie AAA et ce en y intégrant les deux approches d'ordonnancement statique bi-objectifs proposées. Les expérimentations effectuées en vue de valider les approches proposées y sont également présentées.

Conclusion générale est perspectives : Dans cette partie, nous présentons une récapitulation de la démarche suivie ainsi que notre contribution avec ses points forts et ses point faibles. Nous finirons, enfin, par quelques perspectives à notre travail.

Chapitre 1

Systèmes embarqués temps réel

Résumé

Dans ce chapitre, nous abordons les systèmes embarqués temps réel, objet de notre étude. Nous présentons, dans une première partie, leurs types ainsi que leurs particularités tandis que la seconde partie, nous la consacrons à l'ordonnancement temps réel, étape pertinente dans le développement des systèmes temps réel d'une manière générale.

Sommaire

1.1 Introduction **7**

1.2 Systèmes temps réel **7**

1.2.1 Système 7

1.2.2 Types de systèmes 8

1.2.3 Du réactif au temps réel 9

1.2.4 Fonctionnement d'un système temps réel 10

1.3 Systèmes embarqués temps réel **11**

1.3.1 Systèmes embarqués 11

1.3.2 Systèmes embarqués temps réel et leur classification 12

1.4 Architectures embarquées temps réel 13

1.4.1 Architecture matérielle 13

1.4.2 Architecture logicielle 15

1.5 Systèmes distribués embarqués temps réel 17

1.5.1 Tâche temps réel 17

1.5.2 Ressource 22

1.5.3 Modes d'exécution 22

1.6 Ordonnancement temps réel 22

1.6.1 Problématique de l'ordonnancement 22

1.6.2 Algorithmes d'ordonnancement 23

1.6.3 Exactitude du système 25

1.6.4 Ordonnançabilité/Faisabilité 25

1.6.5 Analyse d'ordonnançabilité 26

1.7 Conclusion 27

1.1 Introduction

Les systèmes temps réels sont des systèmes permettant la réalisation d'applications temps réel. Une application est dite temps réel si elle exécute ses actions dans un délai borné. En effet, un système temps réel diffère d'un système de traitement d'informations classique par le fait que la valeur d'une donnée produite ne dépend pas uniquement de la correction de son calcul mais également de la date à laquelle la donnée est disponible.

On retrouve les contraintes temporelles dans de nombreux secteurs d'activités tels que l'aéronautique au travers des systèmes de pilotage embarqués (avions, satellites), l'automatisation d'ensembles industriels au travers des systèmes de contrôle de procédés (usines, centrales nucléaires), la gestion, les télécommunications, l'automobile, la robotique, . . . etc.

1.2 Systèmes temps réel

1.2.1 Système

D'une manière générale, un système est vu comme un ensemble composite constitué de personnels, de matériels, de logiciels et de procédures. Ces éléments, en interaction mutuelle, sont organisés pour répondre à un besoin et correspondre à une finalité.

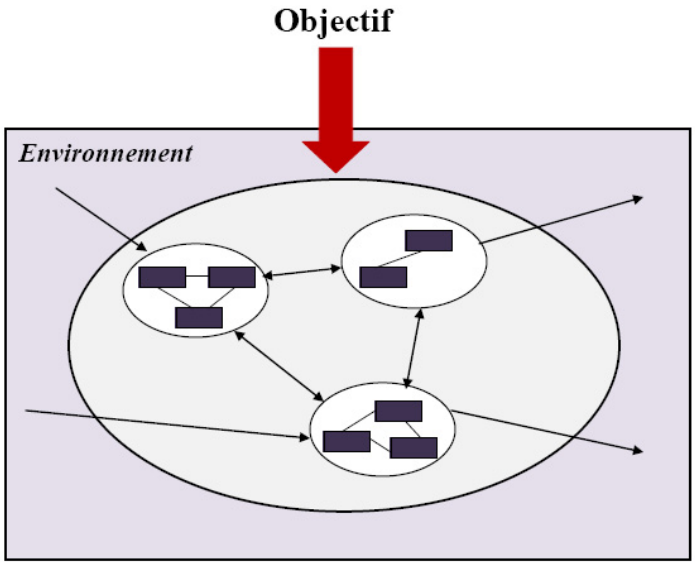

FIGURE 1.1: Système et Environnement.

La définition la plus large stipule qu'un système est un ensemble d'éléments (Figure 1.1) en relation les uns avec les autres et formant un tout. Cela veut dire qu'un système est une entité identifiable qui évolue dans un environnement. Une frontière existe et départage un système de son environnement.

Un système informatique est un type de système, il peut être classé en plusieurs catégories (cf. sous-section 1.2.2).

1.2.2 Types de systèmes

Une classification établie par G. Berry [Berry et al., 1987] se base sur le mode d'interactions entre l'environnement physique et le système. On y distingue deux types de systèmes : transformationnel et réactif. Selon Harel [Harel and Pnueli, 1985], la classification des systèmes informatiques est comme suit :

— ***Système transformationnel***

Un système transformationnel produit des sorties en fonction d'entrées selon un processus de calcul indépendant de l'environnement. Dans ce cas, les instants de production de résultats ne sont pas contraints car pour de tels systèmes, c'est l'exactitude logique qui prime. Un compilateur, un éditeur de factures sont des exemples de systèmes transformationnels.

— ***Système réactif***

Un système réactif interagit continuellement et instantanément avec son environnement. Ce dernier tient une place importante dans le sens où ce sont les dynamiques de l'environnement qui contraignent la temporalité des interactions. De cette manière, un système réactif élabore et produit des sorties en fonction d'entrées et en réaction à des évènements produits par l'environnement. Un contrôleur de processus industriel, une IHM critique sont des exemples de systèmes réactifs.

— ***Système interactif***

Un système interactif réagit constamment avec son environnement tout en imposant le rythme à ce dernier. Dans ce cas, le temps n'intervient pas en tant que tel si ce n'est avec un aspect

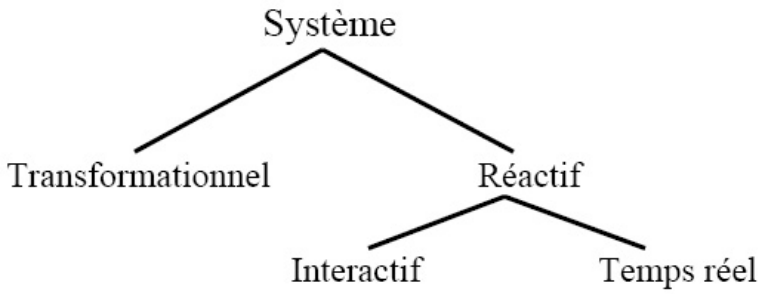

FIGURE 1.2: Classification des systèmes [Harel and Pnueli, 1985].

confort de travail ou qualité de service. Deux exemples de ce type de ce systèmes sont une base de données et une IHM non critique.

— ***Système temps réel***

Un système temps réel doit réagir dans le respect de contraintes temporelles à des sollicitations émises par un environnement extérieur. La réaction du système est déterminée par les évènements reçus et par l'état courant. On y distingue la capacité à pouvoir appréhender un flux d'évènements asynchrones issus d'un processus, sans perdre un seul de ces évènements et de traiter chacun d'eux en un temps déterminé.

1.2.3 Du réactif au temps réel

Les définitions des système temps réel sont nombreuses mais complètement convergentes, nous en citerons :

Définition1 [CNRS, 1988]

Les systèmes temps réel sont des systèmes réactifs qui permettent l'implantation d'applications où le respect des contraintes temporelles est la principale contrainte à satisfaire. Une application temps réel est une application qui met en œuvre un système informatique dont le fonctionnement est assujetti à l'évolution dynamique de l'état d'un environnement (procédé) qui lui est connecté et dont il doit contrôler le comportement.

Définition2 [Stankovic, 1988]

Un système temps réel est un système dont l'exactitude des résultats ne dépend pas seulement de l'exactitude logique des calculs mais aussi de la date à laquelle le résultat est produit. Si les contraintes

temporelles ne sont pas satisfaites, on dit qu'une défaillance système s'est produite.

En effet, la validité d'un système temps réel est conditionnée par deux attributs à savoir les résultats logiques du traitement effectué ainsi que l'aspect temporel, autrement dit, un système temps réel doit satisfaire deux contraintes importantes :

— ***Exactitude logique*** : fournir des sorties adéquates assurant le comportement désiré pour le système suite à des évènements en entrée et aux données communiquées.

— ***Exactitude temporelle*** : respecter des contraintes temporelles associées aux traitements effectués par le système pendant toute la durée de vie du système.

Par conséquent, dans un système temps réel, un résultat hors échéance reste faux même s'il s'avère logiquement correct.

1.2.4 Fonctionnement d'un système temps réel

Un système temps réel est composé essentiellement de deux éléments distincts (Figure 1.3) à savoir une ou plusieurs entités physiques constituant le procédé (le système contrôlé) et un système de contrôle (contrôleur) qui est chargé de surveiller de manière régulière, périodique, l'état du procédé en récupérant les valeurs en provenance des capteurs.

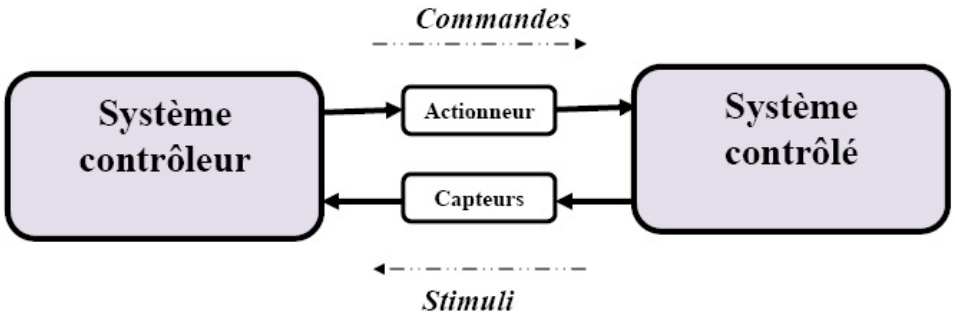

FIGURE 1.3: Principe général d'un système temps réel.

Les capteurs scrutent les évènements du système contrôlé et fournissent des mesures tandis que les actionneurs effectuent les réactions du système.

1.3 Systèmes embarqués temps réel

Les systèmes embarqués temps réel ont depuis quelques années envahi un grand nombre de domaines. Ils sont présents dans le contrôle industriel, les télécommunications, en aéronautique, en automobile ...etc. La fonction principale d'un système embarqué est celle de contrôler l'évolution de son environnement. En plus d'êtres réactifs, de tels systèmes sont souvent soumis à des contraintes strictes imposées par leurs environnements.

1.3.1 Systèmes embarqués

On qualifie de "système embarqué" un système électronique et informatique autonome dédié à une tâche précise, souvent en temps réel, possédant une taille limitée et ayant une consommation énergétique restreinte. Le terme "embarqué" est utilisé lorsque le système est situé à l'intérieur de l'environnement qu'il doit contrôler. Il s'agit donc d'une mise en œuvre conjointe de matériel et de logiciel pour gérer un dispositif qui n'est pas un ordinateur et qui interagit avec le monde extérieur. En plus des données d'entrées à traiter, les systèmes embarqués doivent prendre en compte des contraintes. Celles-ci peuvent être aussi bien au niveau matériel qu'au niveau logiciel :

1. ***Contraintes au niveau matériel*** :

   Ce sont des contraintes particulièrement en relation avec des limitations matérielles :

   — Un faible poids et une faible consommation ;

   — La résistance aux chocs, température ...etc ;

   — La limitation de la réserve d'énergie ;

   — La limitation de la mémoire ;

2. ***Contraintes au niveau logiciel*** :

   Ce type de contraintes concerne notamment les fonctionnalités offertes par le système ainsi que les contraintes temporelles devant être respectées telles que :

   — La sûreté de fonctionnement (capacité à accorder une confiance justifiée au service) ;

— La résistance aux pannes dans le sens où la panne d'un composant ne doit pas remettre en cause la vie du système ;

— Le développement en environnement croisé (cross compilation).

La plupart des systèmes embarqués sont également temps réel, cela signifie qu'ils doivent satisfaire les mêmes exigences que celles d'un système temps réel.

1.3.2 Systèmes embarqués temps réel et leur classification

Dans le contexte des systèmes temps réel, les données ont une validité définie à la fois par le domaine de valeurs admises et par la durée de validité (temporelle) qui dépend naturellement de l'échéance. Les données ont par conséquent une durée d'existence limitée [Zaffalon, 2007]. Selon la criticité des contraintes temporelles, on distingue trois types de systèmes temps réel :

— ***Systèmes à contraintes strictes (temps réel dur)***

Les systèmes temps réel à contraintes strictes ont un comportement déterministe. Dans ce cas, toutes les contraintes doivent être impérativement respectées. En effet, les systèmes à contraintes temporelles dures ne tolèrent qu'une gestion stricte du temps afin de conserver l'intégrité du service rendu. Le non respect des contraintes peut provoquer des conséquences catastrophiques [XU and Parnas, 1991]. Ce genre de systèmes est commun dans les applications touchant la sécurité du public. Les systèmes de contrôle de station nucléaire, les systèmes de contrôle de voies ferrées ainsi que la médecine assistée par ordinateur en sont des exemples.

— ***Systèmes à contraintes relatives (temps réel souple)***

Cette classe de système est moins exigeante quant au respect de toutes les contraintes temporelles. Cela signifie que le non respect des contraintes temporelles est toléré par le système sans que cela ait des conséquences catastrophiques [Chevochot and Puaut, 1999a, Chevochot and Puaut, 1999b]. Les systèmes à contraintes temporelles souples ou molles acceptent des variations dans le traitement des données, on parle alors de *Qualité de Service*. Cela signifie que ce sont des systèmes où la qualité est appréciée par les sens de l'être humain sous la forme

d'un service et qu'une faible probabilité de ne pas respecter des limites temporelles peut être tolérée. c'est le cas de systèmes et d'applications multimédia (téléphonie, vidéo ... etc).

— ***Systèmes à contraintes mixtes***

Ce sont des systèmes composés de tâches à contraintes strictes et de tâches à contraintes relatives [Grolleau, 1999]. Par conséquent, il s'en dégage un sous ensemble de tâches devant impérativement respecter des contraintes temporelles et un autre sous-ensemble de tâches dont le critère d'évaluation est la minimisation de fautes temporelles.

Notons par ailleurs qu'un système temps-réel inclut généralement différents sous-systèmes chacun pouvant avoir ses propres contraintes temporelles (dures, souples ou mixtes).

1.4 Architectures embarquées temps réel

1.4.1 Architecture matérielle

Les systèmes temps réel sont fortement influencés par l'architecture matérielle sur laquelle doit s'exécuter le système informatique. Selon le nombre de processeurs et l'utilisation éventuelle d'un réseau, on distingue [Kocik, 2000].

— ***Architecture centralisée***

Le système est dans ce cas composé d'un calculateur pouvant être monoprocesseur ou multiprocesseur à mémoire partagée, auquel sont reliés des capteurs et des actionneurs. Le calculateur suffit à centraliser l'acquisition, le traitement et la mémorisation des informations. Le système étant centralisé, son architecture matérielle en est simplifiée.

En revanche, ce type d'architecture conduit à un type de câblage de type *étoile* souvent important et couteux [Kocik, 2000]. En plus, un système centralisé est plus vulnérable aux défaillances.

— ***Architecture multi-calculateurs***

Le système est formé d'un ensemble de calculateurs non reliés entre eux où chaque calculateur implémente un certain nombre de fonctionnalités. Dans ce type d'architecture, les câblages sont

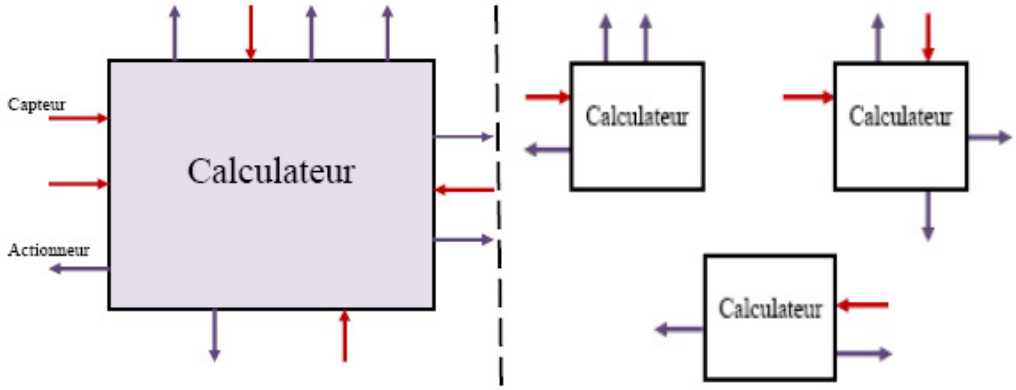

FIGURE 1.4: Architectures centralisée et multi-calculateurs.

réduits, ce qui permet de limiter les coûts et d'améliorer la fiabilité. Cependant, la gestion de la communication entre les différents systèmes est un problème crucial.

— ***Architecture faiblement distribuée***

C'est une architecture dans laquelle des calculateurs sont reliés entre eux par un bus. Dans ce cas, le comportement global est mieux contrôlé et les calculateurs peuvent partager des capteurs.

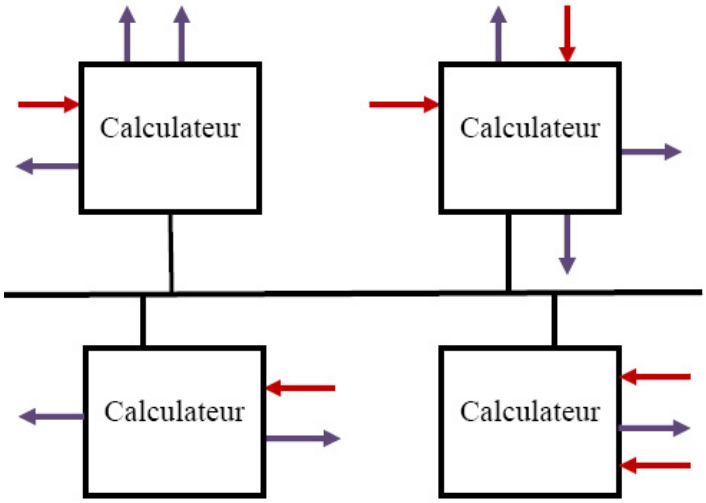

FIGURE 1.5: Architecture faiblement distribuée.

— ***Architecture fortement distribuée***

L'architecture est caractérisée par une plus grande autonomie des composants. En effet, les capteurs ainsi que les actionneurs peuvent être directement connectés sur le bus (Figure 1.6).

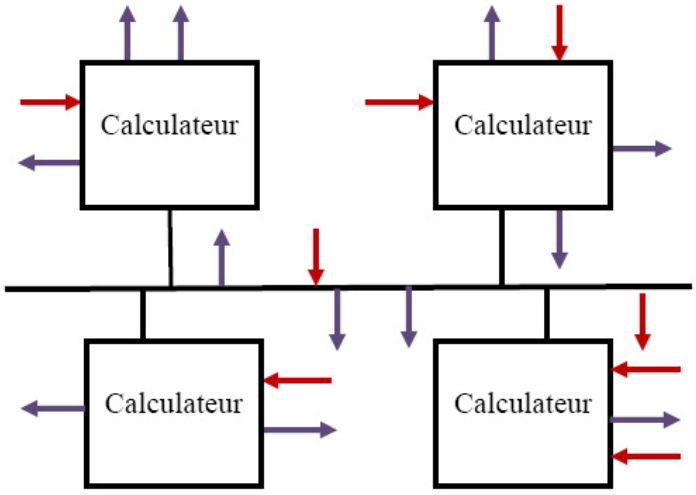

FIGURE 1.6: Architecture fortement distribuée.

— ***Architecture hétérogène***

De nos jours, en plus d'être fortement distribuées, les architectures embarquées temps réel sont aussi hétérogènes. En effet, les composants peuvent être des microprocesseurs, des micro-contrôleurs, différents médias de communication . . . etc. En revanche, la difficulté réside dans la gestion du multiplexage des données issues des capteurs et des calculateurs sur le bus de telle sorte que les contraintes temporelles de chacun des signaux soient satisfaites [Kocik, 2000].

1.4.2 Architecture logicielle

L'architecture logicielle d'un système temps réel (Figure 1.7) consiste essentiellement en un exécutif temps réel ayant le rôle d'intermédiaire avec la couche matérielle ainsi qu'un programme applicatif englobant les fonctions permettant de contrôler le système temps réel.

— ***Exécutif temps réel***

L'exécutif temps réel est composé d'un noyau temps réel fournisseur d'un ensemble de services de base. L'ordonnanceur constitue la partie principale d'un exécutif temps réel, il a pour rôle d'attribuer les différentes ressources aux tâches du système temps réel. La politique d'attribution des tâches aux ressources peut se faire selon différentes approches (cf. sous-section 1.6.2).

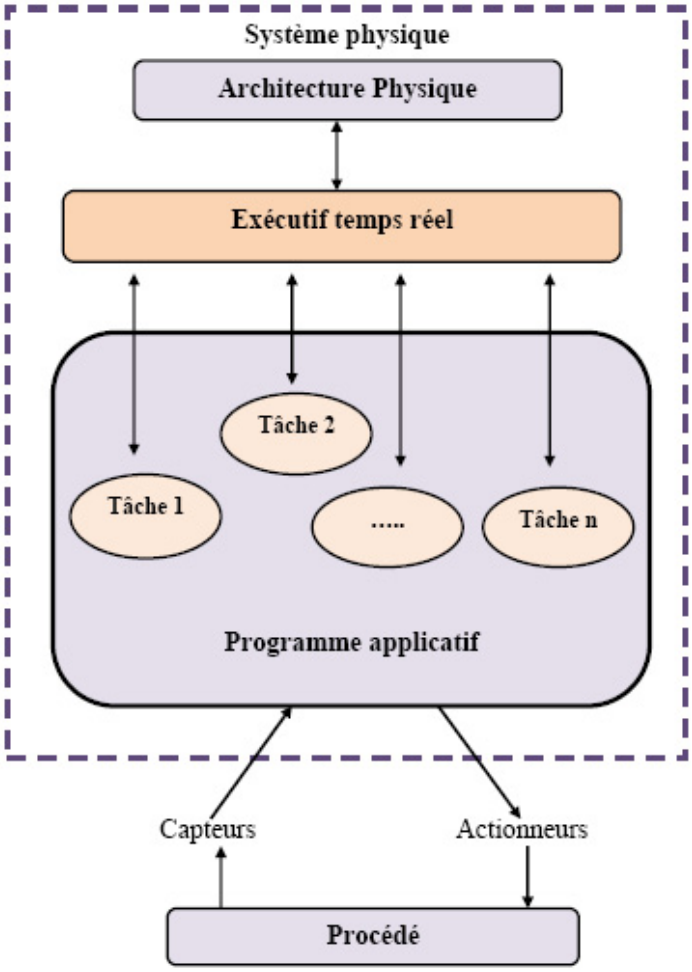

FIGURE 1.7: Architecture logicielle d'un système temps réel [Dorin, 2010].

— ***Programme applicatif***

Le programme applicatif est constitué d'un ensemble d'entités distinctes appelées 'tâches' (cf. sous- section 1.5.1) chacune possédant une fonctionnalité donnée telle que la réalisation d'un calcul, le traitement des entrées/sorties . . . etc.

Des modules tels que le module de gestion de fichiers ou le module de gestion de réseaux complètent le noyau afin de faciliter la conception d'une application. Enfin, il peut y avoir d'autres facilités proposées par un système d'exploitation telles que l'utilisation d'une interface.

1.5 Systèmes distribués embarqués temps réel

La propriété 'distribués' des systèmes embarqués temps réel signifie que le système est constitué de plusieurs processeurs et que les opérations ou les tâches formant l'application globale sont réparties sur ces mêmes processeurs. L'objectif final est atteint grâce à une coopération et une coordination entre les différentes tâches.

1.5.1 Tâche temps réel

Un programme temps réel est composé d'un ensemble d'entités appelées *tâches temps réel*, chacune correspondant à un traitement ayant une fonctionnalité bien précise au niveau du système. Chaque tâche doit répondre à un stimulus à l'intérieur d'un délai. A son tour, une tâche peut entraîner le déclenchement d'une autre tâche.

— ***Modèles de tâches***

Une tâche informatique peut être modélisée selon deux points de vue. Le modèle comportemental définit l'état de la tâche à un instant donné de l'exécution tandis que le modèle canonique en caractérise les propriétés temporelles.

— *Modèle comportemental (conceptuel)*

Une tâche peut être exécutée une multitude de fois durant la vie d'un système. Une instance d'une tâche est une exécution ou occurrence de celle-ci. Le comportement d'une tâche est représenté par l'automate à états fini (Figure 1.8) décrivant les transitions possibles d'un état à un autre.

Une tâche (supposée déjà créée) est initialement dans un état 'Inactif'. Lorsqu'à un instant t la tâche se réveille, elle passe en état 'Prêt'. Une fois élue par l'ordonnanceur, pour qu'elle soit exécutée, la tâche passe dans l'état 'exécution'.

Depuis l'état 'Exécution', plusieurs changements d'état sont atteignables :

— Une tâche peut être interrompue par une autre tâche plus prioritaire et passe en état 'Prêt' ;

— elle peut avoir besoin d'une ressource non disponible et se mettre de ce fait en état 'Bloqué', la mise à disposition de la ressource la ramène à l'état 'Prêt' ;

— la tâche peut être stoppée en cours d'exécution, son état courant enregistré, et retourne à l'état 'Prêt'.

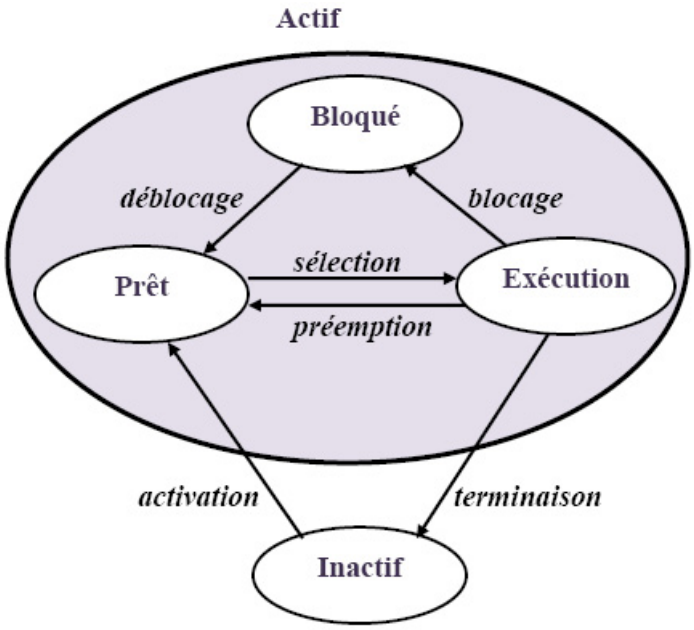

FIGURE 1.8: Etats et transitions d'une tâche [Meumeu Yomsi, 2009].

La tâche dans l'état 'Inactif' est celle qui soit n'existe pas encore car elle attend son activation, soit qui a terminé son exécution et attend une prochaine activation. La sélection est opérée par l'ordonnanceur tandis que l'évènement 'terminer' est produit par la tâche elle-même.

A l'arrivée de l'évènement *activer* d'une des tâches, l'ordonnanceur compare sa priorité avec celle de la tâche en exécution et celles de toutes les tâches qui sont dans l'état 'Prêt'. Si la priorité de cette tâche est supérieure, il produit un évènement *préempter*, sinon il ne fait rien. De même, à l'arrivée de l'évènement *terminer* de la tâche en cours, l'ordonnanceur compare les priorités des tâches dans l'état 'Prêt' et produit l'évènement *sélectionner* pour la tâche la plus prioritaire.

— *Modèle canonique*

Le modèle de tâches, introduit par Liu et Layland [Liu and Layland, 1973] pour l'étude des tâches périodiques est le plus utilisé lors de la conception des systèmes temps réel. Dans ce

modèle, une tâche ti est définie par un ensemble de paramètres :

r_i (ready time ou release time) : date à laquelle la tâche t_i peut commencer son exécution ;

S_i (Start time) : date à laquelle la tâche t_i commence son exécution sur le processeur, appelée donc date de début d'exécution ;

E_i (End time) : date à laquelle la tâche termine son exécution sur le processeur, appelée donc date de fin d'exécution ;

RT_i (Response time) : Ceci représente $E_i - S_i$;

C_i (Computing time) : durée d'exécution de la tâche t_i . Ce paramètre est considéré dans la majorité des travaux comme le pire cas des temps d'exécution (WCET pour Worst Case execution Time) sur le processeur où elle sera exécutée ;

D_i (Deadline) : l'échéance relative au release time de la tâche ou la date au plus tard, elle représente l'instant auquel l'exécution d'une tâche doit être terminée et dont le dépassement provoque une transgression de la contrainte temporelle.

($D_i - C_i$) (Latence) : durée pendant laquelle une tâche dans un système peut être retardée sans que sa date de fin d'exécution ne dépasse son échéance ;

$P_i =$: période de t_i ;

L_i : (Laxity) : la laxité d'une tâche t_i , elle représente le temps restant avant l'occurrence de sa date de début d'exécution ou de reprise au plus tard. Si à un instant donné, ce paramètre est nul, la tâche correspondante doit être impérativement exécutée à cet instant et sans interruption sinon son échéance sera inévitablement dépassée.

Les tâches étant périodiques :

La date de la k^{ime} activation de la tâche t_i est donnée par : $r_{ik} = r_{i1} + (k-1)P_i$;

Elle doit terminer son exécution avant la date $d_{ik} = r_{i1} + (k-1)P_i + D_i$.

— ***Nature d'une tâche***

La loi d'arrivée d'une tâche définit sa nature, on distingue trois grandes classes de tâches :

Tâche périodique

Une tâche périodique t_i (Figure 1.9) est une tâche dont les inter-arrivées sont strictement périodiques. Cela veut dire qu'elle doit s'exécuter une fois chaque période P_i.

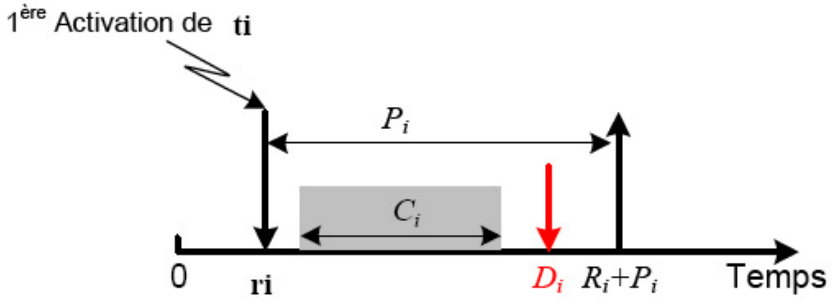

FIGURE 1.9: Modèle d'une tâche temps réel périodique [Meumeu Yomsi, 2009].

Tâche apériodique

Une tâche apériodique possède souvent des délais critiques mais pas de période ni intervalle minimal séparant deux instances. Une telle tâche est déclenchée par un évènement pouvant être produit à tout instant.

Tâche sporadique

Une tâche sporadique t_i [Kim and Naghibdadeh, 1980, Mok, 1983] est caractérisée par une loi d'arrivée sporadique, où la période d'activation n'est pas connue a priori, par contre un intervalle minimal de temps T_i séparant deux activations successives est donné.

Les dates d'activation des différentes instances d'une tâche sporadique ne peuvent pas être déterminées. Une instance peut s'activer à n'importe quelle date après la date T_i relative à la date d'activation de l'instance précédente. En résumé, trois paramètres temporels caractérisent une tâche sporadique : Une durée d'exécution C_i, un délai critique Di et un intervalle minimal T_i séparant deux occurrences de t_i.

Concrètes/Non-concrètes

Si un scénario d'activation particulier est imposé aux tâches, elles sont dites concrètes. Par contre, si les dates de première activation ne sont pas connues a priori, les tâches sont considérées comme non-concrètes.

Synchrones/Asynchrones

Dans le cas synchrone, un scénario d'activation est imposé et répété durant toute la vie d'un système, l'exécution des actions/lectures est réalisée à des intervalles réguliers fixés par un timer (time driven system). En revanche, dans le cas asynchrone, le système attend des sollicitations pour réagir, il est piloté par des évènements (event driven system).

— **Dépendance et précédence**

Dans un système temps réel, une tâche peut être indépendante ou dépendante dans le sens où elle produit des données pour une autre tâche qui les consomme. Une dépendance impose une précédence entre la fonction qui produit et celle qui consomme. Les fonctions deviennent des tâches dès qu'on les a caractérisées temporellement.

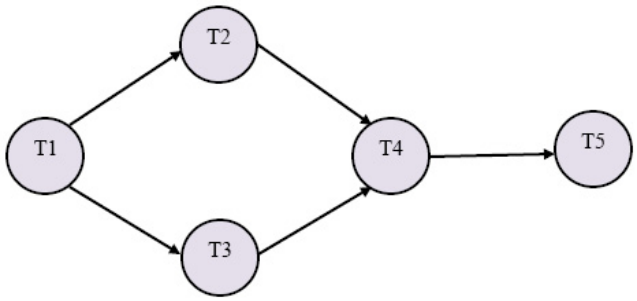

FIGURE 1.10: Exemple de graphe de précédence.

Une contrainte de précédence entre deux tâches impose un ordre entre ses tâches [Orozco et al., 1997]. Le terme de 'précédence' désigne un ordre entre deux tâches (Figure 1.10) sans qu'il y ait transfert de données alors que le terme de 'dépendance' se dit lorsqu'une tâche a besoin du résultat d'exécution d'une autre tâche pour pouvoir être exécutée à son tour, ce qui impose évidemment, une précédence entre ces deux tâches. La dépendance entre tâches est souvent modélisée par un graphe orienté dans lequel les nœuds représentent les tâches et les arcs les relations de dépendance entre les tâches.

1.5.2 Ressource

Une ressource est un moyen technique ou humain destiné à être utilisé pour la réalisation d'une tâche et disponible en quantité limitée. On parle de ressource *renouvelable* lorsqu'après l'avoir allouée à une ou plusieurs tâches, elle est de nouveau disponible en même quantité, les personnes et l'équipement en général en sont des exemples. Tandis que dans le cas contraire, la ressource est dite *consommable*, par exemple les matières premières ou bien les budgets. Qu'elle soit renouvelable ou consommable, la disponibilité d'une ressource peut varier dans le temps. La gestion d'un tel paramètre s'avère importante notamment dans l'activité d'ordonnancement temps réel.

1.5.3 Modes d'exécution

On distingue deux principaux modes d'exécution :

— ***Non-préventif*** : L'ordonnanceur ne peut pas interrompre l'exécution d'une tâche possédant le processeur en faveur d'une autre tâche. Il doit donc attendre à ce que la tâche en cours se termine avant de commencer l'exécution de toute autre tâche. Si une tâche non-préventive est interrompue, son exécution doit être reprise de nouveau depuis le début.

— ***Préventif*** : L'exécution d'une tâche peut être interrompue par une autre tâche plus prioritaire. Son exécution est reprise ultérieurement.

1.6 Ordonnancement temps réel

1.6.1 Problématique de l'ordonnancement

Disposant d'un ensemble de tâches et d'un ensemble de processeurs, la problématique de l'ordonnancement revient à définir comment agencer les exécutions des tâches sur chaque processeur. C'est l'ordonnanceur, principal gérant d'un système temps réel, qui se charge de faire respecter les contraintes temporelles des tâches. L'ordonnancement consiste donc à choisir une politique pour l'affectation des tâches (en concurrence) aux différents processeurs durant la vie du système afin

d'optimiser d'autres critères temporels tels que le temps de réponse ou la longueur d'ordonnancement.

1.6.2 Algorithmes d'ordonnancement

Plusieurs classifications peuvent être établies quant aux approches d'ordonnancement [Cardeira, 1994] :

— ***Monoprocesseur/Multiprocesseur***

Lorsque l'architecture ne dispose que d'un seul processeur, on dit que le problème d'ordonnancement est monoprocesseur. Si plusieurs processeurs sont disponibles, alors le problème est multiprocesseur.

— ***En ligne/Hors ligne***

Un ordonnancement est hors ligne si la séquence d'ordonnancement définissant l'ordre dans lequel les tâches doivent être exécutées, est établie avant le lancement de l'application et ce pour être répétée à l'infini. La séquence est générée par un algorithme et implantée dans une table qui sera consultée par un ordonnanceur. Une telle approche est efficace lorsque tous les paramètres des tâches sont connus. En revanche, l'adaptation à l'environnement devient difficile [Funk et al., 2001].

L'autre approche utilisée est celle en ligne où la séquence est construite dynamiquement. Un ordonnancement est dit en ligne s'il se fait au fur et à mesure que les tâches arrivent dans le système. Dans ce cas, l'algorithme d'attribution de tâches aux ressources se base sur les paramètres temporels des tâches ainsi que sur l'occurrence d'évènements aléatoires, à l'instant de décision, afin de choisir la tâche à ordonnancer parmi les tâches présentes dans la file d'ordonnancement.

— ***A priorité fixe/dynamique***

Un autre classement des algorithmes d'ordonnancement temps réel peut se faire selon leur utilisation des priorités pour choisir quelle tâche doit être ordonnancée. Une première approche dite à 'priorités fixes' consiste à attribuer à chaque tâche une priorité avant l'exécution. Les algorithmes d'ordonnancement monoprocesseur Rate Monotonic (RM) [Liu and Layland, 1973]

et Deadline Monotonic (DM) [Leung and Whitehead, 1982] utilisent cette solution. D'autres solutions permettent de fixer les priorités au niveau des instances de tâches comme par exemple Earliest Deadline First (EDF) [Liu and Layland, 1973]) ou d'avoir des priorités complètement dynamiques tel que Least Laxity First (LLF) [Dertouzos and Mok, 1989].

— ***Préemptif/non-préemptif***

Selon la politique de choix, l'ordonnanceur peut stopper l'exécution d'une tâche en cours au profit d'une autre, on parle alors de préemption. Cet arrêt d'exécution entraîne une sauvegarde de l'état courant de la tâche, afin de pouvoir la reprendre plus tard. A l'inverse, un algorithme d'ordonnancement non préventif ne permet aucune préemption, une instance active ne peut donc être arrêtée.

— ***Oisif/Non oisif***

Un algorithme d'ordonnancement oisif est un algorithme d'ordonnancement où un processeur peut être inactif alors qu'une ou plusieurs tâches sont susceptibles d'être exécutées. A l'inverse, dans un ordonnancement non oisif, le fonctionnement se fait sans insertion de temps creux.

La majorité des algorithmes d'ordonnancement temps réel sont non-oisifs, ce qui permet de simplifier la conception de l'algorithme d'ordonnancement. En effet, la seule décision à prendre par l'algorithme d'ordonnancement est de choisir la tâche à exécuter, et non de décider si une tâche doit être exécutée ou pas.

— ***Optimal/Non Optimal***

Un algorithme d'ordonnancement est dit optimal [Zhu et al., 1995] pour un problème donné s'il permet de produire un ordonnancement respectant toutes les contraintes lorsqu'un tel ordonnancement existe. Si l'algorithme optimal ne trouve pas de solution alors aucun autre algorithme ne pourra le faire. Quant à l'algorithme d'ordonnancement non-optimal, il vise à trouver des solutions approchées.

1.6.3 Exactitude du système

Lorsqu'un système temps réel est conçu, il est impératif de s'assurer de l'exactitude logique des résultats ainsi que du respect des contraintes temporelles [Cormick et al., 2011] :

— ***Exactitude de fonctionnement*** : Il s'agit de vérifier que le système produit un résultat correct pour chaque ensemble de données en entrée. Pour cela, des techniques de vérification telles que le test et la preuve formelle sont utilisées.

— ***Exactitude du comportement temporel*** : Dans ce cas, la vérification concerne le non dépassement des échéances requises de la part des temps d'exécution des tâches. Cette analyse du comportement temporel est appelée analyse d'ordonnançabilité.

1.6.4 Ordonnançabilité/Faisabilité

L'étude de l'ordonnancement des systèmes temps réel se focalise principalement sur deux éléments essentiels :

— ***La faisabilité*** : En prenant en compte les tâches, les processeurs ainsi que les contraintes, le but est de dire s'il existe un ordonnancement qui satisfait toutes les échéances. Un système de tâches est par conséquent dit faisable s'il existe un algorithme d'ordonnancement permettant d'ordonnancer ce système de tâches sans aucune violation d'échéances. La condition que doit vérifier l'ensemble des tâches pour être faisable est appelée *condition de faisabilité*.

— ***L'ordonnançabilité*** : En considérant également les tâches, les processeurs, les contraintes, le but est de vérifier, cette fois-ci, si cet ensemble de tâches est ordonnançable relativement à un algorithme donné. Un système de tâches est dit ordonnançable si un ordonnancement existe permettant de satisfaire toutes les contraintes temps réel. La condition que doit vérifier l'ensemble des tâches pour être ordonnançable est appelée *condition d'ordonnançabilité*.

1.6.5 Analyse d'ordonnançabilité

Plusieurs approches d'analyse d'ordonnançabilité existent [Kermia, 2009] :

— ***Approche analytique***

Cette approche consiste à identifier le ou les pires cas d'exécution et à déterminer analytiquement une condition d'ordonnançabilité des tâches. Dans cette approche, peut être utilisé :

— *le facteur d'utilisation des tâches* : dans certains cas, le facteur d'utilisation des tâches permet de conclure si un ensemble de tâches est ordonnançable ou pas ;

— *le pire temps de réponse* : le pire temps de réponse d'une tâche est calculé puis comparé à son échéance. Si une tâche a un pire temps de réponse inférieur ou égal à son échéance relative alors elle est ordonnançable ;

— *Le temps de demande du processeur* : c'est le temps requis par le processeur pour exécuter un ensemble de tâches activées puis terminées dans un intervalle donné. Si ce temps est inférieur à la longueur de l'intervalle considéré alors les tâches activées dans cet intervalle sont faisables, sinon elles ne le sont pas.

— ***Approche par simulation***

La simulation consiste à modéliser le comportement du système à l'aide d'un formalisme adéquat puis à l'exécuter avec un outil de simulation sur des scénarios définis par l'utilisateur. Dans le cas des systèmes temps réel, il s'agit de simuler le déroulement de l'ordonnancement du système pendant 'une période' et de vérifier que toutes les instances de tâches respectent leurs échéances.

— ***Approche par model-checking***

L'approche par model-checking permet une exploration exhaustive de l'espace d'états du système. Les formalismes généralement utilisés sont les systèmes de transitions temporels tels que les automates ou encore les réseaux de Petri.

1.7 Conclusion

Dans ce chapitre, nous avons présenté les systèmes embarqués temps réel. En plus des fonctionnalités offertes, ces systèmes doivent satisfaire des contraintes temporelles dont le non respect, notamment dans le cas des systèmes à contraintes dures, peut être catastrophique. Le développement des systèmes embarqués temps réel est un processus délicat d'où l'intérêt grandissant pour des méthodologies prenant en charge les exigences entourant ce type de systèmes. Le chapitre suivant passe en revue les différentes approches permettant le développement de tels systèmes.

Chapitre 2

Conception des systèmes embarqués temps réel

Résumé

Notre travail est centré sur le développement des systèmes embarqués temps réel. Aussi, cette partie est-elle un état de l'art en matière de méthodologies prenant en charge le développement de tels systèmes. Un récapitulatif des méthodologies, moyennant quelques critères, est présenté à la fin de ce chapitre.

Sommaire

2.1 Introduction **30**

2.2 Exigences posées par les systèmes temps réel **30**

2.2.1 Exigences fonctionnelles 30

2.2.2 Exigences non fonctionnelles 31

2.3 Approches de développement **31**

2.3.1 Génie logiciel classique 31

2.3.2 Co-design 34

2.3.3 Méthodologies orientées UML 36
2.3.4 Approches à base de composants 38
2.3.5 Ingénierie dirigée par les modèles 39
2.3.6 Techniques formelles 44
2.4 Méthodolgie AAA (Adéquation Algorithme Architecture) 46
2.5 Langages de développement 48
2.6 Quelques critères de comparaison 49
2.7 Conclusion 50

2.1 Introduction

Nombreuses sont les applications de la vie quotidienne qui sont concernées par le domaine du temps réel, nous en citerons le guidage et la navigation (de bateaux, d'avions, de trains ou d'automobiles), le contrôle de processus industriels, la robotique, les télécommunications (téléphone portables, satellites), les transactions bancaires et enfin les jeux. Pour de tels systèmes, certaines défaillances peuvent engendrer des pannes ou des pertes de données menant à des situations catastrophiques. Pour cela, le problème de la fiabilité des systèmes temps réel demeure posé depuis des décennies.

Traditionnellement, l'utilisation de langages de bas niveau était de rigueur dans le développement des systèmes temps réel et ce afin de garantir le contrôle sur leur comportement. En revanche, leur complexité devenue de plus en plus grande à travers leur distribution d'une part et leur embarquabilité d'autre part, a rendu leur développement plus fastidieux. C'est ainsi que se pose la question de méthodologies de développement des systèmes distribués embarqués temps réel, aspirant à satisfaire les contraintes inhérentes à ce type de systèmes.

2.2 Exigences posées par les systèmes temps réel

Dans un système, on distingue deux types d'exigences [Hull et al., 2004] :

2.2.1 Exigences fonctionnelles

Ces exigences expriment les différentes fonctionnalités que le système devra satisfaire. En plus des fonctionnalités attendues, sont également prises en compte des propriétés générales telles que l'absence de blocage non désiré, l'absence de fonctionnement cyclique infini non désiré ou encore la possibilité de revenir à l'état initial.

2.2.2 Exigences non fonctionnelles

Elles correspondent aux critères de qualité attendus du système, particulièrement en termes de performances temporelles tels que les délais maximum et minimum à satisfaire.

2.3 Approches de développement

Les méthodologies de développement des systèmes embarqués temps réel furent et sont toujours un axe de recherche très actif. L'objectif est de proposer des méthodologies pouvant prendre en compte les exigences de ce type de système. Nous proposons une classification des approches de développement comme suit :

2.3.1 Génie logiciel classique

Dans ce cas, le développement d'une application suit un cycle comprenant essentiellement une étape d'expression des besoins, une étape de spécification, une étape de conception, une étape d'implémentation, une étape d'intégration et de test, une étape d'installation et de vérification et enfin une étape de maintenance.

La spécification est une description abstraite du futur système, elle décrit ce que ce dernier doit faire sans pour autant donner des détails de la manière dont il le fait. La conception aboutit à la création de modèles conceptuels qui seront par la suite implémentés dans la phase d'implémentation.

L'ordre des étapes définit un modèle de développement particulier. Néanmoins, ces étapes font partie de tous les cycles de développement de systèmes indépendamment de la nature, du domaine, de la taille et de la complexité du système à développer, aussi, les systèmes distribués embarqués temps réel en sont-ils concernés. Il existe divers modèles de développement d'une application, nous en citerons le modèle en cascade [Saksena and Karvelas, 2000], le développement incrémental [Hirsh, 1985] et le modèle en V [Sommerville, 1988], ce dernier étant très communément utilisé dans le développement des systèmes temps réel en général.

Le modèle en V est constitué de deux branches : La première branche est un modèle en cascade dont le rôle est d'analyser les besoins, de spécifier le logiciel, de le concevoir et enfin de l'implémenter. La deuxième branche quant à elle, consiste à assembler les constituants et à effectuer un ensemble de tests afin de valider le système.

Notons que les étapes de la deuxième branche utilisent celles de la première branche pour établir les validations ainsi que les vérifications nécessaires. La validation d'une étape de conception entraine la remontée vers une étape de validation de niveau hiérarchique supérieur tandis que la non validation d'une étape de conception entraine la réexécution de cette étape et de certaines ou éventuellement de toutes les étapes inférieures ainsi que leur revalidation [Gaudel et al., 1988, Lenoir, 1999, .Perez, 1990].

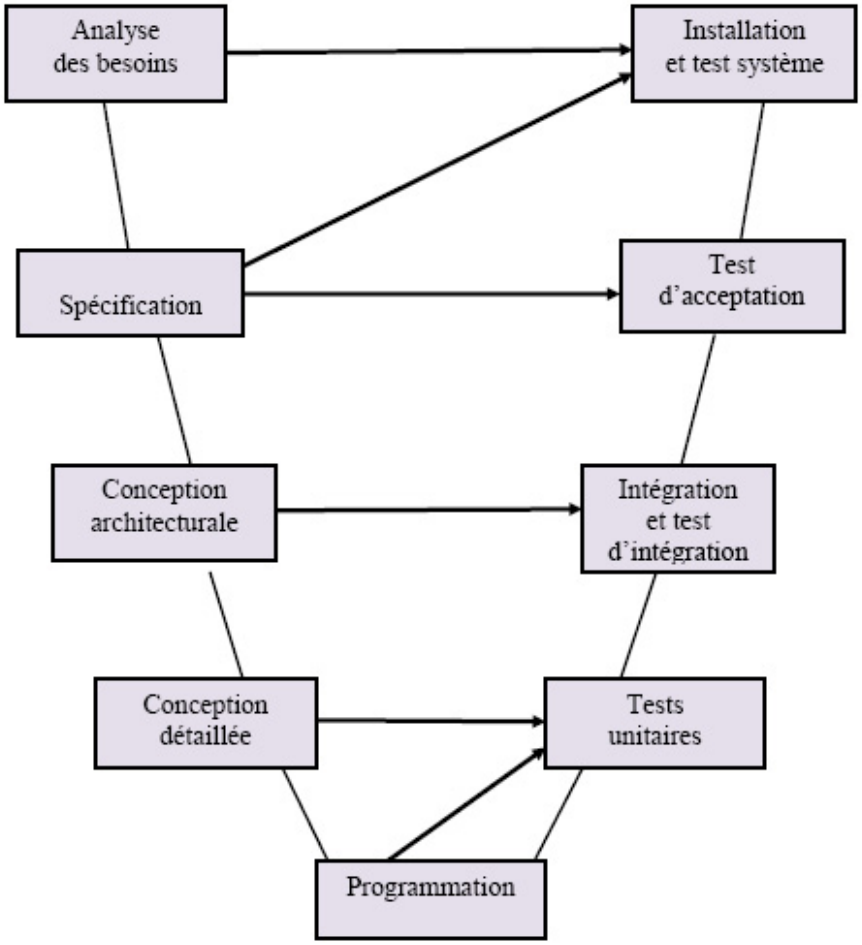

FIGURE 2.1: Le modèle de développement en V.

Trois étapes de validation complètent le modèle à savoir les tests unitaires, l'intégration et les tests d'intégration et enfin la validation du système. Une des raisons de l'utilisation de ce modèle de développement pour les systèmes embarqués temps réel est le fait que beaucoup d'importance est accordée à la partie 'tests', étape très pertinente pour ce type de systèmes qualifiés de 'critiques'.

Par ailleurs, dans cet axe de recherche, on distingue deux sous-axes : le premier a pour but l'enrichissement d'étapes du cycle de développement tandis que le second concerne l'amélioration des transitions entre les différentes étapes.

— ***Enrichissement intra-étape***

Dans ce cas, il s'agit de se concentrer sur une des étapes du développement telles que la spécification, la conception ou l'implémentation avec l'objectif d'y apporter des modifications. Les travaux menés par Carlson [Carlson, 2002] ont pour objectif d'exprimer les contraintes temps réel dans un langage de spécification. L'introduction du temps dans les diagrammes de classe UML est l'objet du travail présenté par Roubtsova [Roubtsova et al., 2002]. L'intégration des techniques d'ordonnancement temps réel dans une conception orientée objet [Saksena and Karvelas, 2000] est également une approche d'amèlioration de l'étape de conception où le langage utilisé est UML-RT [Selik et al., 1994]. L'étape d'implémentation est prise en compte à travers la traduction automatique d'un programme ESTEREL vers un code microprocesseur [Andre and Peraldi, 1992].

— ***Enrichissement inter-étapes***

Dans ce cas, l'intérêt est porté sur le passage d'une étape à une autre plutôt que sur une étape donnée. La génération du code à partir d'un modèle conceptuel établi en SDL (Specification and Description Language) ainsi que son optimisation est le travail réalisé par Langendoerfer [Langendoerfer and koenig, 1999]. UML et SDL sont combinés [Kuusela and Kettunen, 1993] où des règles sont définies permettant la transformation automatique d'un sous-ensemble formalisé d'OMT noté OMT* [Awad et al., 1996] vers SDL. Un profil UML (SDL with UML) pour SDL est défini [ITU-T, 1999], il permet d'utiliser un ensemble de concepts d'UML en

conjonction avec SDL.

2.3.2 Co-design

Les systèmes embarqués temps réel étant avant tout des systèmes hybrides (matériel et logiciel), leur développement doit prendre en compte les deux aspects. Le processus de conception de tels systèmes est caractérisé par une partie conjointe matériel/logiciel (Figure 2.2) :

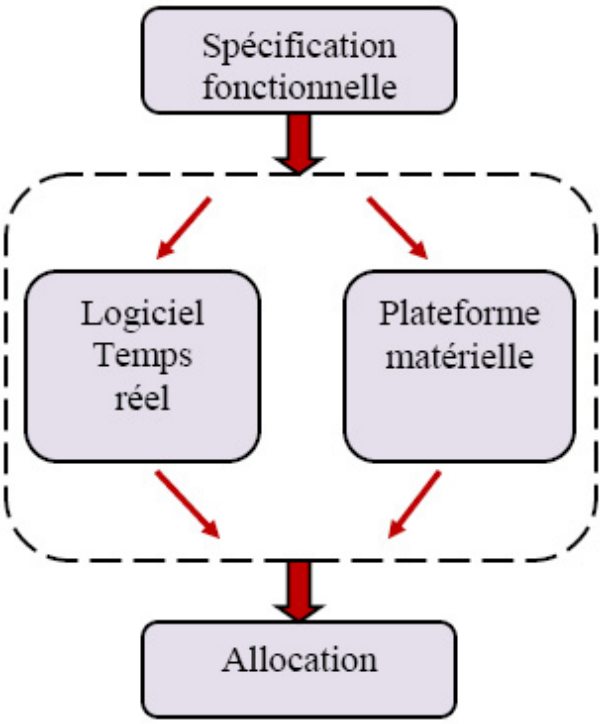

FIGURE 2.2: Processus simplifié de codesign.

Spécification fonctionnelle

Prenant comme base le cahier des charges, cette étape consiste en une description des fonctionnalités du système. Le standard SysML [Object Management Group, 2007a] qui applique l'IDM à l'ingénierie système définit deux diagrammes de blocs : *Block Definition Diagram* et *Internal Block Diagram* appropriés à la spécification fonctionnelle. Le concept de *block* en SysML est similaire à celui de *composant* dans UML.

Partionnement

Pour chaque composant matériel identifié lors de l'étape de spécification fonctionnelle, le partitionnement consiste à fixer un choix d'implémentation. On distingue trois flots possibles (Figure 2.3) :

— La réutilisation d'IPs : Il s'agit de réutiliser des composants déjà conçus et validés qu'on appelle 'intellectual properties' ;

— La conception de circuits matériels : concerne aussi bien les circuits imprimés que les circuits configurables ;

— La conception de logiciels : consiste en deux flots fortement couplés qui sont le logiciel embarqué et sa plateforme d'exécution.

L'objectif de l'étape de partitionnement est de trouver le meilleur compromis entre les différents flots tout en minimisant les coûts et en maximisant les performances.

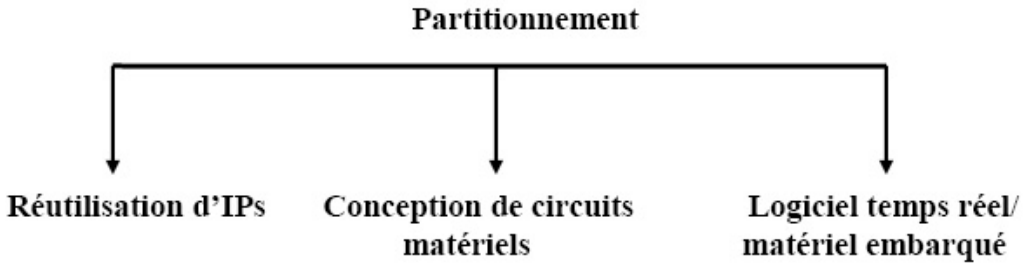

FIGURE 2.3: Flots de conception.

Deux flots parallèles permettent la conception du logiciel et la réalisation de la plateforme d'exécution. Les deux flots utilisent des environnements de développement différents, ce qui rend la communication entre les deux équipes délicate. Malgré l'existence de points de synchronisation, des erreurs peuvent avoir lieu et tard dans le processus de développement.

Allocation

L'allocation consiste à affecter à chaque partie du logiciel le matériel supportant son exécution.

Ptolemy II et Metro II sont des exemples d'approches de co-design :

— **Ptolemy II**

Ptolemy II [Lee and Neuendorffer, 2007] est un environnement de modélisation d'un système en termes de composants hétérogènes indépendants de toute plateforme d'exécution. La communication entre composants est le point central de l'approche, elle est définie par des sémantiques appelées 'Modèles de Calculs' (MoC). La modélisation se base sur la notion d'*acteurs* qui définissent des composants en Java et de *directeurs* qui définissent la sémantique de communication entre les acteurs. L'environnement Ptolemy permet la modélisation fonctionnelle de haut niveau.

— **Metro II**

Metro II [Davare et al., 2007] est un environnement de conception basé sur la technique Platform-based design [Sangiovanni-Vincentelli et al., 2004]. Le langage SystemC est utilisé pour la description du comportement interne des composants. Disposant des spécifications textuelles ou graphiques de l'application, de l'architecture matérielle et des contraintes de conception, Metro II suit un processus itératif afin d'aboutir à un modèle du système respectant le cahier des charges. Cette approche permet également l'intégration des IPs (Intellectual Properties) et donc la possibilité de réutilisation de modèles.

2.3.3 Méthodologies orientées UML

UML (Unified Modeling Language) [Object Management Group, 2011b], préconisé par l'OMG comme langage de modélisation, s'articule autour d'un ensemble de diagrammes permettant de décrire un système selon plusieurs points de vue. Il existe deux catégories de diagrammes, la première concerne les diagrammes structurels dont le rôle est de décrire l'architecture du système tandis que la seconde concerne les diagrammes comportementaux qui permettent la description de l'aspect dynamique du système. UML est un langage applicable dans plusieurs domaines comme il peut également être utilisé avec différentes plateformes d'implémentation dont CORBA (Common Object Request Broker Architecture), J2EE (Java 2 Enterprise Edition) ou encore Microsoft.NET.

UML est un langage de modélisation adapté à de nombreux domaines d'application. Cependant, il existe des domaines spécifiques auxquels ce langage n'est pas complètement approprié, en l'occurrence, celui du temps réel. La notion de 'profil', apparue dans le standard UML 1.3, a été créée afin de permettre la spécialisation d'UML à différents contextes. L'OMG propose un ensemble de profils standardisés adaptés à différents domaines, nous en citerons :

— **MARTE**

MARTE (Modeling and Analysis of Real Time and Embedded systems) [Object Management Group, 2007b] est un profil UML standard de l'OMG, il a pour but la spécification et la validation des propriétés temps réel d'un système. Pour cela, il offre le langage VSL (Value Specification Language) sous la forme d'une extension du langage déclaratif OCL (Object Constraint Language) afin de supporter les contraintes comportementales. Le profil MARTE est constitué essentiellement de trois paquetages à savoir le paquetage *foundation* prenant en charge, entre autres, la modélisation des propriétés non fonctionnelles, le paquetage *design* qui permet la modélisation des applications ainsi que des plateformes d'exécution et enfin le paquetage *analysis* qui fournit les mécanismes permettant d'annoter les modèles à des fins d'analyse. En plus, il existe un quatrième paquetage *annexes* qui contient, entre autres, la bibliothèque des modèles prédéfinis $MARTE_{Model}$ Library et le langage de spécification de valeur VSL (Value Specification Language).

— **TURTLE**

TURTLE (Timed UML and RT-LOTOS Environment) est un profil UML, il spécialise le langage UML en prenant en charge les systèmes temps réel. TURTLE est basé sur l'algèbre des processus temporisés RT-LOTOS (Real Time LOTOS) [Courtiat et al., 2000] construite elle-même sur la technique de description formelle LOTOS (Language of Temporal Ordering of Sequences) [Bolognesi and Brinksma, 1989].

— **RT-LOTOS**

RT-LOTOS (Real Time LOTOS) est une extension de LOTOS, il y intègre trois opérateurs temporels :

— *delay(d)* : est l'opérateur de délai, il permet de retarder un processus d'un temps d ;

— *latency* : est l'opérateur de latence, il permet de retarder un processus d'un certain temps appartenant à l'intervalle de latence [0 :1] ;

— *aT* : est l'opérateur de restriction temporelle, il limite le temps pendant lequel une action observable a peut être offerte à son environnement.

L'approche est supportée par un ensemble d'outils TTOOL (TURTLE Toolkit), celui-ci inclut un éditeur graphique de diagrammes, un analyseur syntaxique, un générateur de code ainsi que des outils de visualisation et d'analyse des résultats de simulation et de validation.

— **PEARL**

La spécification PEARL [Gumzej et al., 2006] permet de définir un système embarqué temps réel par l'allocation d'une configuration logicielle à une configuration matérielle. La spécification des communications ainsi que celle des architectures tolérantes aux pannes est possible. Cependant, les concepts définis par la spécification PEARL ne permettent pas de spécifier des contraintes temporelles complexes [Krichen, 2013]. Par ailleurs, en se basant sur les concepts introduits dans la spécification PEARL, un patron de reconfiguration UML pour les systèmes embarqués [Gumzej et al., 2009] est décrit par un profil UML-RT. Il permet la modélisation et la gestion des reconfigurations des architectures logicielles et matérielles.

2.3.4 Approches à base de composants

Le développement à base de composants est une approche qui a considérablement été adoptée depuis plus d'une dizaine d'années. Elle permet essentiellement de :

— Réduire la complexité des systèmes et ce en les concevant par assemblage de composants préexistants ;

— Accélérer le développement des systèmes grâce à la réutilisation des composants dans différentes applications ;

— Faciliter l'évolution des systèmes vu les structures modulaires engendrées par une telle approche.

L'approche par composants se trouve être une approche très appropriée à la conception des systèmes en général. Néanmoins, les modèles de composants connus tels que CORBA/CCM de l'OMG, (D)COM/COM+ de Microsoft et Enterprise JavaBeans de SUN ne sont que très rarement utilisés pour la conception de systèmes temps réel. Ceci est dû à leur manque de support pour les propriétés temporelles. Il existe également d'autres modèles de composants dont la technologie AutoComp [Sandstrom and Fredriksson, 2004], le modèle RTCOM du projet ACCORD [Tesanovic et al., 2003], KOALA [Van Ommering and Van der Linden, 2000], SaveCCM du projet SAVE [Hansson et al., 2004], PECOS [Muller and Stich, 2001] ... etc.

— **AADL**

AADL (Architecture Analysis and Design Language) [Feiler et al., 2006] est un langage de description d'architecture destiné aus systèmes embarqués. Il est un standard international publié par la SAE (Society of Automative Engineers). AADL repose essentiellement sur la notion de *composant* pour lequel une interface précise est définie. La description de l'interface est séparée de son implémentation. L'architecture d'un système y est décrite comme un ensemble de composants hiérarchisés, composés et inter-connectés.

On distingue trois types de composants : les composants logiciels, les composants matériels et les composants composites. D'autre part, la communication entre composants est décrite à l'aide de ports et de connexions permettant plusieurs types de communication. Il est possible de compléter la sémantique de AADL en ajoutant des éléments composant un modèle.

2.3.5 Ingénierie dirigée par les modèles

La complexité croissante des logiciels a créé des exigences dans le domaine du génie logiciel. En effet, la séparation des préoccupations ainsi que la multiplicité des plateformes a renforcé le passage au paradigme des modèles. Force est de constater que dans le domaine de l'ingénierie, la réalisation d'un

système passe par une bonne compréhension de celui-ci à toutes les étapes du cycle de développement. Une telle compréhension est d'autant plus réussie que lorsque la modélisation du système a lieu. En effet, modéliser un système permet de le comprendre, de l'analyser et éventuellement de l'améliorer.

L'IDM ou Model Driven Engineering (MDE) en anglais, est une approche du génie logiciel qui s'articule autour du concept de *modèle* [Bézivin, 2005] en se plaçant d'abord à un niveau abstrait de description et en affinant à travers des niveaux jusqu'à atteindre celui de la programmation. C'est ainsi que l'IDM se démarque par l'usage systématique des modèles tout au long du développement logiciel.

L'architecture dirigée par les modèles (MDA) [Object Management Group, 2003] dérivée de l'IDM par l'OMG, est basée sur les modèles, ceux-ci sont décrits dans un langage de modélisation dont le méta-modèle est exprimé en MOF (Meta Object Facility) [Object Mannagement Group, 2006]. L'approche MDA se base sur les notions de modèle, méta-modèle et transformation de modèles. Via les modèles, plusieurs facettes d'une application peuvent être décrites tout en faisant abstraction des détails d'implémentation.

— ***Modèle***

  Un modèle est une description abstraite d'un système, il peut être textuel et/ou graphique ou encore formel. Un système peut être modélisé de plusieurs manières correspondant à différents point de vue. Le rôle d'un modèle est celui de comprendre, de concevoir, de simuler et d'améliorer un système.

— ***Méta-modèle***

  Les concepts et la sémantique d'un modèle sont définis par une grammaire, ces règles auxquelles le modèle doit être conformes s'appelle 'méta-modèle'. Par conséquent, tout modèle doit être conforme à un méta-modèle.

  Quatre couches d'abstraction (Figure 2.4) permettent de délimiter différents concepts :
  Le niveau M0 correspond à la réalité. Celle-ci est représentée par un premier niveau d'abstraction qui est le modèle. Ce dernier est conçu en respectant une syntaxe qui est le méta-modèle du

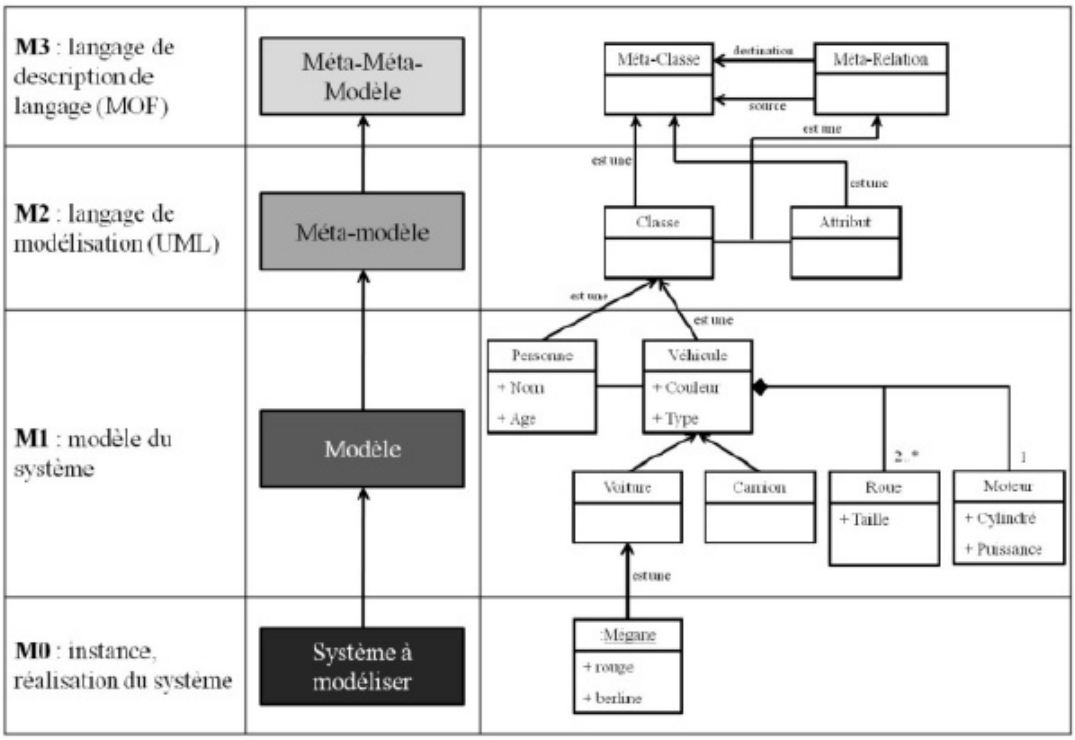

FIGURE 2.4: Les différentes couches de la méta-modélisation de l'OMG.

niveau M2. Le méta-modèle est à son tour un modèle et doit par conséquent être conforme à une syntaxe. Le méta-méta-modèle est le langage qui permet de décrire le méta-modèle et il constitue le niveau M3. Notons que dans la pratique, on considère qu'un méta-méta-modèle est conforme à lui-même.

— ***Transformation***

La transformation consiste à passer d'un modèle à un autre tous deux conformes à leurs méta-modèles respectifs. Un jeu de règles définit le cadre de la transformation en précisant comment un élément du méta-modèle source sera traduit en un élément du méta-modèle destinataire.

Plusieurs langages peuvent décrire les jeux de règles, nous en citerons l'EMF (Eclipse Modeling Framework) [Steinberg et al., 2008], Kermeta [Muller et al., 2005] ou encore le standard de l'OMG QVT (Query Views Transformation) [Object Management Group, 2011a].

Il est à noter, toutefois, que MDA n'est pas une méthodologie de conception mais plutôt un cadre générique pour mettre en application l'utilisation de modèles dans un processus de développement d'un système.

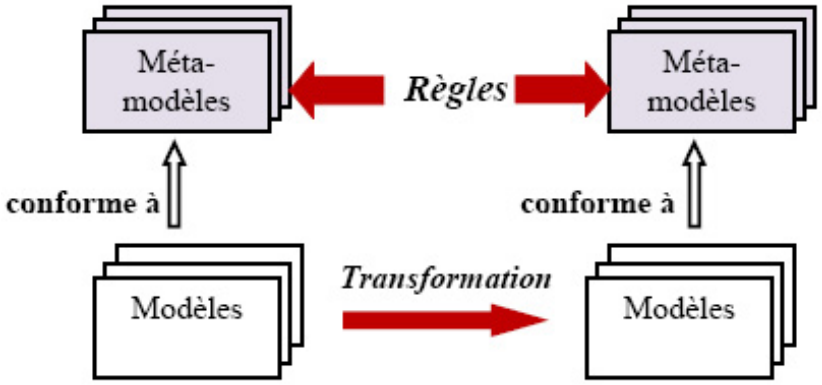

FIGURE 2.5: Transformation de modèles.

— **ACCORD|UML**

La méthodologie ACCORD|UML [Gérard et al., 2002] permet le développement de modèles d'applications temps réel distribuées et embarquées. Basée sur UML et dirigée par les modèles, son objectif est de masquer autant que possibles les détails d'implémentation [Phan et al., 2004]. ACCORD|UML prend en compte la modularité au niveau de la structure ainsi que le parallélisme au niveau des services.

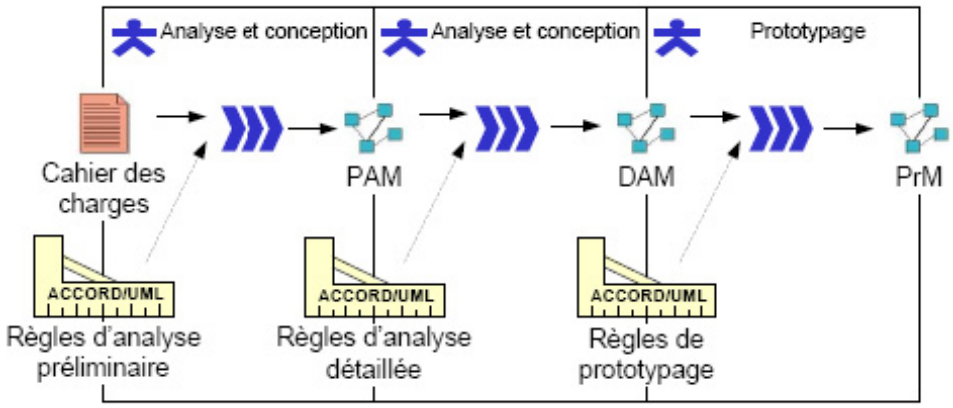

FIGURE 2.6: Vue globale du processus de développement *ACCORD/UML* [Taha, 2008].

Trois phases constituent le processus de la méthodoogie ACCORD|UML : une première phase d'analyse permet de produire le modèle de haut niveau PAM (Preliminary Analysis Model). Se basant sur ce dernier, la seconde phase d'analyse qui est plus détaillée se charge de produire le DAM (Detailed Analysis Model). Enfin, la phase de prototypage produit le PrM (Prototype Model). Celui-ci étant un modèle complet, l'atelier ACCORD peut alors procéder à une

génération automatique du code. Concernant le temps réel, la méthodologie ACCORD|UML définit deux concepts qui sont *RealTimeObject* et *RealTimeFeature*.

— **GASPARD2**

GASPARD2 (Graphical Array Specification for Parallel and Distributed Computing) [Gamatié et al., 2008] est un environnement qui permet une modélisation MDA associant une modélisation UML avec le profil MARTE dédié à la modélisation de systèmes embarqués temps réel.

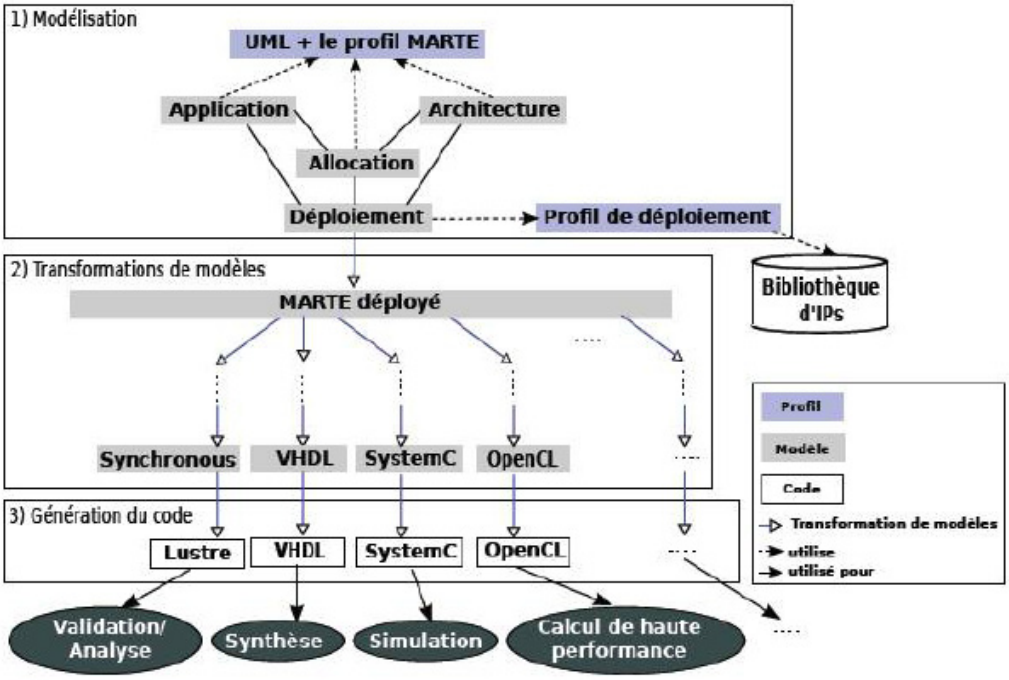

FIGURE 2.7: Flot de co-conception basé *MDA* dans l'environnement *Gaspard2* [Cherif, 2014].

L'approche est particulièrement adaptée à la modélisation d'applications massivement parallèles et distribuées supportées par des plateformes d'exécution homogènes. GASPARD2 permet la génération de code VHDL et SystemC à partir de modèles UML. L'aspect reconfigurable d'un système aux deux niveaux applicatif et matériel est également pris en compte.

— **ModES**

ModES (Model-driven dESign approach) [do Nascimento Francisco Assis et al., 2007] est une approche basée sur l'IDM. A la fois une méthodologie et un ensemble d'outils de conception, d'estimation et de génération de code, ModES permet de concevoir des systèmes embarqués. Des modèles d'application ainsi que des modèles de plateforme, spécifiés dans le langage UML,

sont en premier lieu transformés en des modèles conformes respectivement au méta-modèle d'application interne IAMM (Internal Application Meta-Model) et au méta-modèle de plate-forme interne IPMM (Internal Platform Meta-Model). Une seconde phase consiste à réaliser un mapping entre ces deux modèles conformément au méta-modèle de mapping MMM (Mapping Meta Model). La troisième phase se charge enfin de transformer les modèles en des modèles d'implantation conformes à un méta-modèle d'implantation IMM (Implementation Meta-Model). Des outils permettent d'analyser les propriétés du système telles que la consommation d'énergie, l'empreinte mémoire . . . etc.

2.3.6 Techniques formelles

En génie logiciel, on parle de technique formelle lorsque celle-ci utilise un langage formel dans la définition et la manipulation des différentes entités ainsi que dans le système de preuve qu'elle possède.

Les systèmes embarqués temps réel étant des systèmes particulièrement à contraintes, beaucoup de travaux proposent des techniques formelles afin de s'assurer que les contraintes exigées soient satisfaites. Lors du développement, le recours aux techniques formelles permet d'une part de s'assurer que les systèmes répondent aux exigences des cahiers des charges et que les contraintes exigées sont respectées et de vérifier d'autre part que le développement en soi est correct.

En effet, on distingue deux cadres d'utilisation des techniques formelles [Alkhodre, 2004] :

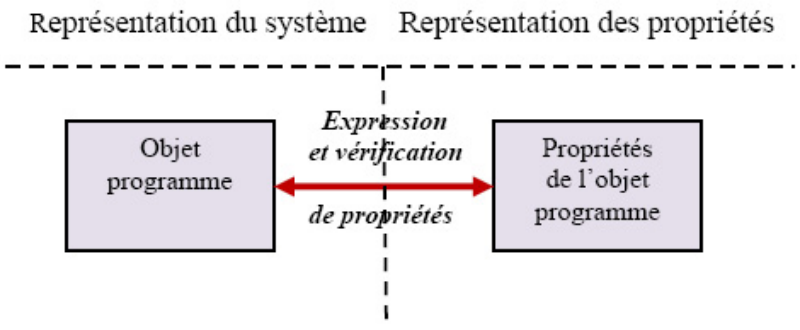

FIGURE 2.8: Techniques formelles pour la vérification.

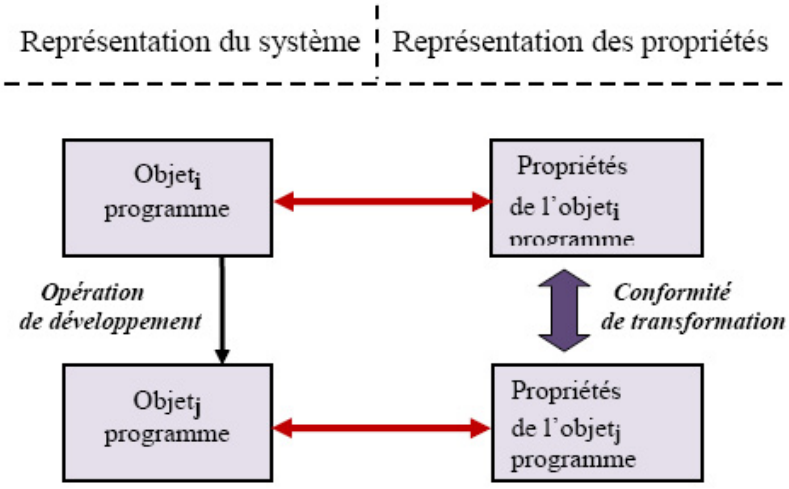

FIGURE 2.9: Techniques formelles lors d'une transformation.

— ***Vérification des propriétés d'une application*** :

Les propriétés d'une application sont exprimées tel que leur représentation formelle soit possible. Il s'agit alors de vérifier la correction du modèle (Figure 2.8).

— ***Maintien des propriétés du programme lors d'une opération de transformation*** :

Il s'agit, dans ce cas, de vérifier la conformité de la transformation d'un modèle (Figure 2.9), autrement dit, c'est le processus de développement en soi qui est vérifié.

Bien que la démarche formelle soit fastidieuse, quelques travaux y ont eu recours :

— **PROSEUS**

PROSEUS (Development method for PROtotyping embedded SystEms by using UML and SDL)

[Babau and Alkhodre, 2001] est une méthode de développement de systèmes embarqués basée sur le langage SDL [ITU-T, 1996]. Afin d'intégrer les contraintes temps réel dans le modèle du système, l'approche offre un typage des signaux échangés entre système et évènements qui modélise les contraintes de temps de l'application à développer. La définition de l'architecture matérielle et logicielle s'appuie sur des structures types définies en SDL qui représentent les unités de traitement du système, les médiums de communication et l'environnement.

— **MeMVaTEx**

MeMVaTEx [MeM, 2005] (Méthode de Modélisation pour la Validation et la Traçabilité des Exigences) est une méthodologie basée sur le profil UML MARTE [Object Management Group, 2007a], elle prend en compte le traitement des exigences en utilisant les diagrammes SysML [Object Management Group, 2001]. Les exigences sont modélisées comme des classes et demeurent telles quelles durant tout le cycle de développement.

2.4 Méthodolgie AAA (Adéquation Algorithme Architecture)

La méthodologie AAA (Adéquation Algorithme Architecture) [Grandpierre et al., 1999] est une méthode de co-design qui offre un environnement de prototypage rapide. Disposant d'une part d'une application sous forme de graphe de tâches éventuellement parallèles et d'autre part d'une plateforme d'exécution exprimée par des ressources inter-connectées les unes aux autres exprimant ainsi le parallélisme effectivement disponible et enfin d'un ensemble de contraintes temporelles, il s'agit alors d'appliquer un algorithme d'adéquation basé sur des heuristiques et qui permet de déterminer un partitionnement. Celui-ci est réalisé en projetant le modèle fonctionnel sur le modèle de la plateforme matérielle de telle sorte que les contraintes imposées soient respectées.

Les résultats obtenus sont un graphe temporel et un code d'exécution générique. Il est également possible de faire une analyse du partitionnement des tâches applicatives sur les ressources de la pateforme d'exécution. Cette méthodologie est supportée par l'outil Syndex [Sorel, 2004].

La méthodologie AAA est fondée sur des modèles de graphes pour la spécification des algorithmes applicatifs ainsi que pour celle des architectures matérielles distribuées. L'adéquation consiste alors à résoudre un problème d'optimisation consistant à choisir l'implémentation la plus optimale, autrement dit, celle qui satisfait au mieux les contraintes temps réel et d'embarquabilité imposées.

— ***Modèle d'algorithme***

Le modèle d'algorithme est un graphe de dépendances de données conditionné factorisé. Chaque

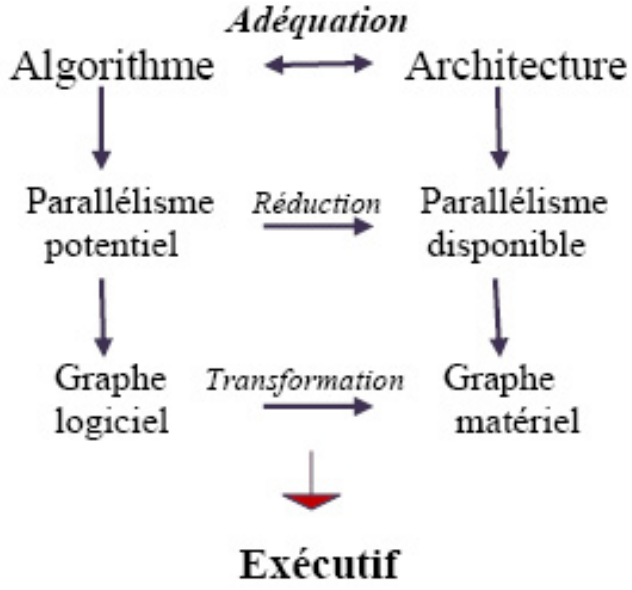

FIGURE 2.10: Illustration de la méthodologie *AAA* [Lecomte, 2011].

sommet est une opération qui consomme des données afin de réaliser une certaine tâche et produit ensuite des données résultats. Les arcs sont des dépendances de données entre les différentes opérations et expriment de ce fait une relation d'ordre partiel sur l'exécution des opérations correspondant au parallélisme potentiel de l'algorithme spécifié par l'utilisateur.

— ***Modèle d'architecture***

Le modèle d'architecture est un graphe dont chaque sommet est un processeur et chaque arc une connexion physique entre deux processeurs. L'hétérogénéité dans ce cas, signifie que les processeurs peuvent avoir chacun des caractéristiques différentes et qu'en plus certaines opérations ne peuvent être exécutées que par certains processeurs.

L'implantation d'un algorithme sur une architecture consiste en une distribution et un ordonnancement des opérations de l'algorithme sur les différents processeurs ainsi que des opérations de communication sur les liens reliant ces mêmes processeurs. La recherche d'une implantation optimisée prenant en compte les contraintes temps réel et d'embarquabilité correspond à une adéquation entre l'algorithme et l'architecture.

2.5 Langages de développement

Les langages, qu'ils soient de spécification, de conception ou d'implémentation, constituent une base pour le développement de tout système. Pour ce qui est des systèmes temps réel en général, le paramètre 'temps' et par conséquent, la notion d'échéance sont cruciaux. Des langages intégrant le temps comme paramètre sont nombreux, nous en citerons les suivants :

— **SA-RT**

Afin de remédier à l'inconvénient majeur de l'analyse structurée et qui concerne son incapacité à spécifier l'aspect dynamique d'un système, le langage SA-RT (Stuctured Analysis for Real Time) [Demarco, 1979] fut développé. En plus de la description des processus de transformation, SA-RT prend en compte la spécification de la logique de contrôle du système spécifié. La spécification s'opère d'une manière descendante à travers des niveaux de description permettant de décrire différents sous-systèmes. Cependant, SA-RT constitue un moyen informel de spécification de systèmes temps réel et la vérification de tels systèmes s'avère impossible.

— **UML-SPT**

UML-SPT (UML profile for Scheduling, Performance and Time) [Object Management Group, 2002] défini par l'OMG, est un profil qui permet d'améliorer la possibilité de modélisation de la qualité de service liée au temps (échéances et durées). En plus , il permet le déploiement du système sur une architecture cible.

— **UML-RT**

UML-RT [Selik et al., 1994] est une extension de la notation UML à base de stéréotypes, c'est une notation qui permet de représenter des objets actifs appelés 'capsules' interagissant entre eux via des ports.

D'autres travaux optent pour l'association d'UML à un langage tel que SDL [Mammeri, 2000] ou plus récemment IF [Ober and Graf, 2003] dans le but de valider les modèles conçus.

— **SDL**

Spécifier un système moyennant un langage de description formel présente l'avantage de

pouvoir vérifier les propriétés du système. SDL (Specification and Description language) est un langage de description formel utilisé notamment dans le domaine des télécommunications mais également pour les systèmes embarqués ou encore multimédia. Dans SDL, les éléments structurels de base sont les types. Cependant, les concepts d'objet et de classe ont été rajoutés. La hiérarchisation du système permet la visibilité de ce dernier. En revanche, SDL n'offre pas la possibilité de spécifier clairement l'écoulement du temps, ce qui rend délicat la vérification du respect des contraintes temporelles. Afin d'y remédier, les aspects temps et ordonnancement y ont été introduits [Bause and Buchholdz, 1990, Diefenbruch, 1997].

— **LACATRE**

Afin de faire abstraction des détails d'implémentation, les langages de conception proposent des descriptions relativement abstraites. LACATRE (Langage d'Aide à la Conception d'Aplications Temps REel) [Schwarz et al., 1991, Schwarz, 1992] est un langage graphique présentant un modèle graphique et un modèle textuel pour la conception préliminaire et détaillée d'applications temps réel.

En ce qui concerne la validation de la conception, un couplage entre LACATRE et un deuxième formalisme tel que CRSM [Belarbi, 2003, Bozga et al., 1999] ou encore IF [37] est réalisé.

2.6 Quelques critères de comparaison

Comparer des méthodes de développement de systèmes en général est une tâche fastidieuse et constitue un projet à part. Nous avons opté, de ce fait, pour un récapitulatif des approches en soulignant quelques critères à savoir l'élément central de l'approche, le type de processus méthodologique adopté ainsi que le type d'outils utilisé pour la modélisation.

Description des méthodologies			
Critères de description / **Approches de développement**	*Elément central*	*Processus*	*Outils*
GL classique	étape	complet	graphique/formel
Co-design	composant	complet	graphique/formel
Orientées UML	profile UML	complet/partiel	graphique/formel
A base de composant	composant	complet	graphique/formel
IDM	modèle	complet	graphique/formel
Techniques formelle	syntaxe formelle	complet	formel

TABLE 2.1: Tableau récapitulatif des approches de développement.

2.7 Conclusion

Nous avons, au cours de ce chapitre, abordé l'aspect développement via quelques méthodologies prenant en charge les systèmes embarqués temps réel. Ces méthodologies suivent des approches variées avec différents niveaux de formalisation.

Par ailleurs, toute méthodologie se caractérise par un enchaînement d'étapes plus ou moins pertinentes dont l'ordonnancement. Celui-ci constitue une étape clé dont le traitement doit prendre en compte plusieurs objectifs. La résolution d'un tel problème fait appel au domaine de l'optimisation multi-objectifs, c'est à cette dernière que le chapitre suivant est consacré.

Chapitre 3

Optimisation multi-objectifs

Résumé

L'ordonnancement temps réel, objet de notre travail, est un problème d'optimisation. Étant contraint par un ensemble de critères souvent conflictuels, un tel problème ne peut admettre une solution unique. C'est à la jointure de la décision multicritère et de l'optimisation combinatoire que la résolution d'un problème multicritère s'avère possible. Il s'agit de l'optimisation multi-objectifs, objet du présent chapitre.

Sommaire

3.1 Introduction **53**

3.2 Aide à la décision **53**

3.2.1 Problème de décision multicritère 54

3.2.2 Processus décisionnel 55

3.3 Optimisation combinatoire **56**

3.3.1 Problème d'optimisation 57

3.3.2 Processus d'optimisation 58

3.4 Optimisation Multi-Objectifs : La solution aux problèmes réels d'optimisation 60

3.4.1 Optimisation multi-objectifs 61

3.4.2 Notion de dominance et d'optimalité 61

3.4.3 Équilibre Intensification/Diversification 64

3.5 Méthodes de résolution 65

3.5.1 Classification 65

3.5.2 Quelques métaheuristiques 68

3.6 Ordonnancement multi-objectifs : Une application de l'OMO 73

3.7 Conclusion 75

3.1 Introduction

Dans la vie quotidienne, on se retrouve souvent confronté à des problèmes nécessitant une prise de décision tenant compte du contexte dans lequel on se situe, des informations dont on dispose et de ses préférences. Divers domaines y sont liés à savoir le choix d'investissement, la mise en place de systèmes de recommandation, la planification d'opérations, l'évaluation d'un projet . . . etc. Lorsque l'enjeu est important, une assistance concernant la prise de décision devient alors primordiale. C'est ainsi que le domaine de l'aide à la décision intervient pour proposer des modèles de décision formels dont l'objectif est d'aider un ou plusieurs décideurs dans leur prise de décision. Par ailleurs, le domaine de l'optimisation combinatoire se focalise sur la mise en œuvre d'algorithmes dont le but est de déterminer les solutions optimales relatives à des domaines de très grande taille.

Lorsque le domaine de l'aide à la décision, qui en réalité, fait intervenir plusieurs critères, est associé à celui de l'optimisation combinatoire, on parle alors d'optimisation multi-objectifs.

3.2 Aide à la décision

Selon Roy [Roy, 1990], "'l'aide à la décision est définie comme l'activité de celui qui, en prenant appui sur des modèles clairement explicités mais non nécessairement complètement formalisés, aide à obtenir des éléments de réponse aux questions que se pose un intervenant dans un processus de décision et à recommander un comportement de nature à accroître la cohérence entre l'évolution du processus et les objectifs de cet intervenant."
L'aide à la décision utilise des techniques et des méthodologies issues de divers domaines dont les mathématiques appliquées telles que la théorie de la décision, l'optimisation, les statistiques ainsi que des domaines moins formels tels que l'analyse des organisations et les sciences cognitives. Perny [Perny, 2000] définit un problème de décision par la donnée d'un triplet (S, I, P) où :

— S est un ensemble de solutions réalisables (ou d'actions potentielles) qui peut être défini en extension, par une énumération explicite de ses éléments, ou en compréhension, par une

propriété caractéristique de ses éléments ;

— I est l'information préférentielle ;

— P est la problématique.

Disposant de ces éléments, l'aide à la décision a pour but de faciliter la prise de décision d'un acteur confronté à une hypothèse de travail.

3.2.1 Problème de décision multicritère

Lorsqu'un décideur doit effectuer un choix, il est le plus souvent confronté à un ensemble de critères à prendre en compte. Prenons l'exemple de l'achat d'un appartement, les critères à prendre en considération dans ce cas sont le prix, la superficie, l'emplacement ... etc. Le problème qui en découle est que les critères sont souvent conflictuels, autrement dit, l'amélioration d'un critère se fait généralement au détriment d'au moins un autre critère.

Formellement, un critère c_i est défini comme une fonction, définie sur l'ensemble S des solutions réalisables, qui prend ses valeurs dans un ensemble totalement ordonné, et qui représente les préférences du décideur selon un point de vue [Vincke, 1989].
$c(s) = (c_1(s), \dots c_m(s))$ est désigné alors comme le vecteur de performances i.e. de valeurs $c_i(s)$ de la solution s.
La prise en compte explicite de différents critères induit les préférences partielles $\geq_i$ sur l'ensemble de solutions S :

$$\forall s, s^{'} \in S \quad s \geq_i s^{'} \Leftrightarrow c_i(s) \leq c_i\left(s^{'}\right) \tag{3.1}$$

où $s \geq_i s^{'}$ signifie que la solution s est préférée à la solution $s^{'}$ selon la relation de préférence $\geq_i$.
Or, dans un cadre multicritère, il faudrait définir une préférence globale $\geq$ de telle sorte que tous les critères soient pris en compte simultanément. La relation de préférence globale $\geq$ est considérée comme synthétisant les préférences partielles [Galand, 2008]. Un problème de décision multicritère peut alors être défini de la manière suivante :

Définition :

Étant donné une relation de préférence globale $\geq$ et un ensemble S de solutions, il s'agit alors de déterminer l'ensemble $M\left(S,\geq\right)$ des éléments maximaux de S selon la relation de préférence $\geq$, où $M\left(S,\geq\right)$ est défini par :

$$M\left(S,\geq\right)=\left\{s\in S,\forall s^{'}\in S,non\left(s^{'}\geq s\right)\right\}. \tag{3.2}$$

On en déduit que dans le domaine de l'aide à la décision multicritère, l'objectif est essentiellement la formalisation du problème, des préférences du décideur ainsi que des décisions susceptibles d'être prises.

3.2.2 Processus décisionnel

Le processus de décision est un processus structuré, il consiste en un enchainement de quatre phases :

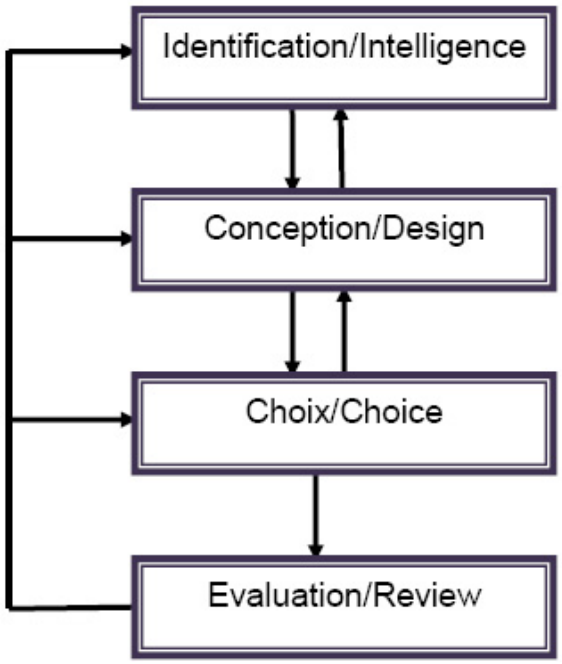

FIGURE 3.1: Le processus de décision selon le modèle IDCR (de Simon).

1. ***Identification (compréhension)*** : consiste à analyser la situation et à bien comprendre le problème :
   - Définir le problème (situation décisionnelle) en se posant des questions ;

   — Structurer les objectifs ou buts ;

   — Rechercher les actions possibles sur le système.

2. ***Modélisation*** : consiste en une formulation du problème et la description des solutions potentielles. Une première étape consiste en l'identification des alternatives ou actions potentielles. C'est une étape délicate car les alternatives ne sont pas toujours fournies directement.

   — Analyser le problème, le champ de la décision et les différentes actions possibles ;

   — Choisir le ou les modèles de décision en fonction de la complexité du problème à traiter ;

   — Déterminer les variables de décision, les critères d'évaluation ainsi que les relations entre ces variables et construire les différentes alternatives ;

   — Décrire/prévoir l'état du système si on lui applique une action.

3. ***Choix*** : une solution est choisie en fonction de critères appréhendés par le décideur. Le choix peut être réalisé grâce à des outils et techniques d'aide à la décision. L'action ou les actions retenue(s) est celle ou sont celles qui donne(nt) au système l'état ou les états les plus désirables par rapport aux objectifs.

4. ***Recommandation*** : L'évaluation des alternatives se fait à l'aide d'un ensemble d'attributs permettant l'estimation du degré d'atteinte des objectifs par chacune des alternatives.
   Cette phase est importante, en particulier dans le cas où la décision s'intègrerait dans un processus dynamique où de nouvelles informations pertinentes peuvent influencer tel ou tel choix, voir le modifier complètement.

3.3 Optimisation combinatoire

Alors que l'aide à la décision est essentiellement un problème de modélisation du problème à travers la définition de décisions possibles, des critères d'évaluation et enfin des préférences du décideur, l'optimisation combinatoire se charge, quant à elle, de mettre en œuvre des algorithmes dont le but est de déterminer les solutions optimales relatives à des domaines de très grande taille.

3.3.1 Problème d'optimisation

Un problème d'optimisation consiste en la recherche d'un optimum (minimum ou maximum) d'une fonction donnée.

Formellement, un problème d'optimisation est défini comme suit [Collette and Siarry, 2002] :

$$\begin{cases} minimiser & f(\vec{x}) & (fonction\ objectif\ à\ optimiser) \\ avec & \vec{g}(\vec{x}) \leq 0 & (m\ contraintes\ d'inégalité) \\ et & \vec{h}(\vec{x}) = 0 & (p\ contraintes\ d'égalité) \end{cases} \tag{3.3}$$

$\vec{x} \in \Re^n$: vecteur regroupant les variables de décision qui sont les paramètres d'optimisation ;

$\vec{g}(\vec{x}) \in \Re^m$: représentent m contraintes d'inégalité ;

$\vec{h}(\vec{x}) \in \Re^p$: représentent p contraintes d'égalité.

L'ensemble des contraintes restreignent l'espace de recherche de la solution optimale.

Les éléments définissant un problème d'optimisation sont donc :

1. Une fonction-objectif f ;
2. Un ensemble de solutions S ;
3. Un ensemble de contraintes C que doit satisfaire une solution afin d'être admissible.

Une solution réalisable x peut être décrite par un vecteur de décision $(x_1, x_2, \ldots x_n)$ auquel est associée une valeur objectif $f(x)$. La solution optimale est celle dont la valeur de l'objectif est la plus petite (grande) dans un contexte de minimisation (maximisation) parmi l'ensemble des solutions. Par conséquent, l'objectif de l'optimisation est de déterminer les extremums d'une fonction.

Quelques définitions

— *Fonction-objectif* : La fonction qu'on désire optimiser ;

— *Variables de décision* : Elles sont regroupées dans le vecteur $\vec{x}$ qui correspond à l'ensemble des variables du problème ;

— *Minimum global* : Un point $\vec{x}^*$ est appelé minimum global de la fonction f si : $\forall \vec{x}, \vec{x} \neq \vec{x}^* \Rightarrow f(\vec{x}^*) < f(\vec{x})$;

— *Voisinage* : Un voisinage *V* est une fonction $V : D \rightarrow P(D)$, qui associe à chaque $x \in D$ un sous ensemble $N(x)$ de *D* des voisins de x. L'application *N* est appelée 'structure de voisinage' ;

— *Minimum local fort* : $\vec{x}^*$ est un minimum local fort de la fonction f si :
$\forall \vec{x} \in V(\vec{x}^*), \vec{x} \neq \vec{x}^* \Rightarrow f(\vec{x}^*) < f(\vec{x})$ où $V(\vec{x}^*)$ représente le voisinage de $\vec{x}^*$;

— *Minimum local faible* : $\vec{x}^*$ est un minimum local faible de la fonction f si :
$\forall \vec{x} \in V(\vec{x}^*), \vec{x} \neq \vec{x}^* \Rightarrow f(\vec{x}^*) \leq f(\vec{x})$.

3.3.2 Processus d'optimisation

Tandis que l'aide à la décision se concentre particulièrement sur la modélisation du problème, l'optimisation consiste en plus de la modélisation du problème, en sa résolution. Plus particulièrement, quatre étapes constituent le processus d'optimisation à savoir :

1. ***Modélisation du problème*** : il s'agit de définir l'espace de recherche ainsi que les solutions ;
2. ***Formulation mathématique*** : consiste à définir une fonction-objectif ainsi que les contraintes ;
3. ***Application d'une méthode d'optimisation*** : la méthode appropriée au problème est effectivement appliquée ;
4. ***Obtention d'une solution*** : consiste à produire le résultat.

Bien qu'on qualifie généralement de 'combinatoires' les problèmes dont la résolution se ramène à l'examen d'un nombre fini de combinaisons, cette résolution se heurte très souvent à une explosion du nombre de combinaisons à explorer. En effet, la principale difficulté dans l'optimisation combinatoire

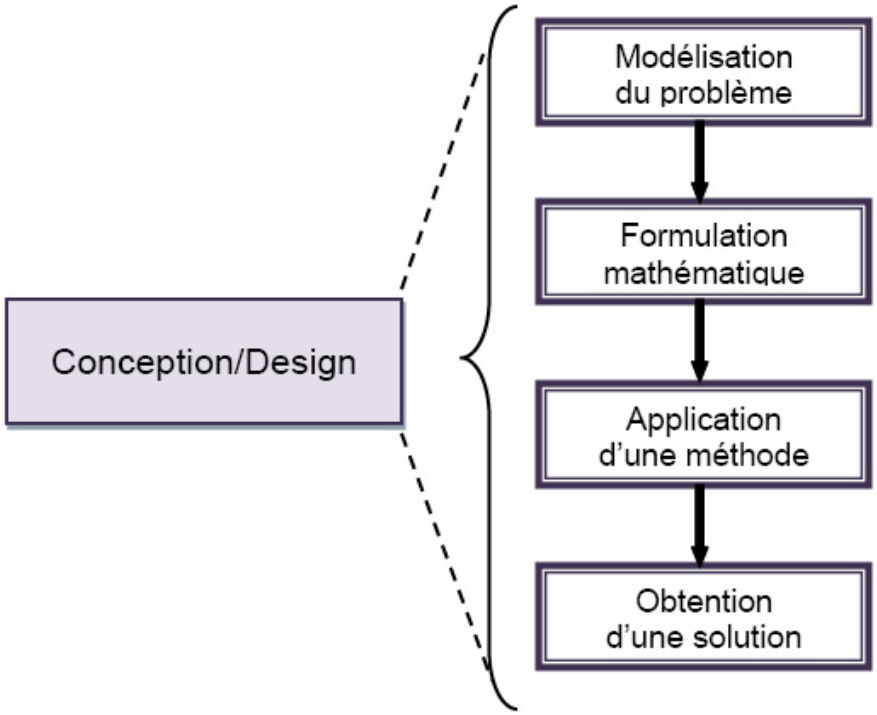

FIGURE 3.2: Le processus d'optimisation.

concerne l'explosion combinatoire, autrement dit, l'augmentation de la taille des données engendre l'augmentation exponentielle de la taille de l'espace de recherche. Deux grandes catégories de méthodes sont utilisées en optimisation combinatoire :

— ***Méthodes exactes*** : Ce sont des méthodes qui permettent de trouver la solution optimale mais qui, cependant, sont contraintes par la taille des problèmes à résoudre. En effet, Le temps de recherche de ces méthodes augmente d'une façon exponentielle avec la taille du problème.

— ***Méthodes approchées*** : Ce sont des méthodes qui effectuent des recherches guidées à l'intérieur de l'espace de recherche afin de caractériser rapidement une solution de bonne qualité. De telles méthodes permettent de trouver une solution de bonne qualité mais non optimale. Contrairement aux méthodes exactes, les méthodes approchées ne sont pas contraintes par la taille des problèmes.

L'aide à la décision et l'optimisation combinatoire sont reliées via un cycle alternant recherche de solutions et décisions prises autour de ces mêmes solutions.

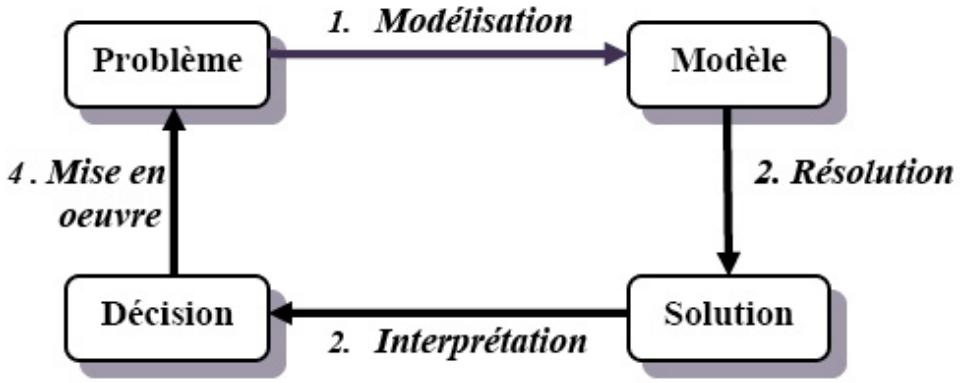

FIGURE 3.3: Cycle Décision/Optimisation.

La modélisation du problème permet de définir un modèle sur lequel un raisonnement est réalisé et ce afin de définir la solution. L'interprétation de la solution permet de faire un choix qui peut être l'acceptation ou bien le refus de la solution. Dans ce dernier cas, une reformulation du problème s'avère nécessaire et une phase de modélisation est alors entamée déclenchant ainsi le cycle suivant.

3.4 Optimisation Multi-Objectifs : Point de rencontre de l'aide à la décision et de l'optimisation combinatoire

A la jointure de ces problématiques décisionnelles et algorithmiques, le domaine de l'optimisation combinatoire multi-objectifs prend à la fois les deux aspects à savoir la prise en compte de plusieurs critères ainsi que le risque d'un nombre combinatoire de solutions potentielles. Plus particulièrement, l'optimisation multi-objectifs se focalise sur la phase de recherche des solutions de compromis.

L'optimisation multi-objectifs possède ses racines dans les travaux en économie de Edgeworth [Edgeworth, 1881] et Pareto [Pareto, 1896]. Initialement utilisée en économie et dans les sciences du management, elle a été graduellement introduite dans les sciences de l'ingénieur.

En réalité, la plupart des problèmes réels d'optimisation sont décrits à l'aide de plusieurs critères souvent conflictuels devant être optimisés simultanément. Il s'agit, par conséquent, de définir des fonctions objectifs permettant l'évaluation des critères et de déterminer les meilleurs compromis

possibles permettant de résoudre le problème. Cet ensemble de solutions optimales offertes aux décideurs est appelé 'front de Pareto'.

3.4.1 Optimisation multi-objectifs

L'optimisation multi-objectifs consiste à optimiser un vecteur de fonctions composé d'un ensemble de fonctions objectifs. Dans ce cas, il n'existe pas de solution unique au problème mais un ensemble de solutions de compromis appelé 'solutions de Pareto'.

En effet, toute solution de cet ensemble est optimale dans le sens où aucune amélioration ne peut être faite sur une composante du vecteur sans dégradation d'au moins une autre composante de ce vecteur [Srinivas and Deb, 1995]. Le choix d'une solution par rapport à une autre nécessite la connaissance du problème ainsi que les besoins courants. En effet, une solution qui convient à une situation donnée peut ne plus l'être si l'on fait varier certaines données.

Formellement, un problème d'optimisation multi-objectifs est formulé comme suit [Collette and Siarry, 2002] :

$$\begin{cases} optimiser & F(x) = (f_1(x), f_2(x), \dots f_n(x)) \\ sous & x \in D \end{cases} \tag{3.4}$$

où n est le nombre de critères $(n \geq 2)$ et $x = (x_1, x_2, \dots x_k)$ est le vecteur représentant les variables de décision, D représente l'ensemble des solutions réalisables et chacune des fonctions $f_i(x)$ est à optimiser.

Des fonctions-objectifs sont respectivement associées aux critères considérés. A chaque solution dans l'espace des solutions est associée une valeur dans l'espace des fonctions-objectifs. Ces dernières permettent l'évaluation des critères correspondants.

3.4.2 Notion de dominance et d'optimalité

Une solution réalisable $x^* \in D$ est *Pareto optimale* (efficace, non dominée) si et seulement s'il n'existe pas de solution $x \in D$ telle que x domine x^*. On dit qu'une solution $y = (y_1, y_2, \dots y_k)$

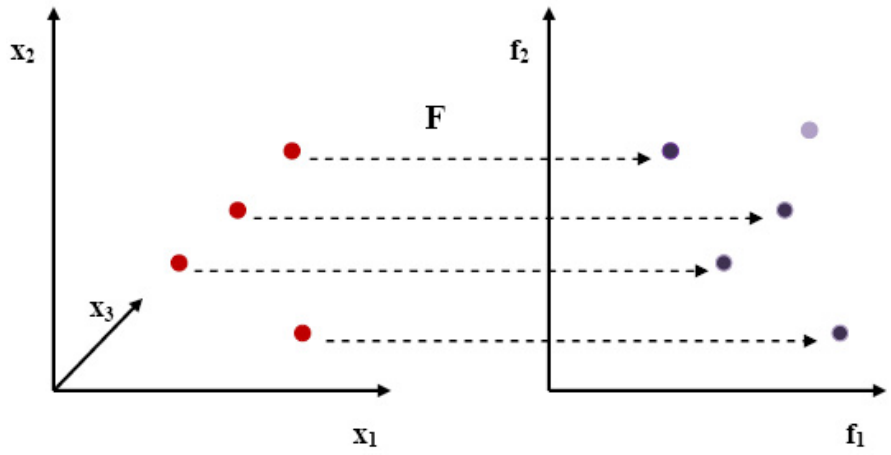

FIGURE 3.4: Espaces des solutions réalisables et des fonctions-objectifs.

domine une solution $z = (z_1, z_2, \ldots z_k)$ dans le cas d'une minimisation d'objectifs, ssi :

$$\forall i \in [1 \ldots n], f_i(y) \leq f_i(z) \;\; et \; \exists i \in [1 \ldots n] \, telque f_i(y) < f_i(z) \tag{3.5}$$

Si x^* est une solution optimale de Pareto (appelée également solution efficace), alors $f(x^*)$ est appelé 'Point non dominé'. Un point non dominé étant l'image d'une ou de plusieurs solutions efficaces, celles-ci ne sont donc pas comparables entre elles et représentent de bons compromis potentiels.

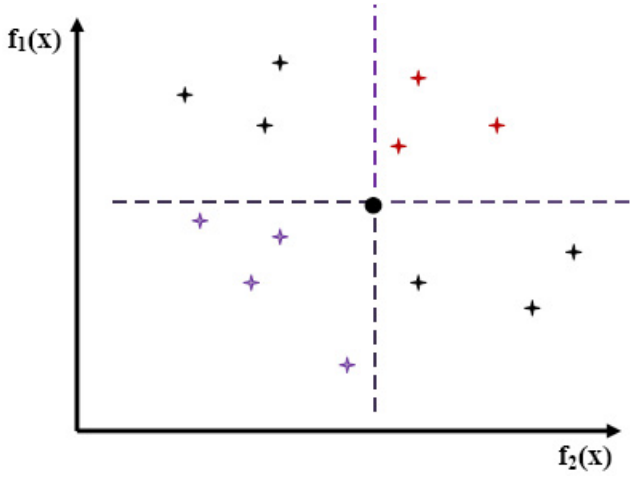

FIGURE 3.5: Exemple de dominance (cas de deux objectifs à minimiser).

Selon l'exemple (Figure 3.5), le point noir domine chacune des croix rouges, est dominé par chacune des croix violettes et est équivalent aux croix noires au sens de la dominance. Notons par ailleurs que la relation de non dominance est une relation transitive.

Points particuliers

Des points particuliers, pouvant représenter des solutions réalisables ou pas, sont définis dans l'espace objectif, ils permettent d'évaluer les solutions trouvées.

— *Point idéal* : Le point idéal z_I est le point qui a comme valeur pour chaque objectif la valeur optimale de l'objectif considéré.

$$z^I \text{ tel que } \forall i \in [1...n], f_i\left(z^I\right) = opt_{x \in D} f_i\left(x\right)$$

Ce point ne correspond pas à une solution réalisable vu que les objectifs sont conflictuels et une telle solution ne peut avoir lieu.

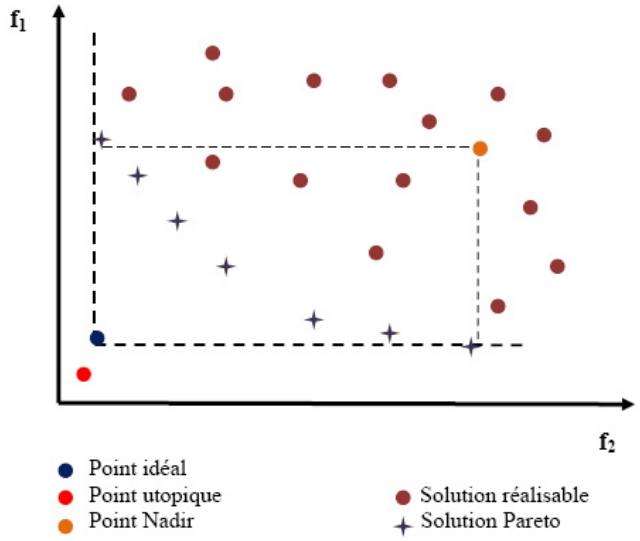

FIGURE 3.6: Types de solutions dans l'espace des solutions.

— *Point utopique* : Le point utopique est défini comme suit :

$$z^u = z^I - \epsilon U$$

où $\epsilon > 0$ et U est le vecteur unitaire $(U = (1, ..., 1) \in \Re^n)$.

Un tel point, comme son nom l'indique, est irréalisable.

— *Point Nadir* : Le point Nadir est défini en bi-objectifs par :

$$z^N \text{ tel que } \forall i \in [1...2], f_i\left(z^N\right) = opt_{x \in D / f_j(x) = f_j(z^I)} f_i(x) \text{ avec } j \neq i$$

.

Cela revient donc à affecter pour chaque objectif du point Nadir la meilleure valeur possible parmi les solutions optimisant l'autre objectif. Par ailleurs, une notion joue un rôle important dans la localisation d'un point, c'est le concept du Voisinage (cf. sous-section 3.3.1).

3.4.3 Équilibre Intensification/Diversification

Dans un problème d'optimisation multi-objectifs, deux paramètres sont pris en compte à savoir la diversification et l'intensification. Au cours de la résolution, équilibrer entre ces deux concepts est essentiel :

— ***Diversification***

Le but de la diversification ou l'exploration est de permettre à la recherche locale d'échapper à des optima locaux de l'espace de recherche et de trouver des solutions diversifiées le long de la frontière Pareto. L'un des mécanismes qui peut servir à la diversification est la détection de nogoods [Habet, 2004]. Un *nogood* est, par définition, une configuration qui ne peut être étendue à une solution, ce qui fait des éléments d'une telle configuration non opportuns à la recherche.

— ***Intensification***

L'intensification ou l'exploitation a pour but de focaliser la recherche autour des zones de l'espace des configurations jugées pertinentes. L'objectif est de converger vers la frontière Pareto. Ceci est accompli par une relance de l'exploration tabou à partir des éléments de la ou des meilleures configurations trouvée S^*.

La figure (Figure 3.7) montre trois cas de qualité à savoir le cas où la convergence l'emporte sur la diversité (a), ensuite le cas inverse (b) et enfin le cas d'une disposition idéale des solutions (c) faisant ainsi un compromis entre convergence et diversité.

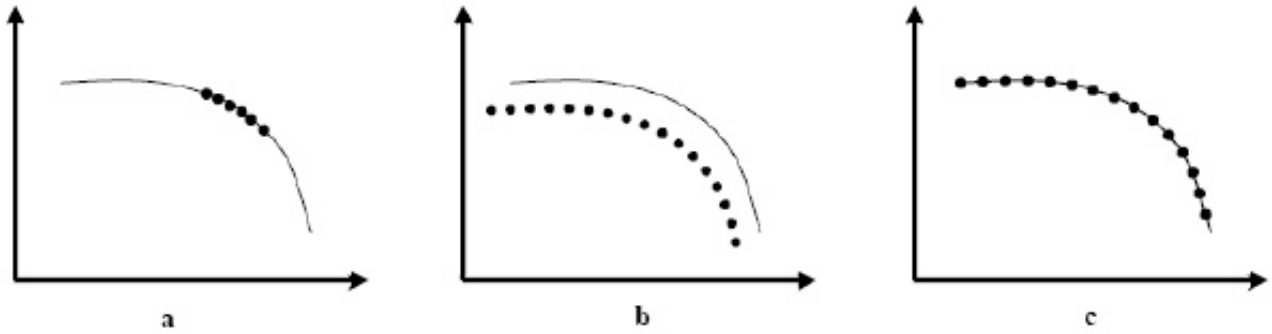

FIGURE 3.7: Convergence et Diversité.

3.5 Méthodes de résolution

3.5.1 Classification

La classification des méthodes multi-objectifs peut se faire relativement à plusieurs points de vue. On distingue trois classifications [Talbi, 2002].

— ***Du point de vue décideur***

Cette première classification adopte un point de vue décideur. Selon la coopération de ce dernier avec la méthode d'optimisation, on distingue trois classes :

— *Méthodes a priori*

Dans ce cas, le décideur est supposé avoir choisi le compromis à faire et par conséquent avoir fixé le poids de chaque objectif.

— *Méthodes interactives*

Une coopération progressive s'opère entre le décideur et la méthode d'optimisation. Les connaissances dont le décideur dispose tout le long du processus lui permettent d'affiner son choix jusqu'à aboutir à celui qui correspond à ses besoins.

— *Méthodes a posteriori*

Dans ce cas, le choix du décideur se fait après que les solutions soient produites par la méthode d'optimisation. Une telle stratégie est appropriée lorsque le nombre de solutions est admissible.

— ***Selon l'algorithme de résolution***

Selon l'algorithme adopté, on distingue essentiellement deux classes de méthodes de résolution d'un problème d'optimisation à savoir les méthodes exactes et les méthodes approchées.

— *Méthodes exactes*

Ces méthodes conviennent particulièrement à la résolution de problèmes de petite taille. En effet, le nombre de combinaisons possible, dans ce cas, est relativement faible pour que l'exploration de l'espace de solutions se fasse en un temps raisonnable. La procédure de séparation et d'évaluation *Branch and Bound* (*B&B*) [Sen et al., 1988, Le Pape, 1995], l'algorithme A^* [Stewart and White, 1991] ainsi que la *programmation dynamique* [White, 1982, Bellman, 1986] appartiennent à la classe des méthodes exactes.

— *Méthodes approchées*

Pour des problèmes à plusieurs critères ou de grande taille, les méthodes exactes s'avèrent inefficaces vu les difficultés liées à la fois, à la complexité NP-complet et le cadre multi-critère du problème. Dans ce cas, des méthodes dites 'approchées' sont utilisées, elles ne garantissent pas de trouver d'une manière exacte l'ensemble des solutions optimales, en revanche, elles permettent de s'en approcher.

Les méthodes approchées peuvent être classées en deux sous classes [Talbi, 2002] :

1. *Heuristiques* :

   Les heuristiques sont des méthodes empiriques basées sur des règles simplifiées pour optimiser un ou plusieurs critères. Elles sont spécifiques à un problème dont les connaissances sont prises du domaine auquel appartient le problème en question [Gandibleux et al., 1994].

2. *Métaheuristiques* :

   Les métaheuristiques sont des algorithmes généraux applicables à une grande variété de problèmes d'optimisation multi-objectifs. Une métaheuristique vise en plus, à réaliser un compromis entre intensification et diversification en concentrant la recherche dans les zones prometteuses et en explorant de nouvelles zones de l'espace

de recherche.

— ***Selon l'approche de résolution***

Une troisième classification s'appuie sur le type d'approche de résolution utilisé, on en distingue trois sous-classes [Talbi, 2002] :

— *Approches basées sur la transformation du problème en un problème mono-objectif*

Les approches de cette classe transforment le problème initial en un ou plusieurs problèmes mono-objectif. Cette transformation se fait généralement par une *agrégation* combinant les différentes fonctions coût f_i du problème en une seule fonction objectif *F*. Ceci exige du décideur une bonne connaissance du problème.

La méthode ϵ-*contrainte* [Hertz et al., 1994] est une autre approche qui appartient également à cette catégorie. Elle consiste à retenir une fonction f_k comme objectif unique à optimiser, tandis que les objectifs restants sont transformés en contraintes. Différentes valeurs de ϵ peuvent être données afin de générer plusieurs solutions Pareto optimales.

Enfin, une troisième approche est la *Programmation par but* [Sandgren, 1994] où le décideur doit définir des buts qu'il désire atteindre pour chaque objectif. Ces valeurs sont introduites dans la formulation du problème, le transformant en un problème mono-objectif.

— *Approches non-Pareto*

Contrairement aux précédentes et au lieu de combiner les fonctions coût, les approches non-Pareto utilisent des opérateurs de recherche qui traitent séparément les différents objectifs. L'exemple le plus classique est *VEGA* [Schaffer, 1985]. Ces méthodes ont souvent du mal à trouver les solutions de compromis puisqu'elles focalisent sur les portions extrêmes du front [Dhaenens-Flipot, 2005].

— *Approches Pareto*

Les approches Pareto utilisent la notion d'optimalité de Pareto avec un processus de sélection basé sur la notion de non-dominance. L'un des premiers à discuter de l'intérêt de l'utilisation d'une telle notion pour la recherche de solutions a été Goldberg [Glover, 1989].

Notons que par la suite, les algorithmes génétiques, utilisant le concept de non-dominance, se sont largement imposés dans la résolution de problèmes d'optimisation multi-objectifs.

3.5.2 Quelques métaheuristiques

En raison de l'approximation des solutions optimales en des temps de calcul raisonnables, les méthodes approchées et notamment les métaheuristiques sont de plus en plus utilisées dans les problèmes d'optimisation multi-objectifs. Parmi ces méthodes, on distingue le Recuit simulé, la Recherche Tabou ou encore les algorithmes génétiques.

— ***Recuit simulé***

Le Recuit simulé [Kirkpatrick et al., 1983] trouve ses origines dans le phénomène thermodynamique de recuit des métaux. Se basant sur l'évolution d'un système thermodynamique, l'algorithme du Recuit simulé accepte, pour sortir d'un minimum local, les déplacements dans le voisinage d'une solution en dépit du fait qu'ils dégradent le coût de la solution et cela suivant une probabilité calculée à partir d'une température T du système à un moment donné.

En partant d'une configuration donnée, le système subit une modification élémentaire : si cette transformation a pour effet de diminuer la fonction objectif f (ou énergie) du système, elle est acceptée. Si, par contre, elle induit une augmentation de la fonction objectif, elle est acceptée alors avec la probabilité $\exp^{-f/k_B T}$.

Algorithme : Recuit Simulé

Début

Déterminer aléatoirement une solution initiale s_0 et une température initiale T_0 ;

Poser $s_{min} = s_0$ et $T_i = T_0$;

Calculer $f(s_0)$;

Répéter pour chaque itération $i/i = 1, \cdots, n$

Choisir $s^{'} \in N(s_i)$;

Calculer $\Delta f = f(s^{'}) - f(s_i)$;

Si $\Delta f < 0$ **alors** $s_{min} = s_i$;

Sinon

Trier p dans [0, 1] suivant une distribution uniforme ;

Si $p \leq e^{(-\Delta f/T)}$ **alors** $s_{min} = s_i$;

Sinon s_i est rejeté ;

jusqu'à ce que T_i soit proche de 0 ;

Calculer $T = \gamma T$;

Retourner s_{min} ;

Fin

L'algorithme du Recuit simulé a été utilisé pour le problème multicritère du voyageur de commerce [Serafini, 1992], pour la conception de réseaux de transport [Friesz et al., 1993] ainsi que pour le problème multi-objectifs du sac à dos [Ulungu et al., 1998].

— ***Recherche Tabou***

La recherche Tabou (Tabu Search) [Glover, 1989] est une métaheuristique utilisée dans la résolution de beaucoup de problèmes NP-complets tels que les problèmes de routage de véhicules, de coloration de graphes et également des problèmes d'ordonnancement.

Le fonctionnement de la recherche Tabou est similaire à celui du Recuit simulé dans le sens où une seule configuration courante évolue via les différentes itérations. Le passage d'une

configuration à une autre est réalisé en deux étapes :

Algorithme : Recherche tabou

Début

Poser $x =$ solution initiale aléatoire

Poser $x_{min} = x, f_{min} = f(x), TL = \phi$ (TL = liste tabou) ;

Répéter

Générer un sous-ensemble N de voisinage tel que :

$s_i(x) \subset S(x)$ et $(x, s_i(x)) \notin TL$;

Trouver $\underset{1<=i<=N}{f}(s(x)) - min[f(s_i(x))]$;

Ajouter $\{x, s_i(x)\}$ dans TL ;

Remplacer x par $s(x)$ tel que $s(x)$ soit la meilleure solution de $S_i(x)$;

Si $f(x) \prec f_{min}$ **alors**

$f_{min} = f(x)$

$x_{min} = x$

Fin

Jusqu'à conditions d'arrêt satisfaites ;

Fin

— La construction d'un ensemble de solutions voisines de la solution courante ;

— L'évaluation de la fonction objectif du problème en chacune des configurations appartenant à l'ensemble des solutions voisines afin de choisir la meilleure solution, même si ce choix entraîne une augmentation de la fonction objectif à minimiser.

L'idée principale est de conserver une trace des solutions déjà visitées. Pour cela, une liste Tabou est construite graduellement pour contenir les solutions déjà visitées pour que lors de la recherche locale, le voisinage de la solution courante soit réduit aux solutions ne faisant pas partie de la liste Tabou.

Cependant, un critère d'aspiration [Glover, 1995] peut être utilisé, il permet de lever l'interdiction sur une solution déjà visitée et de l'exploiter différemment.

— ***Algorithmes génétiques***

Un algorithme génétique (*AG*) est une métaheuristique qui manipule une population de solutions potentielles à la fois. Le principe de fonctionnement d'un *AG* est inspiré des principes biologiques de la sélection naturelle et de la survie des individus les mieux adaptés à l'environnement [Darwin, 1876]. S'inspirant d'un tel principe, Holland [Holland, 1975] a posé les bases de la technique d'optimisation appelée "'Algorithmes génétiques"'. En revanche, la forme actuelle des *AGs* a été développée par Goldberg [Goldberg, 1989]. Le but d'un algorithme génétique est d'optimiser une fonction prédéfinie appelée *Fonction objectif* ou *Fitness*.

Dans le vocabulaire des AGs, *l'environnement* se rapporte à l'espace de recherche qui définit l'ensemble des configurations possibles des paramètres de la fonction à optimiser. Un *individu* (solution sous forme d'un chromosome) dans cet environnement est donc une configuration possible des paramètres.

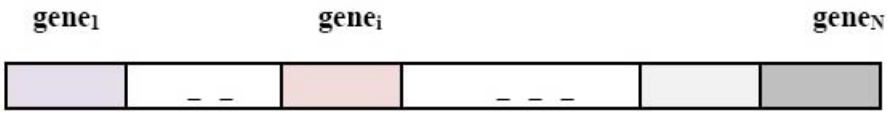

FIGURE 3.8: Structure d'un chromosome.

Un chromosome (Figure 3.8) est constitué d'une chaîne de gènes contenant les caractéristiques génétiques de l'individu. L'adaptation d'un individu à l'environnement est matérialisée par la mesure de la performance de cet individu à travers la fonction à optimiser (Fonction d'adaptation ou Fitness). Lors du passage d'une génération de solutions à une autre, deux mécanismes sont utilisés :

— *Croisement* : consiste à combiner deux moitiés du patrimoine génétique de chacun des parents pour constituer le patrimoine génétique de l'enfant ;

— *Mutation* : consiste à modifier un ou plusieurs gènes de l'enfant.

Si le nouvel individu hérite des bonnes caractéristiques de ses parents, son espérance de vie ainsi que sa probabilité de se reproduire sont alors plus élevées.

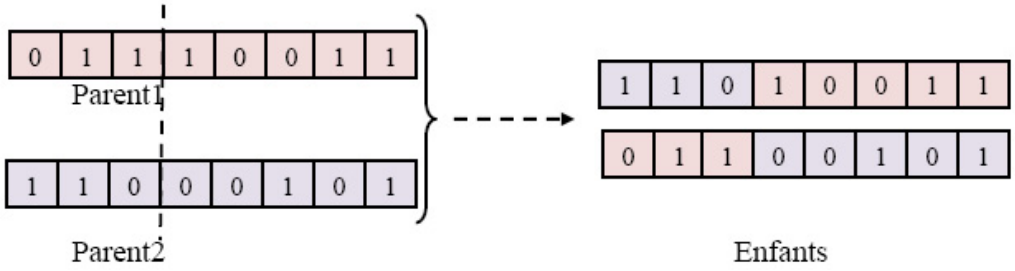

FIGURE 3.9: Exemple de croisement.

FIGURE 3.10: Exemple de mutation.

L'utilisation d'un AG nécessite la définition, au préalable, d'un espace de recherche dont les éléments de base sont les chromosomes et d'une fonction (fonction Fitness) définie sur cet espace.

Algorithme : Algorithme Génétique

Début

Génération d'une population aléatoire initiale de K individus ;

Répéter

Evaluation de la fonction objectif f pour chaque individu ;

Sélection des meilleurs individus ;

Croisement des individus sélectionnés ;

Mutation de certains individus de la population ;

Jusqu'à conditions d'arrêt satisfaites ;

Retour de la meilleure solution ;

Fin

Une population initiale d'individus (solutions possibles au problème) est générée aléatoirement. A chaque génération (itération), les meilleurs individus sont sélectionnés pour former les parents reproducteurs. Des tirages au sort sont effectués pour sélectionner aléatoirement les parents deux à deux, auxquels on applique l'opérateur de croisement pour créer des enfants.

De temps en temps, les gènes de certains enfants subissent une mutation permettant ainsi une diversification de la population. Les enfants générés par le croisement des parents seront utilisés pour former la nouvelle population et le processus reprend jusqu'au critère d'arrêt. Ce dernier peut être un nombre d'itérations fixé au départ, la faible variation de la fonction objectif pendant plusieurs générations ou bien d'autres conditions. Un autre concept tout aussi important est également utilisé par les *AGs*, il s'agit de l'*élitisme* [Goldberg, 1989]. Cela consiste à garder un échantillon des gènes des meilleurs individus afin de se prémunir contre toute détérioration que pourrait engendrer le croisement ou la mutation.

Parmi les algorithmes génétiques les plus utilisés, il y'a VEGA (Vector Evaluated Genetic Algorithm) [Schaffer, 1985], MOGA (Multi-Objective Genetic Algorithm) [Fonseca and Flemming, 1993], NPGA (Niched-Pareto Genetic Algorithm) [Horn et al., 1994], NSGA-II (Non Sorting Dominated Genetic Algorithm II [Srinivas and Deb, 1994] et SPEA2 (Strength Pareto Evolutionary Algorithm) [Zitzler et al., 2001a].

Par ailleurs, plusieurs approches hybrides ont été proposées dans la littérature, elles visent à tirer profit de chacune des approches existantes. Les algorithmes MO-GLS (Muli Objective Genetic Local Search) [Jaszkiewicz, 2002a, Jaszkiewicz, 2002b] et GTS^{MOKP} [Barichard and Hao, 2003] en sont des exemples.

3.6 Ordonnancement multi-objectifs : Un champ d'application de l'optimisation multi-objectifs

Dans la littérature, sont données diverses définitions d'un problème d'ordonnancement, nous en citerons celle proposée par Esquirol et Lopez [Esquirol and Lopez, 1999] :
" Le problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchainement, ... etc) et de contraintes portant sur l'utilisation et la disponibilité de ressources requises ".

Cependant, en plus de veiller au respect des contraintes de temps et de ressource, la fonction ordonnancement vise à satisfaire certains critères. Lorsque ces derniers sont qualifiables et exprimables en fonction des variables d'ordonnancement, ils sont injectés dans le problème d'ordonnancement, soit par ajout de nouvelles contraintes, soit par ajout d'un ou plusieurs critères d'optimisation.

On parle d'un problème d'ordonnancement monocritère lorsque il y'a un seul critère à satisfaire. Une fonction objectif est alors définie et la solution optimale est, dans ce cas, unique. Or, en réalité ce sont deux ou plusieurs critères qui doivent être pris en compte afin de traiter un problème d'une manière efficace. En effet, à partir des années 80 et partant du constat que la notion de qualité était en réalité une notion multidimensionnelle, le problème d'ordonnancement fut désormais traité comme un problème multicritère. Aussi, beaucoup de travaux ont-ils été entrepris et des approches tentant d'optimiser deux ou plusieurs fonctions objectifs ont été proposées [Nagar et al., 1995, Stein and Wein, 1997, Hoogeveen, 2005, Oyetunji and Oluleye, 2010].

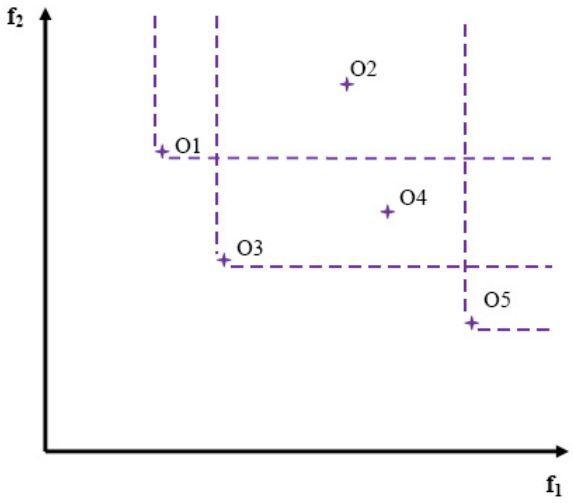

FIGURE 3.11: Espace des objectifs à deux dimensions.

Le point de départ consiste en une application constituée d'un ensemble de tâches, un ensemble de processeurs sur lesquelles ces tâches doivent être ordonnancées et enfin un ensemble de contraintes, il s'agit de trouver pour chaque tâche du système un intervalle de temps d'exécution sur l'un des processeurs tel que les contraintes imposées soient satisfaites et que les objectifs considérés soient optimisés. La résolution aboutit alors à un ensemble de solutions (ordonnancements) dites de compromis.

Lorsque dans un problème d'ordonnancement, on est en présence de deux critères à prendre en compte et qu'à chaque critère est associée une fonction objectif, on parle d'ordonnancement bi-objectifs. Dans ce cas, il peut y avoir plusieurs solutions incomparables et que l'on considère par conséquent comme optimales. Sur la figure (Figure 3.11), f_1 et f_2 sont les fonctions objectifs relatives aux deux critères considérés. Les points O_i expriment des évaluations d'ordonnancements par le biais des deux fonctions objectifs. La dominance de Pareto permet la formalisation du concept d'optimalité dans le cas multi-objectifs.

3.7 Conclusion

Le domaine de l'optimisation multi-objectifs a été abordé via un rappel des principes de base ensuite en présentant les approches de résolution. A défaut de trouver des solutions exactes, les méthodes approchées misent sur le temps d'exécution afin de produire des solutions dites approchées.
En effet, l'objectif d'une méthode approchée est de s'approcher le plus possible de l'ensemble des solutions optimales de Pareto, ce qui signifie, dans l'espace des objectifs, une progression vers le front de Pareto. Dans le cas de l'ordonnancement bi-objectifs, les solutions recherchées désignent les ordonnancements réalisant les meilleurs compromis possibles entre les deux objectifs. Tel est le but des différentes approches ayant traité cette problématique et qui sont présentées dans le chapitre suivant.

Chapitre 4

Ordonnancement bi-objectifs dans les systèmes embarqués temps réel

Résumé

L'ordonnancement temps réel constitue une étape pertinente dans une approche de développement de systèmes embarqués temps réel. Les résultats qui en découlent conditionnent fortement la suite du processus de développement. Notre travail étant centré sur l'ordonnancement, plus particulièrement l'ordonnancement bi-objectifs, nous présentons, dans ce chapitre, les approches d'ordonnancement bi-objectifs relatives aux systèmes embarqués temps réel.

Sommaire

4.1 Introduction **78**

4.2 Agrégation des deux objectifs en un seul **79**

4.2.1 Principe de l'approche 79

4.2.2 Quelques travaux d'agrégation des deux objectifs en un seul 80

4.3 Transformation d'un objectif en contrainte 81

4.3.1 Principe de l'approche 81

4.3.2 Quelques travaux de transformation d'un objectif en contrainte 82

4.4 Approche par hiérarchisation 83

4.4.1 Principe de l'approche 83

4.4.2 Quelques travaux adoptant une approche hiérarchique 83

4.5 Approche génétique 84

4.5.1 Principe de l'approche 84

4.5.2 Quelques travaux adoptant une approche génétique 85

4.6 Comparaison d'algorithmes et Métriques 86

4.7 Conclusion 86

4.1 Introduction

Un problème d'affectation ou d'assignement (Assignment Problem) consiste à faire correspondre, d'une manière optimale les éléments de deux ou de plusieurs ensembles où la dimension du problème est exprimée par les ensembles des éléments à associer [Pentico, 2007]. Le problème d'affectation est apparu en premier lieu en 1952 dans le papier de Votaw et Orden [Votaw and Orden, 1952] et suivi plus tard par un certain nombre de travaux [Kuhn, 1955, Kuhn, 2005, frank, 2005].

Dans le domaine de l'ordonnancement temps réel, qui est avant tout un problème d'affectation , deux ensembles sont pris en compte : d'une part, les tâches (opérations) constituant une application, avec des temps d'exécutions spécifiques et d'autre part les processeurs sur lesquelles les tâches seront exécutées. Il s'agit alors d'affecter les tâches aux processeurs de manière à satisfaire certaines contraintes. Ce problème est considéré comme NP-complet [Gary and Johnson, 1979].

Depuis quelques années, l'ordonnancement bi-objectifs connait un véritable essor. Les travaux qui s'y rapportent sont divers et concernent plusieurs domaines dont les systèmes de production [Dhingra and Chandna, 2010, Li et al., 2010], les traitements parallèles [Dutot and Trystram, 2005, Samur and Bulkan, 2010], les grilles de calcul [Garg and Singh, 2011, Canon et al., 2011] . . . etc.

Quant à l'ordonnancement bi-objectifs dans les systèmes embarqués temps réel, les approches proposées [Dogan and Ozguner, 2002, Assayad et al., 2004, Hakem and Butelle, 2006, Qin and Jiang, 2006, Pop et al., 2007] ont pour but d'ordonnancer des tâches sur une architecture multiprocesseur distribuée tel que les deux objectifs considérés soient optimisés.

Les approches de résolution du problème bi-objectifs dans les systèmes embarqués temps réel peuvent être classées en quatre principales catégories : l'approche par agrégation, l'approche de transformation d'un objectif en contrainte, l'approche hiérarchique et enfin les approches évolutionnaires dont particulièrement les algorithmes génétiques.

4.2 Agrégation des deux objectifs en un seul

4.2.1 Principe de l'approche

Dans cette approche dite aussi de scalarisation, la résolution du problème peut être réalisée en fusionnant les objectifs transformant ainsi le problème en un problème mono-objectif. La transformation utilise un paramètre de compromis θ dont les valeurs appartiennent à l'intervalle $[0, 1]$. Les valeurs attribuées au paramètre θ ont pour rôle de donner différents poids aux deux objectifs. A chacune des extrémités de cet intervalle, l'un des deux objectifs est ignoré.

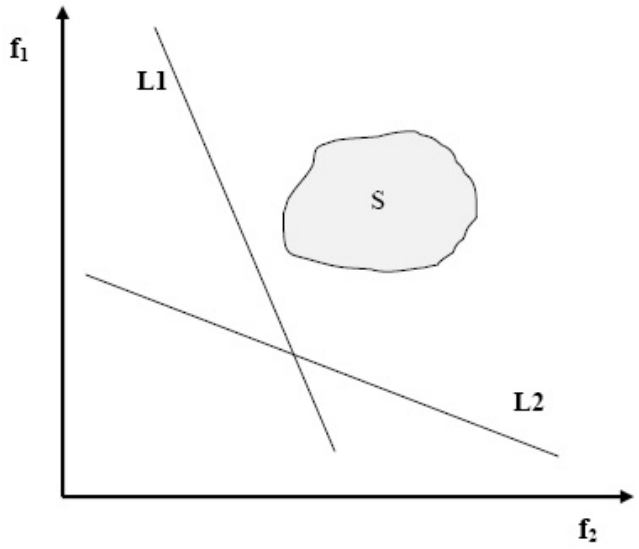

FIGURE 4.1: Méthode de la moyenne arithmétique pondérée.

La moyenne arithmétique pondérée est la technique la plus utilisée. L'idée de base est de définir une fonction coût (fonction-objectif) qui réalise un compromis entre les deux critères. Une telle fonction est sous la forme : $\theta f_1(x) + (\theta - 1) f_2(x)$ où f_1 et f_2 sont les fonctions-objectifs correspondantes respectivement au premier et au second critère.

S (Figure 4.1), représente un ensemble de solutions (ordonnancements) dont les valeurs sont exprimées par celles des fonctions-objectifs f_1 et f_2. L_1 et L_2 représentent les coefficients de pondération. Ces derniers ont un impact direct sur les solutions produites. Cependant, cette approche n'est efficace que pour les problèmes convexes [Canon, 2010].

4.2.2 Quelques travaux d'agrégation des deux objectifs en un seul

Dans l'algorithme d'ordonnancement dynamique *DLS* (Dynamic Level Scheduling) [Sih and Lee, 1993], le placement d'une opération sur un processeur se base sur la valeur de la fonction objectif, celle-ci doit être maximale. Un élément additionnel à la fonction objectif permet de prendre en compte la fiabilité de l'ordonnancement.

Une heuristique d'ordonnancement basée liste est proposée par Dogan [Dogan and Ozguner, 2000] avec une fonction objectif qui concerne aussi bien la longueur d'ordonnancement que la fiabililté. Ce sont également ces deux derniers objectifs qui sont traité par l'heuristique bi-objectifs *RDLS* (Reliability Dynamic Level Scheduling) [Dogan and Ozguner, 2002] qui est une extension de *DLS* [Dogan and Ozguner, 2000]. La fiabilité est prise en compte par une fonction objectif incrémentale permettant à l'algorithme la production d'ordonnancements dynamiques minimisant simultanément les deux objectifs.

L'algorithme *RBSA* (Reliable Bi-Criteria Scheduling Algorithm) [Assayad et al., 2004] utilise la duplication active d'opérations. Une fonction objectif permet d'établir une priorité entre les opérations à ordonnancer ainsi que l'ensemble des processeurs sur lesquels les opérations seront ordonnancées. La duplication active des opérations est la stratégie choisie afin d'améliorer la fiabilité. En plus, une variation de paramètres permet de diversifier les solutions parmi lesquelles le décideur pourra choisir celles qui sont en adéquation avec ses besoins.

Dogan et Ozguner [Dogan and Ozguner, 2005] proposent une extension bi-objective de *DLS* en construisant à chaque étape de l'ordonnancement deux listes contenant tous les couples (tâche t, processeur p) désignant la tâche t prête à être ordonnancée sur le processeur p. Deux fonctions objectifs sont associées aux deux critères. Les rangs relatifs aux deux listes, d'un couple donné, sont combinés où le couple avec le plus petit rang est sélectionné. Dans cette approche, il n'y a pas de duplication des opérations ni de dépendances de données.

Hakem et Butelle proposent *BSA* (Bi-objective Scheduling Algorithm) [Hakem and Butelle, 2006], un

algorithme bi-objectifs meilleur que *RDLS*. Dans *BSA*, les deux objectifs sont combinés en une seule fonction-objectif, le modèle de fiabilité est celui de Shatz et Wang [Shatz et al., 1992] et les tâches ne sont pas dupliquées. La fonction objectif réalise un compromis entre la fiabilité et la longueur d'ordonnancement.

Un algorithme d'approximation [Dongarra et al., 2007], qui est une extension de l'heuristique à base de liste appelée *HEFT* [Topcuouglu et al., 2002], se charge de trouver parmi les ordonnancements qui ne dépassent pas une longueur d'ordonnancement donnée, celui ayant la meilleure fiabilité. L'idée de base est d'ordonnancer les tâches sur les processeurs les plus rapides et les plus fiables et d'utiliser le moins de processeurs possible pour satisfaire la contrainte de temps d'exécution.

Une autre approche d'optimisation simultanée de la fiabilité et de la longueur d'ordonnancement [Jeannot et al., 2008] suppose que les tâches sont indépendantes et les processeurs homogènes. L'algorithme permet de produire une approximation de l'ensemble de Pareto.

Boeres propose une heuristique d'ordonnancement statique [Boeres et al., 2011] optimisant aussi bien la longueur d'ordonnancement que la fiabilité dans les systèmes distribués hétérogènes. Les deux critères sont considérés simultanément et une classification des solutions permet à l'utilisateur d'adapter la fonction objectif à ses besoins.

4.3 Transformation d'un objectif en contrainte

4.3.1 Principe de l'approche

Cette approche consiste à transformer un problème bi-objectifs en un problème mono-objectif. Une fonction prioritaire est choisie pour l'optimisation tandis que l'autre fonction-objectif est considérée comme contrainte. La contrainte permet de délimiter le domaine de recherche des solutions (Figure 4.2).

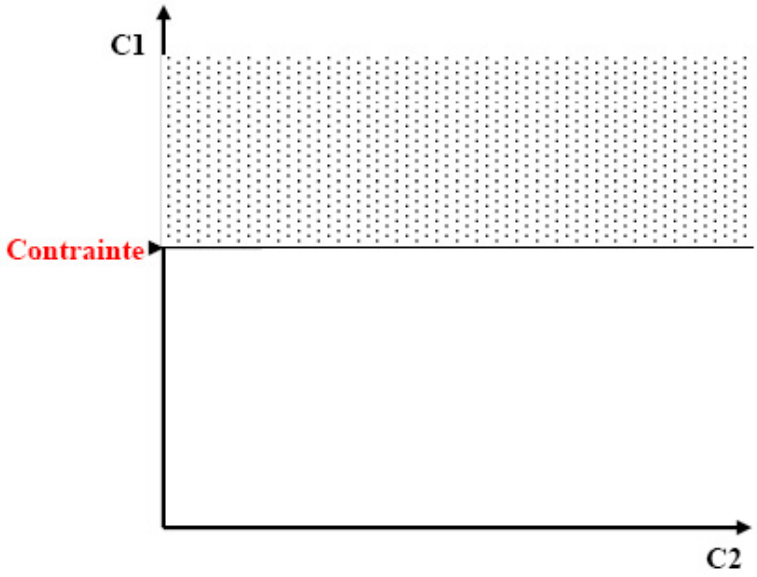

FIGURE 4.2: Transformation d'un objectif en contrainte.

4.3.2 Quelques travaux de transformation d'un objectif en contrainte

La fiabilité est définie via le *GSFR* (*G*lobal *S*ystem *F*ailure *R*ate) [Girault and Kalla, 2009]. Ce dernier est le taux de panne du système considéré comme une seule opération ordonnancée sur un seul processeur. L'algorithme *BSH* (Bi-criteria Scheduling Heuristic) proposé est une heuristique à base de liste. Les résultats montrent que *BSH* produit des ordonnancements où le facteur de duplication des opérations diminue lorsqu'elles sont ordonnancées sur des processeurs plus fiables . En plus, la surcharge induite par les opérations de routage est raisonnable.

TSH [Assayad et al., 2012] est une heuristique d'ordonnancement pour des architectures parallèles hétérogènes. Trois objectifs sont, cette fois-ci, pris en compte, la longueur d'ordonnancement, la fiabilité et la consommation d'énergie. L'heuristique utilise, d'une part, la duplication active des opérations ainsi que les dépendances de données pour l'amélioration de la fiabilité et d'autre part la mise à l'échelle dynamique de la tension afin de minimiser la consommation d'énergie. La longueur d'ordonnancement est minimisée en respectant les contraintes émises sur l'énergie et la fiabilité. Un ensemble de solutions de compromis est produit.

4.4 Approche par hiérarchisation

4.4.1 Principe de l'approche

Dans l'approche hiérarchique (Figure 4.3), l'optimisation des deux objectifs se fait d'une manière alternée. Dans ce cas, un ordre total des objectifs, selon lequel ces derniers seront traités, est établi.

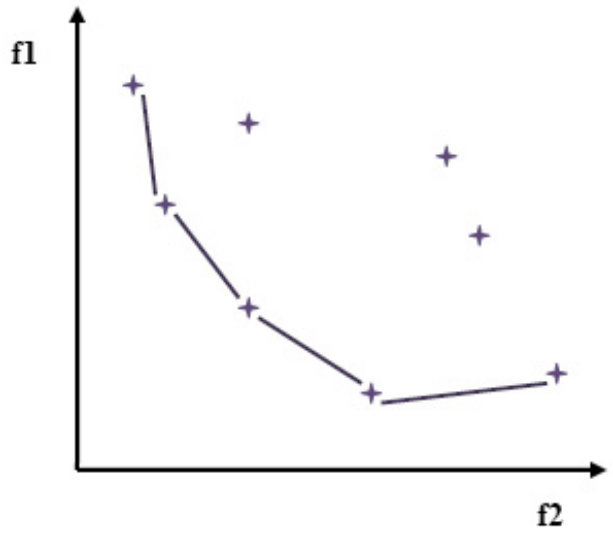

FIGURE 4.3: Approche par hiérarchisation des objectifs.

La répétition du processus permet de produire des solutions non comparables qui, si elles ne coïncident pas avec les solutions optimales de Pareto, elles s'en approchent fortement. Cette approche est de plus en plus utilisée car elle offre une palette de solutions parmi lesquelles le décideur peut choisir celles qui lui conviennent le mieux.

4.4.2 Quelques travaux adoptant une approche hiérarchique

eFRCD (*e*fficient *F*ault-tolerant *R*elaibility *C*ost *D*riven Algorithm) [Qin et al., 2002] est un algorithme qui considère la fiabilité et la date de fin de tâches comme objectifs. Les contraintes de précédence ainsi que l'hétérogénéité des calculs et des communications sont les hypothèses de travail. Toutefois, la priorité est plutôt donnée à la date de fin au détriment de la fiabilité.

Qin [Qin and Jiang, 2006] propose de commencer par ordonnancer les tâches d'une application parallèle selon leurs dates de fin. L'approche est hiérarchique vu que l'identification des processeurs

concerne d'abord ceux maximisant la fiabilité pour enfin sélectionner celui qui minimise la date de début au plus tôt, ce qui favorise la fiabilité.

Un algorithme d'ordonnancement bi-objectifs [Nakechbandi et al., 2006] se base sur l'algorithme *eFRCD* en utilisant la duplication de copies ainsi que les copies de sauvegarde.

Une autre approche [Islam and Suri, 2007] propose une optimisation multivariable pour la conception des systèmes embarqués. La fiabilité est introduite comme premier objectif en limitant les interactions. La longueur d'ordonnancement est alors traitée en seconde position. Une approche de recuit simulé est utilisée pour déterminer les solutions optimales.

L'algorithme d'odonnancement statique [Girault et al., 2009] est constitué de deux étapes. La fiabilité est d'abord maximisée par un algorithme probabiliste pour qu'ensuite, la longueur d'ordonnancement soit à son tour minimisée en appliquant un algorithme d'ordonnancement à base de liste.

4.5 Approche génétique

4.5.1 Principe de l'approche

Les algorithmes génétiques sont de plus en plus utilisés en ordonnancement multi-objectifs. Comme toute approche évolutionnaire, leur but est d'une part la convergence de l'ensemble optimal de Pareto et d'autre part le maintien de la diversité. A base de population, les algorithmes génétiques consistent à exploiter itérativement celle-ci et à l'améliorer en ne conservant à chaque fois que les meilleures solutions.

Une approche génétique débute par un ensemble de solutions aléatoires satisfaisant certaines contraintes. Dans le cas de l'ordonnancement, chaque solution est un ordonnancement ayant les valeurs des deux fonctions objectifs relatives aux deux critères considérés. A chaque itération, des solutions sont sélectionnées à partir de la population et les nouvelles solutions sont obtenues en combinant les solutions courantes. Le mécanisme de sélection est réalisé soit aléatoirement soit selon une stratégie de sélection. La création de nouvelles solutions se fait via le croisement et la mutation. Le critère d'arrêt peut être

le nombre d'itérations ou bien l'obtention de solutions satisfaisant certaines exigences.

4.5.2 Quelques travaux adoptant une approche génétique

Fonseca and Flemming [Fonseca and Flemming, 1993] proposent une approche génétique via l'algorithme génétique MOGA (Multi Objetcive Genetic Algorithm). Le processus inclut le décideur dont l'interaction avec l'algorithme permet de déterminer une solution satisfaisante.

MOGADES (Multi-Objective Genetic Algorithm with Distributed Environment Scheme) [Kamiura et al., 2002] est un algorithme génétique distribué qui utilise des paramètres de poids afin de transformer un problème multi-objectifs en un problème mono-objectif. MOGADES est caractérisé par une résolution parallèle du problème.

NSGA [Srinivas and Deb, 1994] est un algorithme génétique dans lequel le calcul de la fonction objectif permet de séparer la population en plusieurs groupes selon le degré de dominance, au sens de Pareto, de chaque individu. NSGA est également utilisé pour la conception de systèmes embarqués hétérogènes complexes [Rath and Dehuri, 2006]. Une autre approche génétique [Dogan and Ozguner, 2005] consiste à séparer les chromosomes (représentant des solutions) valides de ceux invalides et de les traiter séparément.

Jaadan et al [Jadaan et al., 2009] utilisent une méthode combinant l'algorithme NSGA avec une approche par Pénalité. Il s'agit de pénaliser les solutions non faisables en réduisant leurs valeurs de Fitness proportionnellement à leurs degrés de violation des contraintes. Les résultats sont concluants en démontrant que le nouvel algorithme PP-NRGA produit à chaque fois le meilleur ensemble de Pareto.

D'autres algorithmes d'optimisation multi-objectifs, tels que VEGA [Schaffer, 1984] et SPEA2 [Zitzler et al., 2001b], se basent également sur une approche génétique.

4.6 Comparaison d'algorithmes et Métriques

La production de fronts permet le départage de solutions. Ceci est d'autant plus utile que lorsque s'en suit une comparaison avec d'autres fronts. Quelques algorithmes évolutionnaires sont comparés en utilisant cinq fonctions de test [Grosan and Dumitrescu, 2002]. Les algorithmes sont SPEA (Strength Pareto Evolutionary Algorithm) [Zitzler and Thiele, 1998], PEAS (Pareto Archived Evolution Strategy) [Knowles and Corne, 1999], NSGA II (Non-dominated Sorting Genetic Algorithm) et enfin APA (Adaptive Pareto Algorithm) [Dumitrescu et al., 2001].

Une comparaison [Kamiura et al., 2002] de l'algorithme MOGADES aux algorithmes SPEA2 [Zitzler et al., 2001a] et NSGA II [Deb et al., 2000] est également réalisé. NSGA II, SPEA et Fast PGA [Hyppolite et al., 2008] sont comparés [Dasgupta et al., 2008] avec une version modifiée de KMA (Kuhn-Munkers Algorithm). Ce qui en découle est que les algorithmes évolutionnaires sont capables de produire des solutions compétitives en un temps de calcul raisonnable.

Deux mesures d'exécution [Deb and Jain, 2002] permettent d'évaluer l'évolution des générations dans les algorithmes évolutionnaires. Oyetunji [Oyetunji, 2011] explore le problème d'évaluation de performances des ensembles non dominés produits par des algorithmes d'approximation pour l'ordonnancement multi-objectifs. Les résultats prouvent que les mesures sont d'autant plus significatives que lorsqu'elles sont combinées.

4.7 Conclusion

Ce chapitre a concerné l'ordonnancement bi-objectifs dans les systèmes embarqués temps réel. Différentes approches d'ordonnancement bi-objectifs ont été présentées selon une classification par approche de résolution. Quelque soit l'approche adoptée, le but demeure la production de solutions faisant les meilleurs compromis possibles entre les objectifs considérés. C'est ce à quoi nous aspirons dans le chapitre suivant.

Chapitre 5

Approches proposées

Résumé

Ce chapitre est consacré à notre contribution, nous y présentons les deux approches proposées ainsi que les flots d'implantation obtenus. Dans le premier cas, l'ordonnancement est bi-objectifs adoptant une approche hiérarchique qui consiste en une optimisation alternée des objectifs. Quant au second cas, l'approche est adaptative utilisant un module d'adaptation dont le rôle est l'amélioration de la diversité des solutions. Pour chacune des approches proposées, les expérimentations effectuées sont présentées.

Sommaire

5.1 Méthodologie Adéquation Algorithme Architecture (AAA) 89

5.2 Modèles d'entrée des approches proposées 90

5.2.1 Modèle d'application 90

5.2.2 Modèle de plateforme 91

5.2.3 Contraintes d'exécution et de communication 91

5.2.4 Notations 92

5.2.5 Longueur d'ordonnancement (Makespan) 93

5.2.6 Modèle de fiabilité 94

5.3 AAA-Bi-Hiérarchique 96

5.3.1 Principe de l'approche 96

5.3.2 Algorithme RBB (Reliability-Based Bi-objective scheduling heuristic) 98

5.3.3 Fonctionnement de RBB 99

5.3.4 Algorithme MBB (Makespan-Based Bi-objective scheduling heuristic) 101

5.3.5 Fonctionnement de MBB 102

5.3.6 Expérimentation de l'approche AAA-Bi-Hiérarchique 104

5.4 AAA-Bi-Adaptative 108

5.4.1 Principe de l'approche 108

5.4.2 Algorithme ARBB (Adaptive Reliability-Based Bi-objective scheduling heuristic) 111

5.4.3 Fonctionnement de ARBB 112

5.4.4 Procédure d'adaptation 112

5.4.5 Fonctionnement de la procédure d'adaptation 113

5.4.6 Algorithme AMBB (Adaptive Makespan-Based Bi-objective scheduling heuristic) 114

5.4.7 Fonctionnement de AMBB 115

5.4.8 Flot d'implantation 116

5.4.9 Expérimentation de l'approche AAA-Bi-Adaptative 117

5.5 Conclusion 121

5.1 Méthodologie Adéquation Algorithme Architecture (AAA)

La méthodologie Adéquation Algorithme Architecture (AAA) est une approche développée au sein de l'INRIA. Elle vise le prototypage rapide et l'implantation optimisée d'applications distribuées temps réel embarquées. La méthodologie AAA est fondée sur un modèle unifié de graphes factorisés servant à spécifier aussi bien l'algorithme et l'architecture multicomposant, qu'à déduire les implantations possibles en termes de transformations de graphes.

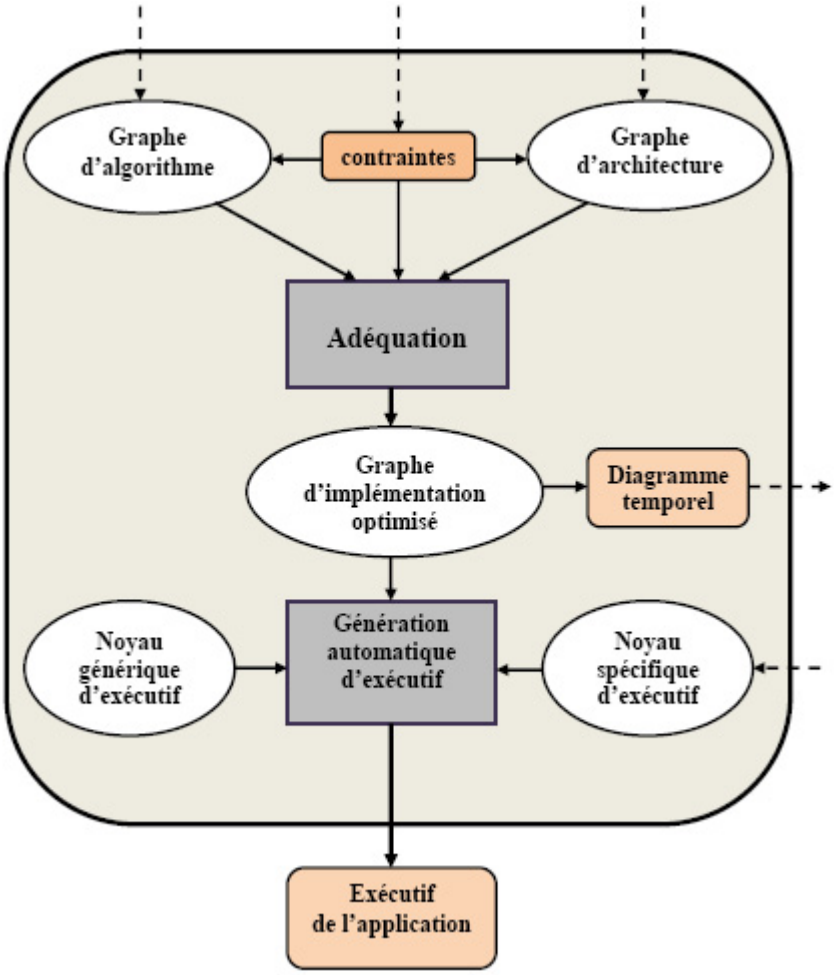

FIGURE 5.1: Flot d'implantation.

En effet, dans la méthodologie AAA, le modèle d'algorithme est un graphe de dépendance de données qui est hiérarchique, conditionné et factorisé [Lavarenne and Sorel, 1997]. L'algorithme est modélisé par un graphe de dépendance infiniment répété et identifié. Le graphe est réduit par factorisation à son motif répété appelé *graphe flots de données* [Dennis, 1975]. D'autre part, le modèle d'architecture

supposé être hétérogène multicomposant, est un graphe dont chaque sommet est une machine à états finis et chaque arc une connexion physique entre deux machines à états finis.

Le graphe de flot de données modélisant l'algorithme est transformé progressivement jusqu'à ce qu'il corresponde au graphe modélisant l'architecture. Ce processus représente aussi bien une allocation spatiale (*distribution*) qu'une allocation temporelle (*ordonnancement*). Il en résulte *un exécutif temps réel distribué* supportant l'implantation de l'algorithme spécifié sur l'architecture en question. L'ordonnancement concerne non seulement les opérations de l'algorithme sur les opérateurs de l'architecture mais également les opérations de communication sur les différents liens de communication.

5.2 Modèles d'entrée des approches proposées

L'ordonnancement temps réel est un problème multi-objectifs. Sa résolution nécessite, en plus des modèles d'entrée utilisés, la définition des objectifs traités. Que ce soit pour la première approche (hiérarchique) ou la seconde (adaptative), l'algorithme et l'architecture sont modélisés par des graphes de flots de données.

5.2.1 Modèle d'application

L'application qu'on désire implémenter sur une architecture donnée est appelée 'Algorithme'. Elle est modélisée par un graphe de flots de données (Graphe d'algorithme) noté ALG. Chaque sommet du graphe est une tâche qui accomplit une certaine fonctionnalité tandis que chaque arc représente une relation entre deux tâches. C'est un lien de dépendance noté $\rightarrow$, qui exprime un transfert de données entre une tâche productrice de données et une tâche consommatrice de données.

$t1 \rightarrow t2$ signifie que $t1$ est un prédécesseur de $t2$ et $t2$ est un successeur de $t1$. Les tâches sans prédécesseurs (respectivement sans successeurs) sont des tâches d'entrée (respectivement de sortie) gérant les événements produits par les récepteurs (respectivement les émetteurs). La figure 5.2 représente un exemple d'application constituée de sept tâches.

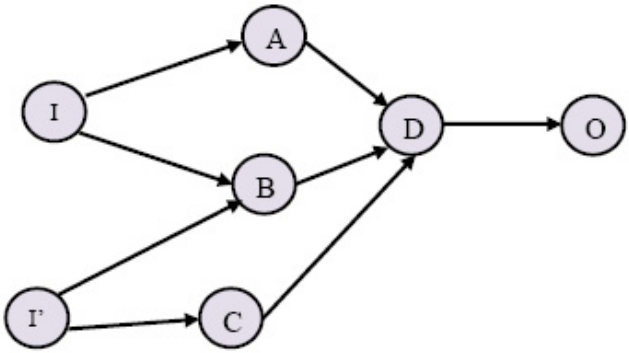

FIGURE 5.2: Exemple de graphe d'algorithme.

5.2.2 Modèle de plateforme

La plateforme ou l'architecture cible est modélisée par un graphe non orienté, noté ARC, dans lequel chaque nœud est un processeur et chaque arc est une liaison point à point.

D'ordinaire, un processeur est constitué d'une unité de calcul, d'une mémoire locale et d'une ou de plusieurs unités de communication.

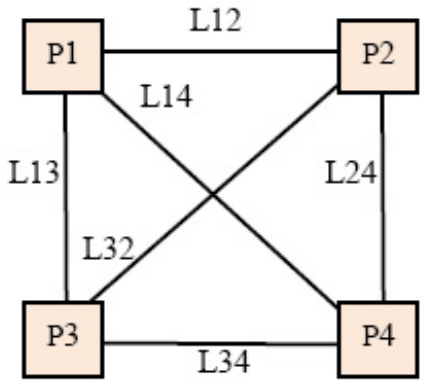

FIGURE 5.3: Exemple de graphe d'architecture.

5.2.3 Contraintes d'exécution et de communication

En plus de l'algorithme et de l'architecture, d'autres données tout aussi importantes sont à prendre en considération. Il s'agit des temps d'exécution ainsi que des temps de communication ou de transmission des données.

T_{exe}	I	$I^{'}$	A	B	C	D	O
P1, P3	4	3	2	6	8	4	4
P2, P4	6	5	3	8	2	4	6

TABLE 5.1: Exemple de temps d'exécution.

Un temps d'exécution est défini pour chaque paire (t_i, p_j), il correspond au pire temps d'exécution de la tâche t_i sur le processeur p_j. En supposant que les processeurs sont hétérogènes, une tâche pourrait avoir des temps d'exécution différents selon le processeur sur lequel elle s'exécute. Un cas particulier est celui où une tâche ne peut s'exécuter sur un processeur donné, auquel cas l'association est exprimée par la valeur "∞".

D'autre part, à chaque paire $(t_i \mapsto t_j, L_{nm})$, est associé un temps qui exprime le pire temps de transfert de données $t_i \mapsto t_j$ sur le lien de communication L_{nm}. Quant au temps de communication intra-processeur, il est supposé être nul.

T_{com}	$I \mapsto A$, $I \mapsto B$	$I^{'} \mapsto B$, $I^{'} \mapsto C$	$A \mapsto D$, $B \mapsto D$, $C \mapsto O$
L12, L13	4	6	2
L21, L24	8	5	3
L34	6	4	2

TABLE 5.2: Exemple de temps de communication.

Les tableaux 5.1 et 5.2 sont respectivement des exemples de temps d'exécution et de communication pour l'algorithme *ALG* et l'architecture *ARC* des figures 5.2 et 5.3.

5.2.4 Notations

Le tableau 5.3 contient les notations [Kalla et al., 2004] utilisées dans nos approches. Le numéro exposant entre parenthèses désigne l'étape de l'heuristique.

Libellé	*Sémantique*
$T_{conc}^{(n)}$	Liste des tâches concurrentes (Une tâche est dite concurrente si tous ses prédécesseurs sont déjà ordonnancés)
$T_{sched}^{(0)}$	La liste des tâches déjà ordonnancées
$S_{best}^{(n)}(t_i, p_j)$	La date au plus tôt à laquelle la tâche t_i peut commencer l'exécution sur le processeur p_j
$S_{worst}^{(n)}(t_i, p_j)$	La date au plus tard à laquelle la tâche t_i peut commencer l'exécution sur le processeur p_j
$\bar{S}^{(n)}(t_i)$	La date de début au plus tard depuis la fin de t_i
$pred(t_i)$	L'ensemble des prédécesseurs de la tâche t_i
$succ(t_i)$	L'ensemble des successeurs de la tâche t_i
$R^{(n)}$	La longueur du chemin critique

TABLE 5.3: Sémantique des notations.

5.2.5 Longueur d'ordonnancement (Makespan)

Dans notre approche, l'ordonnancement est hors ligne, ce qui signifie que les temps de début d'exécution de chaque tâche sur chaque processeur ainsi que les temps de transfert des données sur chaque lien de communication sont connus a priori. La longueur d'ordonnancement (schedule lenght ou Makespan) est le temps de fin d'exécution de la tâche qui, parmi toutes les tâches, s'est terminée en dernier. Il est défini comme suit :

$$M = \max_{p_j} \left\{ \max_{t_i on\ p_j} end(t_i, p_j) \right\} \tag{5.1}$$

où $end(t_i, p_j)$ est le temps de terminaison d'exécution de la tâche t_i sur le processeur p_j.

La longueur d'ordonnancement est un objectif à minimiser. La fonction de *pression d'ordonnancement* (*schedule pressure*) [Sorel, 1994], notée $\sigma^{(n)}$ est définie pour chaque opération $t_i \in T_{conc}^{(n)}$ et chaque

processeur p_j comme suit :

$$\sigma^{(n)}\left(t_i, p_j\right) = S^{(n)}_{best(t_i,p_j)} + \bar{S}^{(n)}_{t_i} \tag{5.2}$$

$$\sigma^{(n)}\left(t_i, p_j\right) = S^{(n)}_{best(t_i,p_j)} + t_{exe} + t_{comm} + \bar{S}^{(n)}_{t_i} \tag{5.3}$$

Elle est utilisée pour sélectionner la tâche qui minimise la longueur du chemin critique.

Une variante de la pression d'ordonnancement, notée $\bar{\sigma}_{(n)}$ est calculée comme suit :

$$\bar{\sigma}^{(n)}\left(t_i, p_j\right) = S^{(n)}_{worst(t_i,p_j)} + \bar{S}^{(n)}_{t_i} \tag{5.4}$$

L'utilisation de S_{worst} permet la réduction de la longueur d'ordonnancement en présence de pannes de N composants arbitraires.

La *pression d'ordonnancement sûre* (*dependable schedule pression*) [Kalla et al., 2004] notée $\delta^{(n)}$ est une fonction coût servant à définir la priorité entre les tâches. Elle est calculée pour chaque tâche $t_i \in t^{(n)}_{conc}$ et un ensemble de processeurs $P_{taches} \subset P$ comme suit :

$$\delta^{(n)}\left(t_i, P_{taches}\right) = \underset{p_j \in P_{taches}(t_i)}{max}\ \bar{\sigma}^{(n)}\left(t_i, p_j\right) \tag{5.5}$$

$$\text{où } P_{taches}\left(t_i\right) = \left\{ p_j / p_j \in \underset{p_j \in P}{\overset{N+1}{min}}\ \bar{\sigma}^{(n)}\left(t_i, p_j\right) \right\}$$

La *pression d'ordonnancement sûre* mesure à quel point l'ordonnancement d'une tâche augmente le chemin critique d'un algorithme en absence de panne. Nous l'utiliserons comme fonction coût pour estimer la longueur d'ordonnancement.

5.2.6 Modèle de fiabilité

La fiabilité est une caractéristique qui concerne aussi bien les processeurs que les liens de communication. Selon le modèle de fiabilité le plus adopté [Shatz et al., 1992], les pannes sont transitoires et la

durée maximale d'une panne n'affecte que la tâche courante s'exécutant sur le processeur où la panne est survenue. Ceci est également valable pour les liens de communication. En plus, l'occurrence de pannes suit une loi Poisson avec un paramètre constant λ.

La fiabilité d'un système mesure finalement sa continuité de service. La fiabilité d'un processeur P (respectivement d'un lien de communication L) pendant la durée d est [Girault and Kalla, 2009] :

$$R = e^{-\lambda d}. \tag{5.6}$$

Par conséquent, le taux de panne d'un processeur P (respectivement d'un lien de communication L) est :

$$F = 1 - R = 1 - e^{-\lambda d}. \tag{5.7}$$

Plus particulièrement, la fiabilité d'une tâche ou d'une dépendance de données X placée sur un composant hardware C est :

$$R\left(X, C\right) = e^{-\lambda_c exe(X,C)} \tag{5.8}$$

L'utilisation de la fiabilité et de la longueur d'ordonnancement comme critères entraîne des difficultés techniques. Ceci est dû au fait que la fiabilité dépend intrinsèquement de la durée des tâches et des communications [Girault and Kalla, 2009]. Par conséquent, plutôt que d'utiliser le modèle classique de la fiabilité [Shatz et al., 1992], nous utiliserons le concept du *GSFR* (Global System Failure Rate) [Girault and Kalla, 2009]. Le *GSFR*, noté $\Lambda(S)$, est le taux de panne par unité de temps de l'ordonnancement multi-processeurs obtenu, il est défini comme suit :

$$\Lambda(S) = -\frac{\log R(S)}{U(S)} \tag{5.9}$$

où $U(S) = \sum_{(t_i,p)\in S} Exe\left(t_i, p\right)$ est l'utilisation totale des ressources hardware et R sa fiabilité.

Une telle formulation de la fiabilité signifie que le système est vu comme une seule tâche exécutée sur

une machine. De cette manière, le taux de panne ne dépend plus de la durée de la tâche. Maximiser la fiabilité consiste en une minimisation du GSFR.

D'autres facteurs sont également à prendre en compte, ce sont les taux de panne relatifs aux processeurs ainsi qu'aux liens de communication. De telles données d'entrée permettent d'évaluer la fiabilité du système.

P1,P2	P3,P4	L12, L14, L32	L13, L24, L34
10^{-5}	10^{-7}	10^{-6}	10^{-4}

TABLE 5.4: Exemple de taux de panne de processeurs/liens.

Le tableau 5.4 représente un exemple de taux de pannes d'un ensemble de processeurs et des liens de communication les reliant.

5.3 AAA-Bi-Hiérarchique

Dans l'approche AAA-Bi-Hiérarchique [Bendib et al., 2013], deux objectifs sont pris en compte. Il s'agit de la fiabilité et de la longueur d'ordonnancement. Étant donné un algorithme (application) et une architecture cible, le but est de produire des ordonnancements réalisant de bons compromis entre les deux objectifs.

5.3.1 Principe de l'approche

Dans le problème d'ordonnancement initial (Figure 5.4), les entrées sont une application ALG, une architecture ARC ainsi qu'un ensemble de contraintes d'exécution et de communication. En plus, une contrainte de départ est injectée au problème afin de limiter l'espace de recherche.

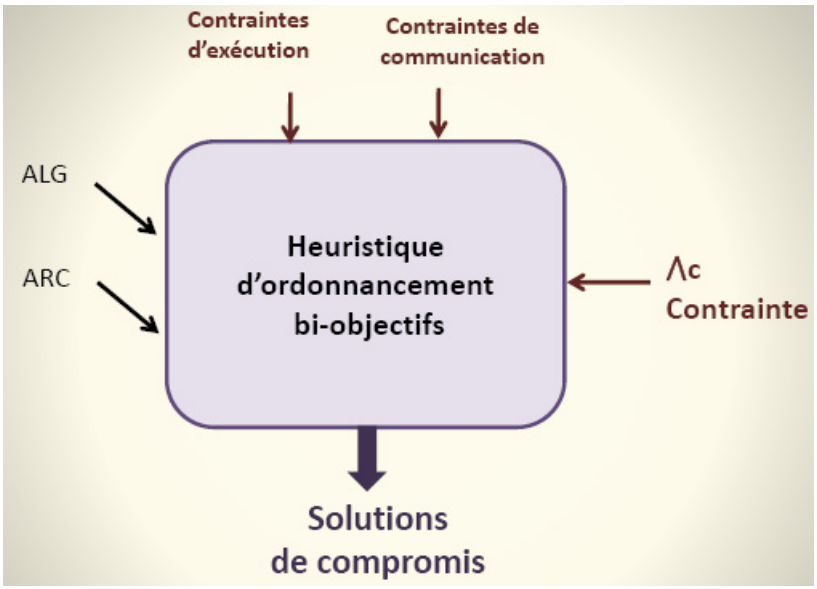

FIGURE 5.4: Problème initial.

Nous proposons de transformer le problème en deux sous-problèmes (Figure 5.5), chacun d'eux traité par une heuristique. Les deux heuristiques coopèrent afin de produire un ensemble de solutions.

La première heuristique RBB (Reliability-Based Bicriteria scheduling heuristic) prend en entrée l'algorithme ALG, l'architecture ARC et une valeur de fiabilité Λ_c comme une contrainte. L'objectif est de minimiser la longueur d'ordonnancement. L'heuristique utilise la fonction coût nommée *dependable schedule pressure* (cf. sous-section 5.2.5, équation 5.5). La solution est un ordonnancement ayant les valeurs (L_{RBB}, Λ_{RBB}) des fonctions objectifs.

La seconde heuristique MBB (Makespan-Based Bi-objective scheduling heuristic) dispose des entrées suivantes : l'algorithme ALG, l'architecture ARC et la longueur d'ordonnancement L_{RBB}, produite par l'heuristique RBB, comme une contrainte. Le but de MBB est de maximiser la fiabilité. La fonction objectif est, dans ce cas, le GSFR (cf. sous section 5.2.6, équation 5.9) et le résultat est un ordonnancement ayant pour valeurs (L_{MBB}, Λ_{MBB}).

En faisant varier la contrainte initiale et en répétant le processus, plusieurs solutions sont produites. Le choix des solutions est réalisé en accord avec les besoins courants.

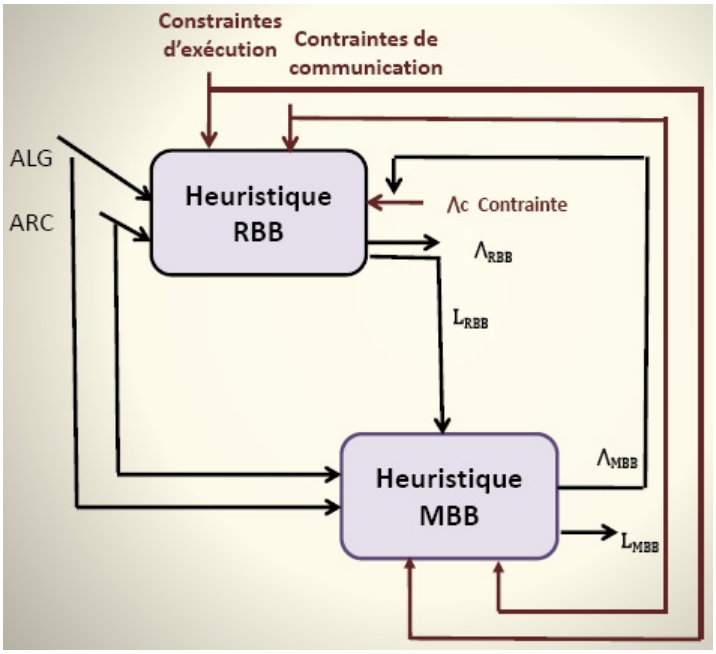

FIGURE 5.5: Principe de l'approche hiérarchique.

5.3.2 Algorithme RBB (Reliability-Based Bi-objective scheduling heuristic)

Basée liste, cette première heuristique émet l'hypothèse selon laquelle la recherche est contrainte par une valeur de GSFR, correspondante donc à une valeur de fiabilité. Le but est de produire un couple de valeurs respectant la contrainte de fiabilité imposée et ayant la valeur optimale de la longueur d'ordonnancement et ce pour une instance du problème.

L'algorithme RBB

Entrées : ALG, ARC, Λ_c

Sorties : L_{RBB}, Λ_{RBB}

Début

Initialiser les listes des tâches concurrentes et des tâches ordonnancées :

$T^{(0)}_{conc} := \{t \in T \mid pred(t) = \emptyset\}$;

$T^{(0)}_{sched} := \emptyset$;

Tant que $T^{(n)}_{conc} \neq \emptyset$ **Faire**

① Calculer la pression d'ordonnancement sûre pour chaque tâche t_i de $T^{(n)}_{conc}$ sur chaque processeur p_j tel que $\Lambda_i < \Lambda_c$;

② Ordonnancer le couple (t_i, p_j) qui minimise la pression d'ordonnancement sûre ;

③ Mettre à jour les listes des tâches concurrentes et des tâches ordonnancées :

$T^{(n)}_{sched} := T^{(n-1)}_{sched} \cup \{t\}$;

$T^{(n+1)}_{conc} := T^{(n)}_{conc} - \{t\} \cup \{t' \in succ(t) \mid pred(t') \subseteq T^{(n)}_{sched}\}$;

Fin Tant que

Fin

$T^{(0)}_{sched}$ est la liste des tâches ordonnancées, $pred(t_i)$ est l'ensemble des prédécesseurs de la tâche t_i, et $succ(t_i)$ est l'ensemble des successeurs de la tâche t_i.

5.3.3 Fonctionnement de RBB

— ***Entrées de RBB***

Les entrées dont dispose l'heuristique RBB sont :

1. Une application ALG sous forme d'un graphe de tâches ;
2. Une architecture ARC sous forme d'un ensemble de processeurs reliés entre eux par des liens de communication ;
3. Une valeur du GSFR, jouant le rôle d'une contrainte et qui de ce fait limite l'espace de

recherche ;

— ***Sorties de RBB***

Les sorties de l'heuristique RBB consistent en deux valeurs à savoir :

1. Une valeur du GSFR exprimant la fiabilité du système ;
2. Une valeur de la longueur d'ordonnancement ou Makespan ;

Ces deux valeurs sont les valeurs respectives des deux fonctions objectifs, elles expriment l'évaluation de l'ordonnancement correspondant.

— ***Initialisation***

Deux listes sont utilisées :

1. $T_{conc}^{(0)}$: est la liste des tâches concurrentes sachant qu'une tâche est considérée comme concurrente si elle n'a pas de prédécesseur ;
2. $T_{sched}^{(0)}$: la liste des tâches ordonnancées et qui vont constituer, à la fin, l'ordonnancement résultat ;

— ***Evaluation***

L'étape d'évaluation (1) consiste à calculer, pour chaque tâche concurrente, sa pression d'ordonnancement sûre sur chacun des processeurs tel que la contrainte courante relative au GSFR soit respectée.

— ***Choix***

Cette étape (2) consiste à réaliser un choix concernant le meilleur couple (tâche, processeur), donc celui qui minimise la pression d'ordonnancement sûre.

— ***Mise à jour***

L'étape de mise à jour (3) consiste d'une part à ajouter la tâche choisie à la liste des tâches ordonnancées et d'autre part à enlever la tâche choisie de la liste des tâches concurrentes ainsi que tous ses successeurs tel que les prédécesseurs de ces derniers soient déjà ordonnancés.

Les étapes (1), (2) et (3) de l'heuristique sont répétées jusqu'à ce qu'il n'y ait plus de tâche concurrente.

Le résultat est un couple de valeurs (L_{RBB}, Λ_{RBB}) contraint par une valeur de GSFR et dont la valeur de la longueur d'ordonnancement est la plus optimale.

5.3.4 Algorithme MBB (Makespan-Based Bi-objective scheduling heuristic)

L'heuristique d'ordonnancement MBB est également basée liste. Elle est, cependant, contrainte par une valeur de la longueur d'ordonnancement produite par l'heuristique RBB.

L'Algorithme MBB

Entrées : ALG, ARC, L_{RBB}

Sorties : L_{MBB}, Λ_{MBB}

Début

Initialiser les listes des tâches concurrentes et des tâches ordonnancées :

$T_{conc}^{(0)} := \{t \in T \mid pred(t) = \emptyset\}$;

$T_{sched}^{(0)} := \emptyset$;

Tant que $T_{conc}^{(n)} \neq \emptyset$ **Faire**

① Calculer le GSFR pour chaque tâche t_i de $T_{conc}^{(n)}$ sur chaque processeur p_j tel que $L_i < L_{RBB}$;

② Ordonnancer le couple (t_i, p_j) qui minimise le GSFR ;

③ Mettre à jour les listes des tâches concurrentes et des tâches ordonnancées :

$T_{sched}^{(n)} := T_{sched}^{(n-1)} \cup \{t\}$;

$T_{conc}^{(n+1)} := T_{conc}^{(n)} - \{t\} \cup \{t' \in succ(t) \mid pred(t') \subseteq T_{sched}^{(n)}\}$;

Fin Tant que

Fin

Son objectif est de produire, dans l'espace limité par la contrainte de longueur d'ordonnancement, le couple de valeurs dont celle du GSFR est la plus optimale pour l'instance en cours.

5.3.5 Fonctionnement de MBB

— ***Entrées de MBB***

Les trois entrées de l'heuristique MBB, dont les deux premières sont les mêmes que celles de l'heuristique RBB, sont :

1. Une application ALG sous forme d'un graphe de tâches ;
2. Une architecture ARC sous forme d'un ensemble de processeurs reliés entre eux ;
3. Une valeur de longueur d'ordonnancement (L_{RBB}) issue de l'heuristique RBB et qui est considérée par l'heuristique MBB comme une contrainte à respecter ;

— ***Sorties de MBB***

Les sorties de l'heuristique MBB consistent en deux valeurs à savoir :

1. Une valeur de la longueur d'ordonnancement ou Makespan ;
2. Une valeur du GSFR exprimant la fiabilité du système ;

Comme pour RBB, les deux valeurs expriment une solution qui est, dans notre cas, un ordonnancement.

— ***Initialisation***

Pour MBB, deux listes sont également utilisées :

1. $T^{(0)}_{conc}$: est la liste des tâches concurrentes sachant qu'une tâche est considérée comme concurrente si elle n'a pas de prédécesseur ;
2. $T^{(0)}_{sched}$: la liste des tâches ordonnancées et qui vont constituer, à la fin, l'ordonnancement résultat ;

— ***Evaluation***

L'étape d'évaluation (1) consiste à calculer le GSFR pour chaque tâche concurrente sur chacun des processeurs tel que la valeur (L_{RBB}) de la longueur d'ordonnancement, issue de l'heuristique RBB, soit respectée.

— ***Choix***

L'étape (2) consiste à réaliser un choix concernant le couple (tâche, processeur) dont la valeur du GSFR est la plus petite, ce qui implique une plus grande fiabilité.

— ***Mise à jour***

Comme pour l'heuristique RBB, l'étape de mise à jour (3) consiste d'une part à ajouter la tâche choisie à la liste des tâches ordonnancées et d'autre part à enlever la tâche choisie de la liste des tâches concurrentes ainsi que tous ses successeurs tel que les prédécesseurs de ces derniers soient déjà ordonnancés.

Les étapes (1), (2) et (3) de l'heuristique sont répétées jusqu'à ce qu'il n'y ait plus de tâche concurrente. Les deux heuristiques RBB et MBB peuvent être ré-exécutées en changeant la contrainte initiale λ_c. Leur coopération permet d'améliorer itérativement les solutions (Figure 5.6).

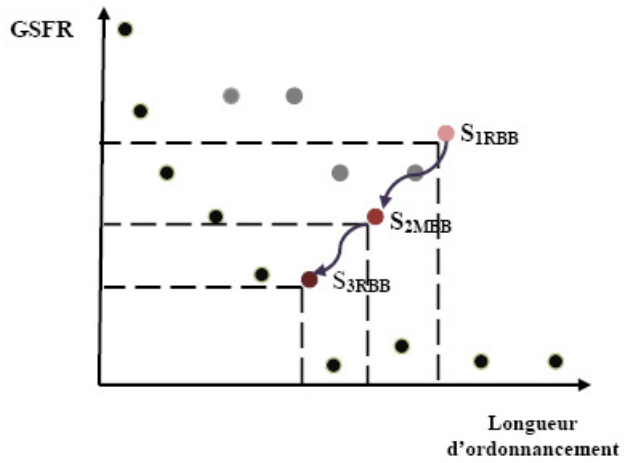

FIGURE 5.6: Restriction itérative de l'espace des objectifs.

L'objectif est de produire des couples de valeurs exprimant des ordonnancements et approximant le plus possible le front de Pareto.

L'application de l'approche hiérarchique dans l'étape d'adéquation de la méthodologie AAA produit un nouveau flot partiel d'implantation (Figure 5.7).

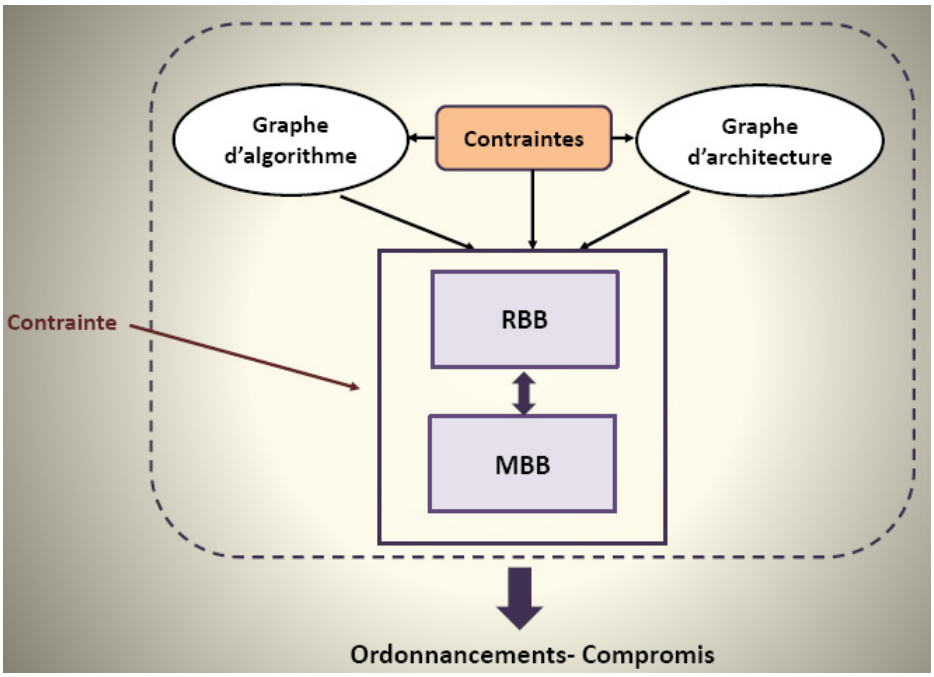

FIGURE 5.7: Flot partiel d'implantation.

Celui-ci prend en considération, en plus des contraintes imposées, la longueur d'ordonnancement et la fiabilité. Le résultat est un ensemble de solutions dont le choix est en adéquation avec les besoins du décideur.

5.3.6 Expérimentation de l'approche AAA-Bi-Hiérarchique

Afin d'évaluer l'approche hiérarchique, nous l'avons implémentée dans l'outil SynDEx, un outil CAO pour l'optimisation et l'implémentation de systèmes embarqués temps réel (http ://www.syndex.org). La comparaison a été réalisée avec l'heuristique RBSA (Reliable Bi-criteria Scheduling Heuristic) proposée au sein de l'INRIA.

Dans ces simulations, nous avons étudié l'impact du nombre de tâches N, du nombre de processeurs P et du ratio CCR (Computation to Communication Ratio) sur la longueur d'ordonnancement et la fiabilité.

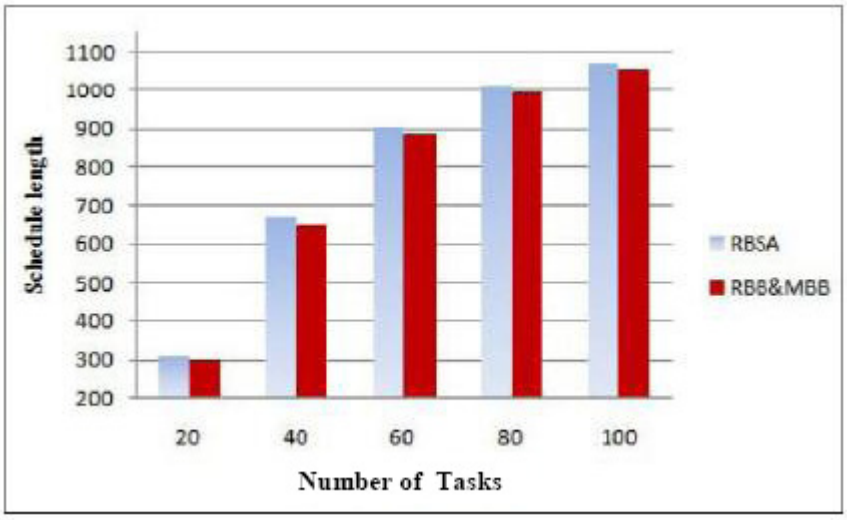

FIGURE 5.8: Impact de N sur le Makespan pour CCR=1 et P=4.

Les Figures 5.8 et 5.9 montrent les variations de la longueur d'ordonnancement et de la fiabilité. Nous pouvons constater dans la figure 5.8 que la longueur d'ordonnancement croit lorsque le nombre de tâches croit et que dans la figure 5.9, la fiabilité décroit lorsque le nombre de tâches croit.

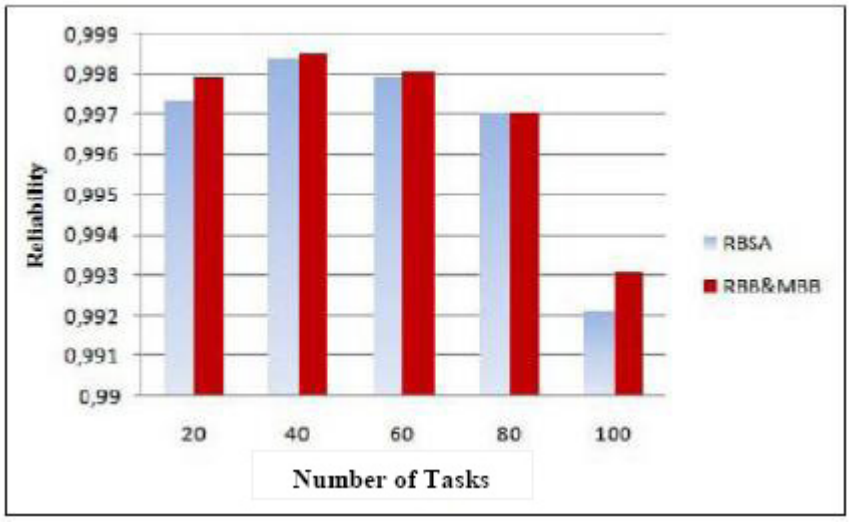

FIGURE 5.9: Impact de N sur la fiabilité pour CCR=1 et P=4.

Ceci était prévisible vu que les deux objectifs, longueur d'ordonnancement et fiabilité, sont antagonistes.

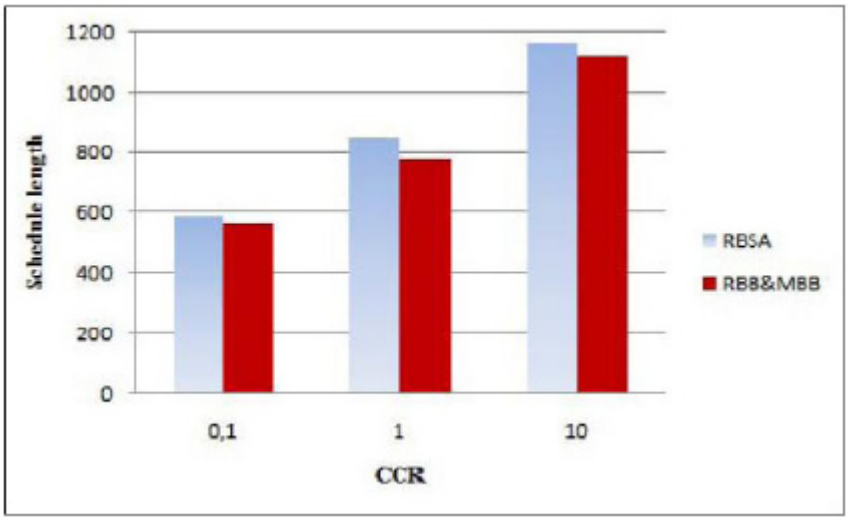

FIGURE 5.10: Impact du CCR sur le Makespan pour N=50 et P=4.

L'impact du CCR sur la longueur d'ordonnancement et la fiabilité est illustré par les figures 5.10 et 5.11 où il est constaté qu'autant pour le premier objectif que pour le second, l'approche proposée MBB-RBB fournit de meilleurs résultats.

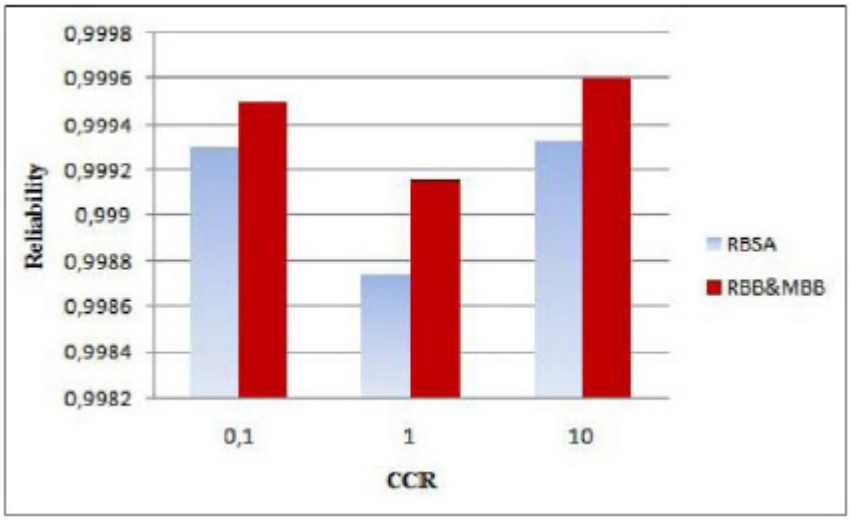

FIGURE 5.11: Impact du CCR sur la fiabilité pour N=50 et P=4.

Ensuite, nous avons appliqué nos heuristiques MBB-RBB ainsi que l'heuristique RBSA à un ensemble de graphes d'algorithme et de graphes d'architecture générés aléatoirement, ces derniers étant composés respectivement de 4, 5 et 6 processeurs. Le générateur de graphes utilisé [Girault and Kalla, 2009] est aléatoire.

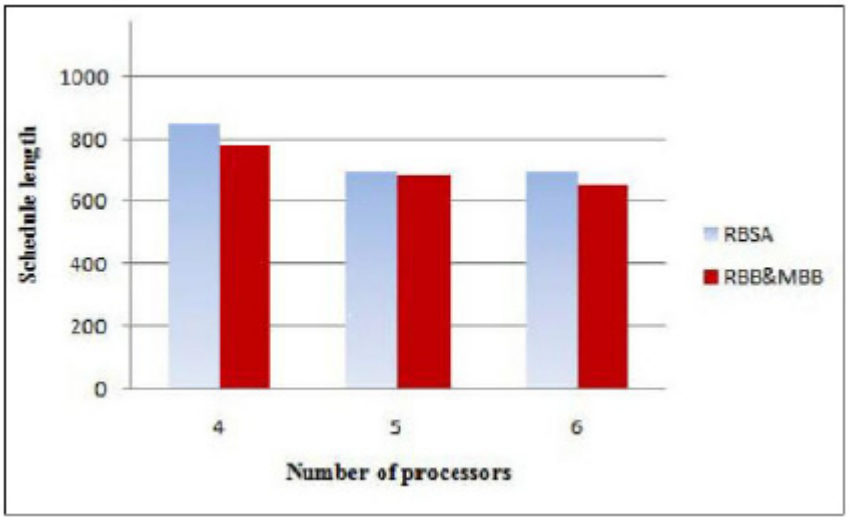

FIGURE 5.12: Impact de P sur le Makespan pour N=50 et CCR=1.

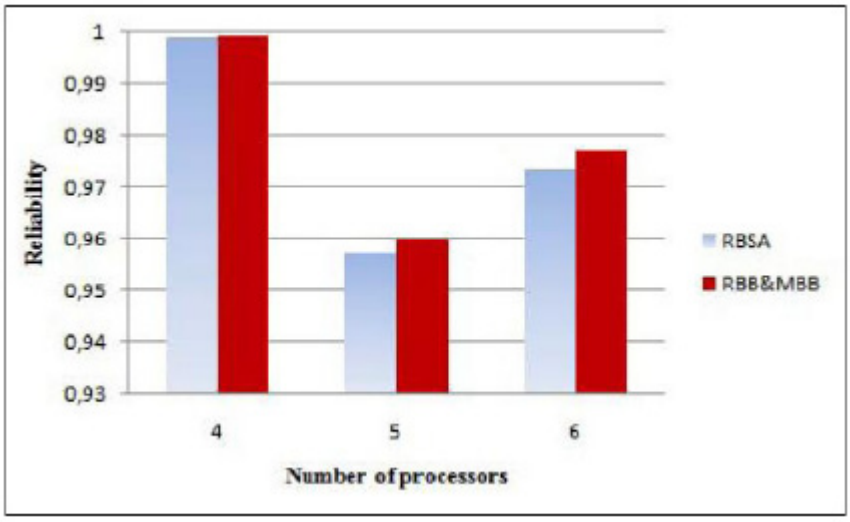

FIGURE 5.13: Impact de P sur la fiabilité pour N=50 et CCR=1.

Les simulations effectuées montrent que notre approche fournit de meilleurs compromis que ceux produits par RBSA. Cependant, il est constaté un manque au niveau de la diversité des solutions.

5.4 AAA-Bi-Adaptative

Dans cette seconde approche, nous traitons en plus le problème de diversité des solutions. A cette fin, nous introduisons un composant que nous appelons 'module d'adaptation'. Ce dernier consiste à ajuster une solution pour ensuite relancer la recherche. Pour ce faire, la notion de 'Voisinage' (cf. chapitre 3, sous-section 3.3.1) est utilisée, elle permet d'étendre l'espace de recherche. La notion de Voisinage est un concept important en optimisation combinatoire, il est défini comme suit :

Définition : Le voisinage d'une solution $s \in S$ est un ensemble de configurations ou solutions de S qui sont directement atteignables par une transformation donnée de s. Il est noté $V(s)$ et une solution $s^{'} \in V(s)$ est dite voisine de s.

Particulièrement, à partir d'une solution donnée, plusieurs structures de voisinage peuvent être établies selon différentes transformations où une transformation est définie par une application :

$$V : S \longrightarrow P(S)$$

où S est un ensemble de solutions et $P(s)$ est un sous-ensemble de S.
Un tel concept est utile dans la structuration de l'espace de recherche ainsi que dans la définition de l'ensemble des solutions qui sont atteignables à partir d'une solution donnée et ce via une série de transformations.

5.4.1 Principe de l'approche

Basée sur l'approche hiérarchique que nous avons proposée [Bendib et al., 2013], l'approche adaptative consiste en une coopération de deux heuristiques (Figure 5.14) qui permet de traiter hiérarchiquement les deux objectifs.

L'heuristique ARBB (Adaptive Reliability Based Bi-objective heuristic) dispose des entrées suivantes : l'application ALG, la plateforme ARC et une valeur de GSFR Λ_c comme une contrainte. L'heuristique ARBB utilise également la pression d'ordonnancement sûre (*dependable schedule pressure*)

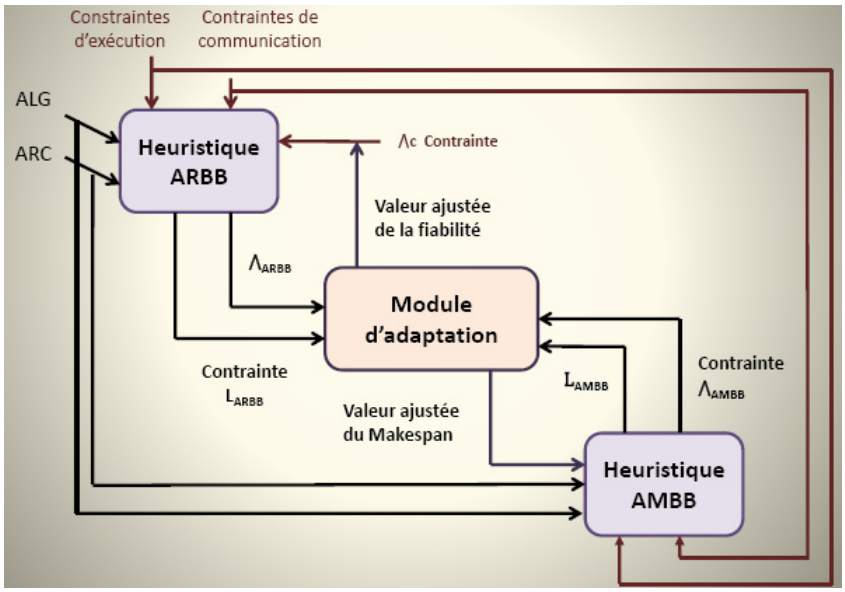

FIGURE 5.14: Principe de l'approche adaptative.

(cf. sous-section 5.2.5, équation 5.5) comme fonction coût dans le but de minimiser la longueur d'ordonnancement. L'exécution de ARBB produit une solution ayant les valeurs (L_{ARBB}, Λ_{ARBB}) qui sont respectivement celles de la longueur d'ordonnancement et du GSFR.

De même, l'heuristique AMBB (Adaptive Makespan Based Bi-objective heuristic) a comme entrées l'application ALG, la plateforme ARC ainsi qu'une solution produite par l'heuristique ARBB. En plus, L_{ARBB} est considérée comme une contrainte afin de maximiser la fiabilité. Le résultat est un ordonnancement avec les valeurs (L_{AMBB}, Λ_{AMBB}).

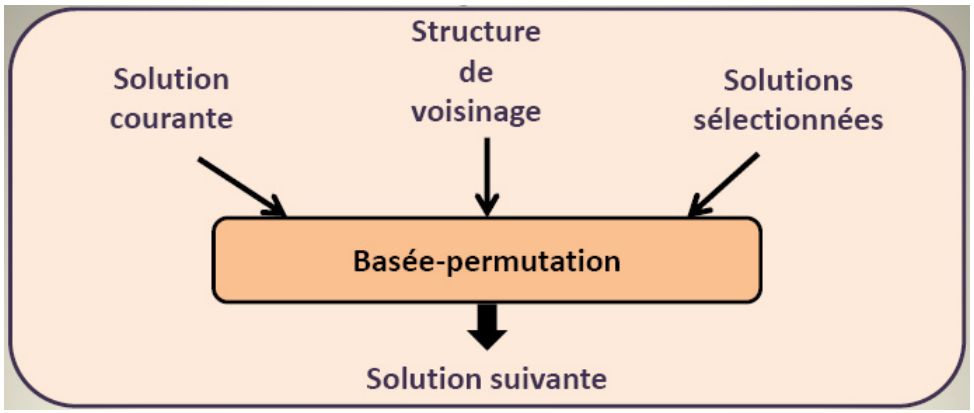

FIGURE 5.15: Principe du module d'adaptation.

Enfin, nous introduisons un module d'adaptation (Figure 5.15) dont le rôle consiste, dans le cas de

solutions trop proches, à se déplacer dans l'espace de recherche afin de mettre à jour la solution courante et de redémarrer le processus de recherche.

Afin d'ajuster une solution, le module d'adaptation utilise le concept de *structure de voisinage*. Cette dernière est à base de permutations, elle est définie par la transformation suivante :

$$V : S \longrightarrow P(S) \text{ tel que : } \forall \text{ la solution } s \in S$$

$$V(s) = \left\{ s'/s' \text{ est une solution associée à une permutation de } s \right\} \tag{5.10}$$

La figure 5.16 est un exemple de deux solutions s (couleur bleue) et s' (couleur rose) trop proches. Dans ce cas, le module d'adaptation a le rôle d'étendre l'espace de recherche en appliquant des permutations sur la solution s. Cette dernière peut être remplacée par l'une de ses solutions voisines.

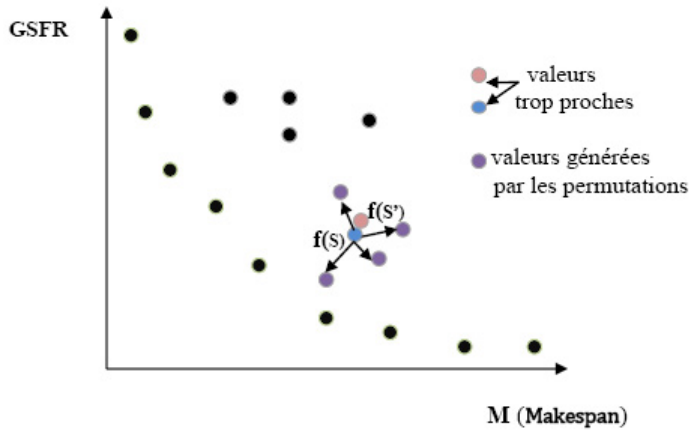

FIGURE 5.16: Génération de solutions voisines.

Les heuristiques basées listes qui implémentent la solution proposée sont ARBB (Adaptive Reliability-Based Bi-objective scheduling heuristic) et AMBB (Adaptive Makespan-Based Bi-objective scheduling heuristic). Les heuristiques coopèrent et se réfèrent, si nécessaire, à une procédure appelée 'Procédure d'adaptation'.

5.4.2 Algorithme ARBB (Adaptive Reliability-Based Bi-objective scheduling heuristic)

L'heuristique ARBB est similaire à l'heuristique RBB (approche hiérarchique). Cependant, durant son exécution, l'heuristique ARBB peut se référer au module d'adaptation pour un ajustement de la solution courante. Une fois l'ajustement réalisé, la nouvelle solution est envoyée à l'heuristique AMBB (cf.sous-section 5.4.6) pour traitement.

L'algorithme ARBB

Entrées : ALG, ARC, Λ_c

Sorties : L_{ARBB}, Λ_{ARBB}

Début

Initialiser les listes des tâches concurrentes et des tâches ordonnancées :

$T_{conc}^{(0)} := \{t \in T \mid pred(t) = \emptyset\}$;

$T_{sched}^{(0)} := \emptyset$;

Tant que $T_{conc}^{(n)} \neq \emptyset$ **Faire**

① Calculer la pression d'ordonnancement pour chaque tâche t_i de $T_{conc}^{(n)}$ sur chaque processeur p_j tel que $\Lambda_i < \Lambda_c$;

② Ordonnancer le couple (t_i, p_j) qui minimise la pression d'ordonnancement sûre ;

③ Mettre à jour les listes des tâches concurrentes et des tâches ordonnancées :

$T_{sched}^{(n)} := T_{sched}^{(n-1)} \cup \{t\}$;

$T_{conc}^{(n+1)} := T_{conc}^{(n)} - \{t\} \cup \{t' \in succ(t) \mid pred(t') \subseteq T_{sched}^{(n)}\}$;

Fin Tant que

Si la solution produite est trop proche de celles déjà sélectionnées **Alors**

Appeler la procédure Adaptation ;

Fin Si

Fin

5.4.3 Fonctionnement de ARBB

Les entrées de l'heuristique ARBB sont similaires à celles de l'heuristique ARB de l'approche hiérarchique à savoir une application ALG, une architecture cible ARC et enfin une valeur de GSFR représentant une contrainte qui limite l'espace des objectifs . De même, les sorties sont deux valeurs respectives du GSFR et de la longueur d'ordonnancement.

L'heuristique ARBB suit également les mêmes étapes adoptées par ARB avec, éventuellement, un appel à la procédure d'adaptation dans le but de relancer la recherche.

5.4.4 Procédure d'adaptation

L'adaptation de solutions permet de considérer de nouveaux espaces de recherche définis par différentes structures de voisinage, ce qui permet d'éviter une convergence rapide des solutions. Il s'agit donc de remplacer la solution courante par l'une de ses voisines. Pour ce faire, nous utilisons la notion de permutation afin de générer de nouvelles solutions.

Procédure Adaptation

Input : ordonnancement courant s, ordonnancements déjà sélectionnés, structure de voisinage

Output : Nouvel ordonnancement s_N

Début

Générer l'ensemble des ordonnancements en appliquant les permutations sur s ;

Sélectionner le sous-ensemble P_S d'ordonnancements satisfaisant les contraintes temps réel ;

Si $\exists$ un ordonnancement $s_k \in P_S$ tel que :

(i) s_k n'est pas déjà sélectionné

et (ii) sa valeur de compromis n'est pas trop proche de celles des ordonnancements déjà sélectionnés

et (iii) parmi les ordonnancements $\in P_S$, $f(s_k)$ est celui non-dominé

Alors

s_k est considéré comme le nouvel ordonnancement courant s_N ;

Sinon

s est maintenu comme ordonnancement courant ;

Fin Si

Fin

L'application de permutations sur s permet de générer des solutions voisines de s parmi lesquelles celle satisfaisant un ensemble de conditions remplacera la solution courante.

5.4.5 Fonctionnement de la procédure d'adaptation

Disposant d'une solution courante, des solutions sélectionnées et d'une structure de voisinage basée permutation, la procédure d'adaptation a pour but de créer des solutions voisines de la solution courante. Cependant, seules les solutions satisfaisant les contraintes temps réel sont prises en compte. Un ordonnancement (solution) s est remplacé par l'un de ses voisins ssi (i) ce dernier n'a pas déjà été sélectionné et (ii) sa valeur de compromis n'est pas trop proche de celles des ordonnancements déjà sélectionnés et (iii) parmi les voisins de s, c'est lui qui correspond à la valeur non-dominée dans l'espace des objectifs.

Aussitôt qu'un ordonnancement satisfaisant les trois conditions est trouvé, il est alors considéré comme le nouvel ordonnancement à partir duquel se poursuivra le processus de recherche. Il est à noter qu'un tel ordonnancement peut ne pas être trouvé auquel cas, l'ordonnancement courant est maintenu.

5.4.6 Algorithme AMBB (Adaptive Makespan-Based Bi-objective scheduling heuristic)

Pour l'heuristique AMBB, l'espace de recherche est limité par la valeur de la longueur d'ordonnancement L_{ARBB} produite par l'heuristique ARBB et considérée comme une contrainte :

L'algorithme AMBB

Entrées : ALG, ARC, L_{ARBB}

Sorties : L_{AMBB}, Λ_{AMBB}

Début

Initialiser les listes des tâches concurrentes et des tâches ordonnancées :

$T_{conc}^{(0)} := \{t \in T \mid pred(t) = \emptyset\}$;

$T_{sched}^{(0)} := \emptyset$;

Tant que $T_{conc}^{(n)} \neq \emptyset$ **Faire**

① calculer le GSFR pour chaque tâche t_i de $T_{conc}^{(n)}$ sur chaque processeur p_j tel que $L_i < L_{ARBB}$;

② Ordonnancer le couple (t_i, p_j) qui minimise le GSFR ;

③ mettre à jour les listes des tâches concurrentes et des tâches ordonnancées :

$T_{sched}^{(n)} := T_{sched}^{(n-1)} \cup \{o\}$;

$T_{conc}^{(n+1)} := T_{conc}^{(n)} - \{t\} \cup \{t' \in succ(t) \mid pred(t') \subseteq T_{sched}^{(n)}\}$;

Fin tant que

Si la solution produite est trop proche de celles déjà sélectionnées **Alors**

appeler la procédure Adaptation ;

Fin Si

Fin

5.4.7 Fonctionnement de AMBB

L'heuristique AMBB fonctionne d'une manière similaire à l'heuristique AMB de l'approche hiérarchique. Comme ARBB, AMBB peur avoir recours au module d'adaptation s'il s'avère que l'ajustement d'une solution est nécessaire.

L'exécution de ARBB et AMBB peut être répétée autant de fois que le décideur le désire. En plus, les contraintes de départ peuvent être modifiées pour créer une nouvelle instance du processus. En variant la contrainte de départ et en utilisant le module d'adaptation, un ensemble de solutions est produit et celles qui correspondent aux besoins courants sont sélectionnées.

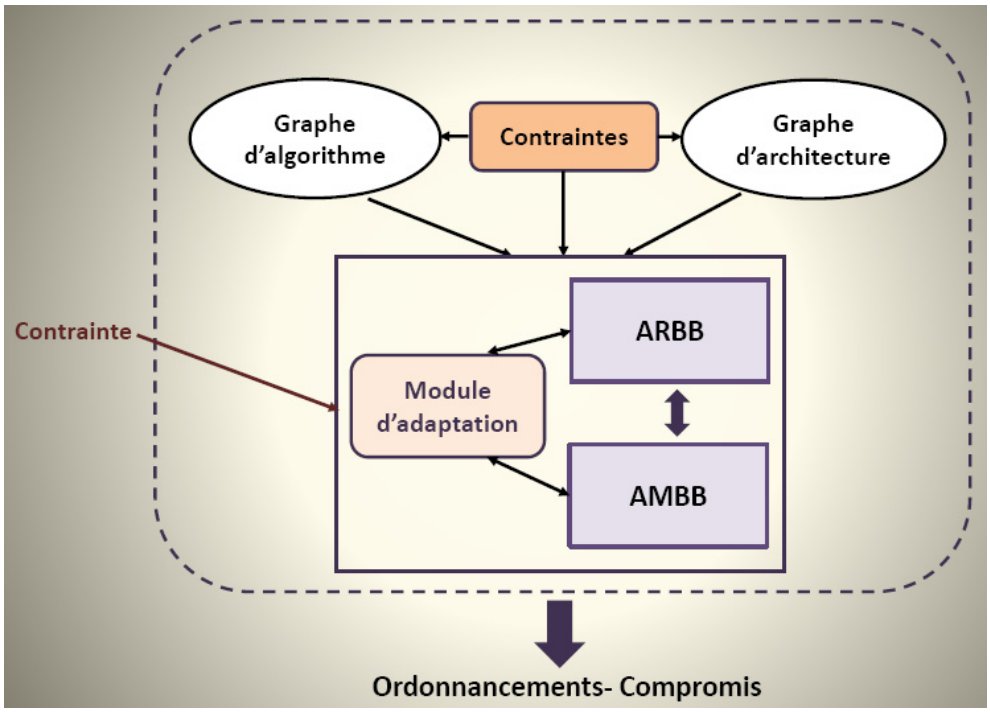

FIGURE 5.17: Flot partiel d'implantation.

L'approche adaptative permet également de définir un flot partiel d'implantation (Figure 5.17) où les solutions produites permettent d'offrir différents compromis entre la fiabilité et la longueur d'ordonnancement.

5.4.8 Flot d'implantation

Que ce soit pour l'approche hiérarchique ou l'approche adaptative que nous avons proposées, leur intégration au sein de la méthodologie AAA permet d'apporter au flot d'implantation des améliorations au niveau de l'étape d'adéquation. En effet, lors de l'étape d'adéquation, plusieurs solutions de compromis sont produites. Ceci permet d'offrir un choix de solutions qui , dans notre cas, est un ensemble d'ordonnancements de tâches sur des processeurs. Chacun des ordonnancements représente un compromis entre les objectifs traités.

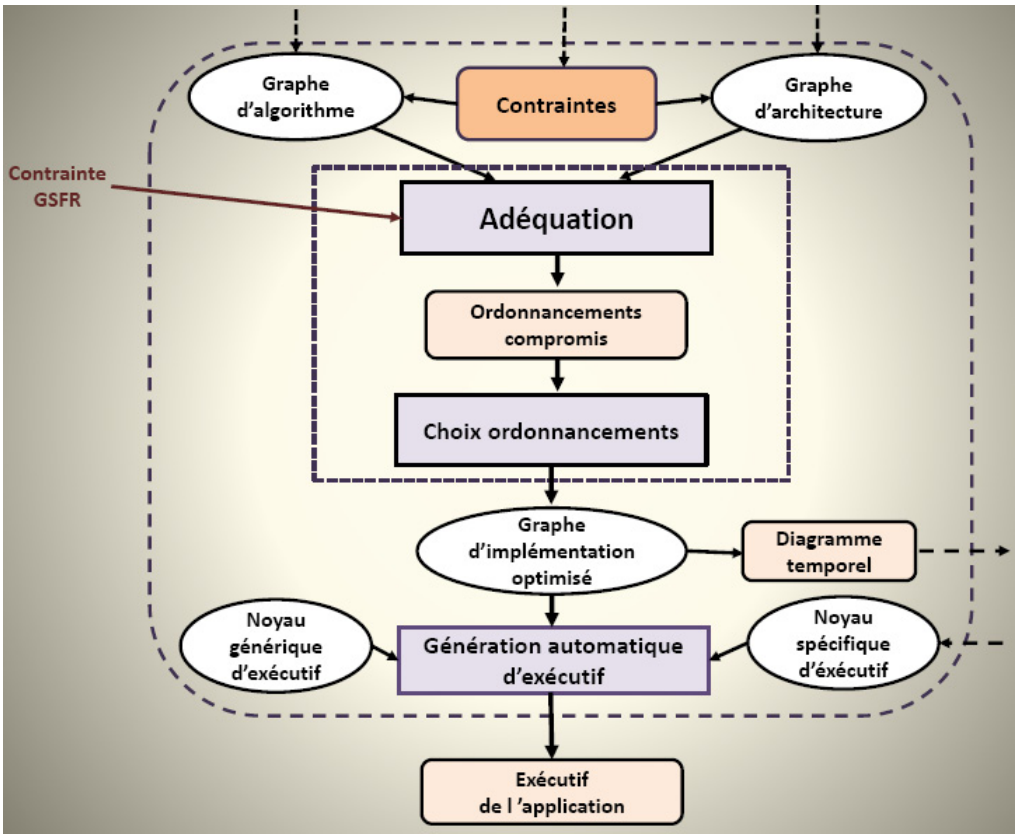

FIGURE 5.18: Flot d'implantation.

En intégrant l'un ou l'autre des flots partiels au processus global, nous obtenons le flot d'implantation (Figure 5.18) avec une étape de choix permettant une flexibilité dans les étapes ultérieures à commencer par la génération du graphe d'implémentation optimisé jusqu'à celle de l'exécutif de l'application.

5.4.9 Expérimentation de l'approche AAA-Bi-Adaptative

De même, nous avons expérimenté l'approche adaptative en l'implémentant dans l'environnement SynDEx. Les figures 5.19 et 5.20 montrent l'impact du nombre de tâches N sur le Makespan et la fiabilité respectivement.

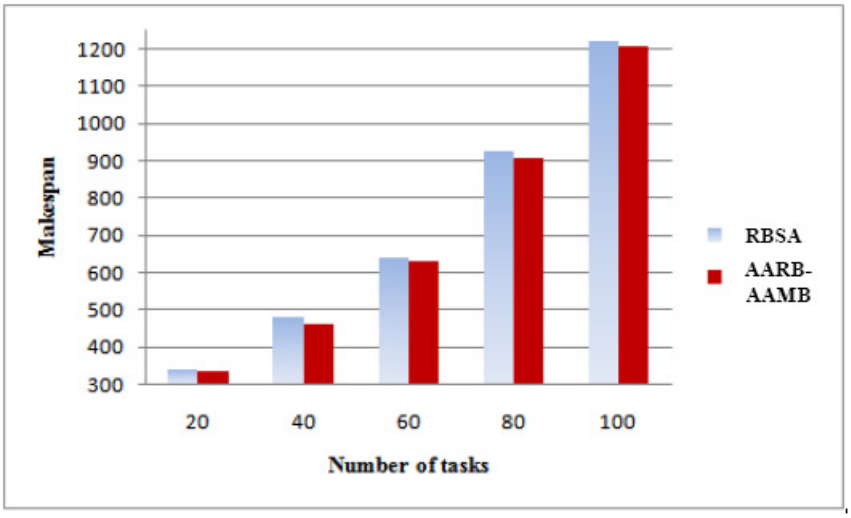

FIGURE 5.19: Impact de N sur le Makespan pour CCR=1 et P=4.

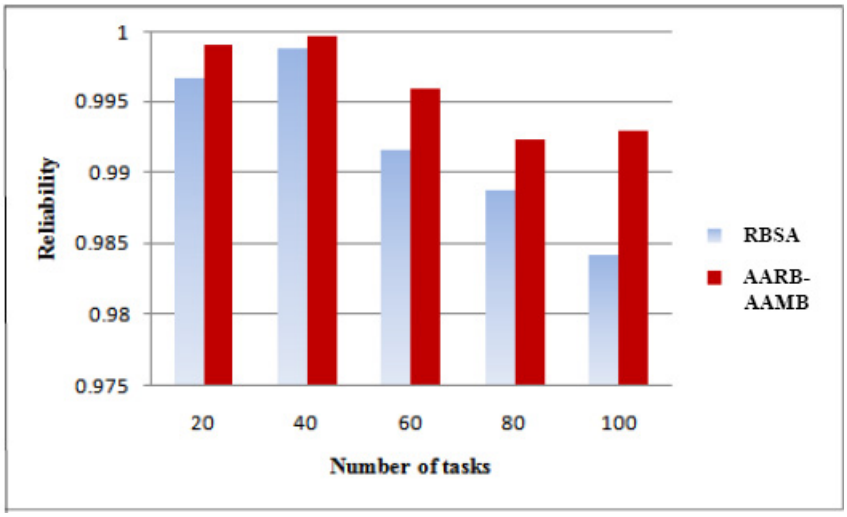

FIGURE 5.20: Impact de N sur la fiabilité pour CCR=1 et P=4.

Nous pouvons voir que les valeurs de Makespan produites par notre approche sont plus petites que celles produites par RBSA. Quant à la fiabilité, elle est également meilleure que celle générée par RBSA.

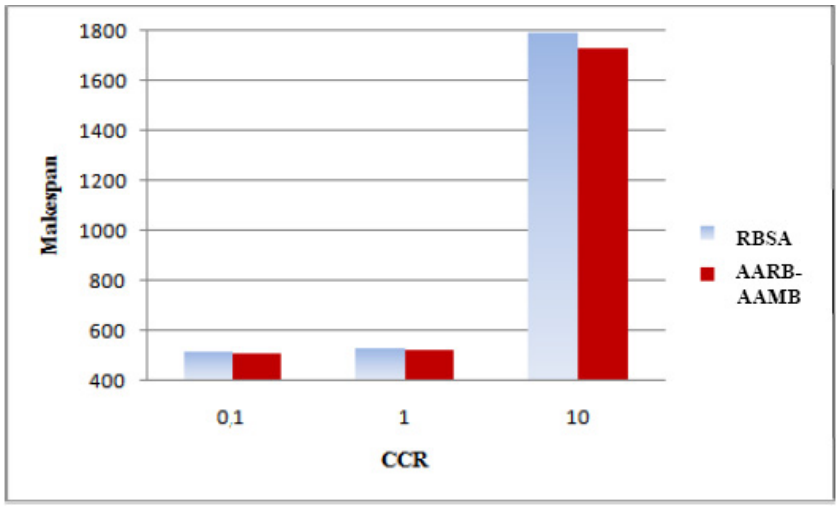

FIGURE 5.21: Impact du CCR sur le Makespan pour N=50 et P=4.

En ce qui concerne l'impact du CCR sur le Makespan et la fiabilité illustré par les figures 5.21 et 5.22, nous constatons qu'autant pour le premier objectif que pour le second, notre approche fournit de meilleurs résultats, autrement dit, un Makespan moindre et une plus grande fiabilité.

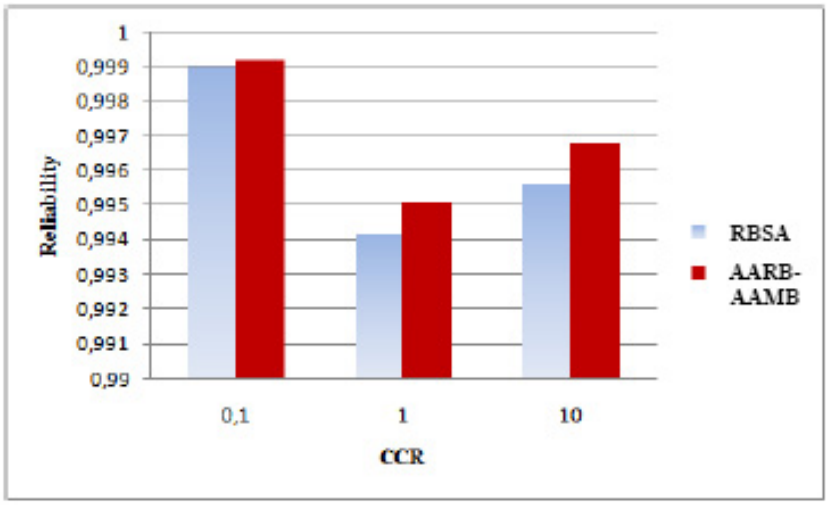

FIGURE 5.22: Impact du CCR sur la fiabilité pour N=50 et P=4.

Enfin, les figures 5.23 et 5.24 décrivent respectivement les variations du Makespan et de la fiabilité en fonction du nombre de processeurs. Les résultats sont également concluants en ce sens que l'approche adaptative AARB-AAMB proposée fournit des compromis meilleurs que ceux de RBSA sur tous les ensembles de données.

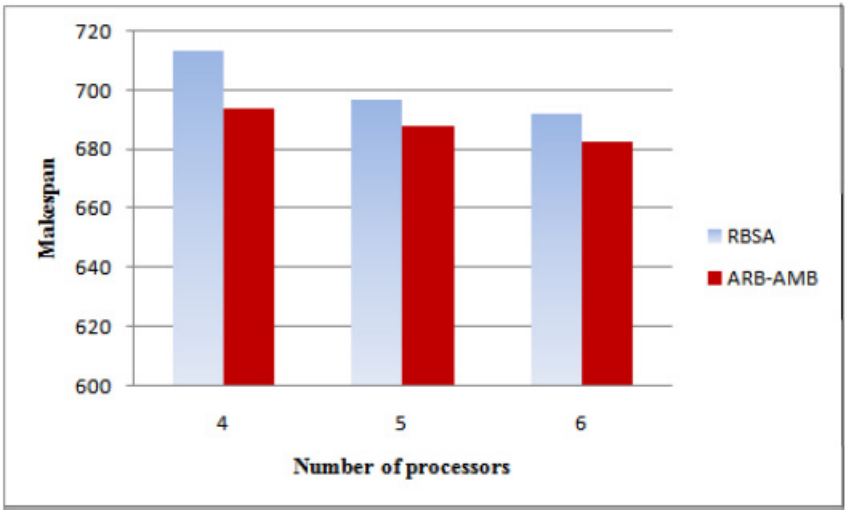

FIGURE 5.23: Impact de P sur le Makespan pour N=50 et CCR=1.

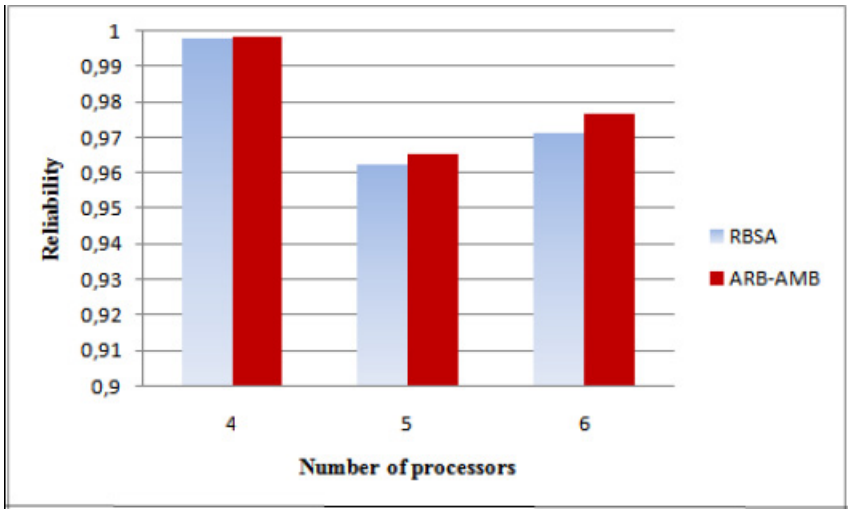

FIGURE 5.24: Impact de P sur la fiabilité pour N=50 et CCR=1.

heuristiques / Exécution	RBSA	RBB-MBB	ARBB-AMBB
Exécution1	M=644.72 R=0.998297	M=616.51 R=0.998539	M=616.51 R=0.998539
Exécution2	M=644.72 R=0.998297	M=608.76 R=0.999029	M=608.76 R=0.999029
Exécution3	M=644.72 R=0.998297	M=608.76 R=0.999029	M=599.70 R=0.999312
Exécution4	M=644.72 R=0.998297	M=608.76 R=0.999029	M=599.70 R=0.999312

TABLE 5.5: Résultats du Makespan et de la fiabilité pour N=60.

Le tableau 5.5 montre les résultats de la longueur d'ordonnancement (Makespan) (M) et de la fiabilité (R) produits par l'exécution des heuristiques avec une application composée de 60 tâches. Nous remarquons que les résultats fournis par RBSA sont les mêmes dans les trois exécutions. Ceci est dû aux données d'entrée qui sont identiques dans les trois itérations. En revanche, pour RBB-MBB et ARBB-AMBB, il est noté une variation de résultats, ce qui est expliqué par le fait que les données d'entrées varient selon l'heuristique courante.

La stabilisation des résultats a lieu après un certain nombre d'itérations. Il est observé que dans les exécutions 3 et 4, les trois heuristiques se sont stabilisées. Enfin, grâce au module d'adaptation qui permet une meilleure exploration de l'espace de recherche, l'approche AARB-AAMB assure une meilleure diversité des solutions.

5.5 Conclusion

Le point de départ de notre travail est la méthodologie AAA. Nous nous sommes focalisés sur une étape pertinente du processus à savoir l'étape d'ordonnancement. Plus particulièrement, nous avons considéré l'ordonnancement bi-objectifs en prenant en compte la longueur d'ordonnancement et la fiabilité. Nous avons ainsi proposé deux approches d'ordonnancement bi-objectifs. La première approche est hiérarchique, elle est supportée par deux heuristiques optimisant alternativement les deux objectifs. Dans la seconde approche, un module d'adaptation y a été introduit, il permet une extension de l'espace de recherche et l'ajustement des solutions, ce qui assure une diversification des solutions. Les intégrations respectives des deux approches proposées au sein de la méthodologie AAA permet dans le premier cas comme dans le second de définir un processus caractérisé par une étape d'adéquation engendrant un ensemble de solutions parmi lesquelles les décideurs pourront choisir celles qui leur conviennent. Les deux approches proposées ont été validées par des expérimentations et comparées à l'heuristique *Reliable Bi-criteria Scheduling Algorithm* (RBSA) développée au sein de l'INRIA. Les résultats montrent clairement que les solutions produites par les deux approches proposées sont meilleures que celles produites par RBSA. En effet, l'approche hiérarchique permet de produire des solutions faisant de bons compromis entre les deux objectifs avec, toutefois, un manque

de diversité des solutions. Cependant, l'approche adaptative fournit, en plus de la qualité des solutions, une meilleure diversification et ce grâce au module d'adaptation.

Conclusion générale et perspectives

Dans cette thèse, nous nous sommes intéressés aux systèmes embarqués temps réel et plus particulièrement à la problématique de leur développement. Nous avons, en effet, réalisé une synthèse concernant les méthodologies de développement en la concluant par une classification prenant en compte un certain nombre de critères.

Par ailleurs, la méthodologie 'Adéquation Algorithme Architecture' (AAA) a constitué la base de notre travail. Nous nous sommes, cependant, focalisés sur une étape pertinente du processus de développement à savoir l'étape d'ordonnancement. Celle-ci consiste à affecter les tâches d'une application aux différents processeurs constituant la plateforme d'exécution tout en respectant les contraintes temps réel imposées.

Cependant, vu que l'ordonnancement est en réalité un problème multi-objectifs, nous devions le traiter comme tel. Il est, en effet, courant que la tâche d'ordonnancement soit conditionnée par un certain nombre d'objectifs qui sont, en plus, conflictuels. La résolution d'un tel problème permet la production de plusieurs solutions dites de compromis.

Ayant pris en considération l'ordonnancement bi-objectifs et sachant qu'une telle problématique est un champs d'investigation de l'optimisation multi-objectifs, nous avons effectué un rappel de cette dernière en insistant particulièrement sur les méthodes de résolution notamment sur celles dites approchées.

Les objectifs traités, dans notre travail, sont la longueur d'ordonnancement et la fiabilité. En ce qui concerne cette dernière, nous avons adopté le modèle selon lequel le système est considéré comme étant une seule tâche ou opération exécutée sur une machine [Girault and Kalla, 2009] et ce pour éviter que le taux de panne ne dépende de la durée de la tâche.

Nous avons proposé deux approches d'ordonnancement bi-objectifs. La première approche est hiérarchique, elle consiste à optimiser alternativement les deux objectifs. En ce qui concerne la seconde approche, nous y avons introduit un module d'adaptation dont le rôle est, lorsque cela est nécessaire, d'étendre l'espace de recherche et d'ajuster la solution courante. L'adaptation est basée-permutation, ce qui signifie que la structure de voisinage d'une solution est créée en opérant des permutations sur les éléments qui constituent la solution.

Les intégrations respectives des approches d'ordonnancement proposées au sein de la méthodologie AAA ont permis d'en dégager deux versions : *AAA-Bi-Hiérarchique* et *AAA-Bi-Adaptative*. Que ce soit pour le premier cas ou le second, l'ordonnancement bi-objectifs engendre un flot d'implantation offrant une certaine flexibilité quant au choix des ordonnancements. Toutefois, l'approche adaptative assure une meilleure diversification des solutions produites.

Des simulations ont été réalisées pour les deux approches d'ordonnancement. L'environnement utilisé est l'outil SynDEx, un outil CAO pour l'optimisation et l'implémentation de systèmes embarqués temps réel (http ://www.syndex.org).

Des comparaisons ont été réalisées avec l'heuristique RBSA (Reliable Bi-Criteria Scheduling Algorithm) proposée au sein de l'INRIA. Il a été constaté que les heuristiques que nous avons proposées, autant l'approche hiérarchique que l'approche adaptative, sont plus performantes. En plus, nous remarquons que dans l'approche adaptative, il y a une meilleure exploration de l'espace de recherche. Cependant, bien que l'espace de recherche soit mieux exploité, la faculté d'adaptation engendre un temps supplémentaire d'exécution.

Le travail présenté dans ce manuscrit laisse des voies ouvertes pour des améliorations futures. Plus

particulièrement, les approches hybrides d'ordonnancement assurent de bons résultats. La combinaison concerne des approches globale/locale ou encore la coopération entre métaheuristiques. Les approches évolutionnaires fournissent également de bons compromis pour les problèmes multi-objectifs en général. Aussi, leur exploitation est-elle un axe de recherche pertinent.

Dans notre travail, nous avons traité l'ordonnancement statique. Celui-ci suppose un certain nombre de paramètres comme étant fixes. Nous nous intéressons, par ailleurs, à l'ordonnancement dynamique où l'imprévisibilité de l'environnement constitue un paramètre important.

La réflexion autour d'approches d'optimisation multi-objectifs peut concerner n'importe quel domaine où le problème prend en compte deux ou plusieurs objectifs conflictuels. Dans ce cas, le paramétrage se fera relativement au domaine traité.

Un autre axe de recherche concerne les modèles adoptés comme données d'entrée au problème d'ordonnancement. Dans la plupart des approches, le modèle utilisé est un graphe de dépendance de données. Nous pensons que la réflexion autour de formalismes éventuellement plus adéquats est un axe de recherche intéressant à explorer.

Par ailleurs, la conception par composants est une approche qui répond aux exigences des systèmes embarqués temps réel. Notons, cependant, que bien que ce domaine de recherche soit relativement actif, autant la question autour de la granularité des composants que l'approche à adopter quant à leur assemblage constituent au jour d'aujourd'hui de grandes voies d'exploration.

Bibliographie

[MeM, 2005] (2005). Memvatex (méthode de modélisation pour la validation et la traçabilité des exigences). www.memvatex.org.

[Alkhodre, 2004] Alkhodre, A. B. (2004). *Développement formel de systèmes temps réel à l'aide de SDL et IF (compilation pour système temps réel)*. PhD thesis, Institut National des Sciences appliquées de Lyon.

[Andre and Peraldi, 1992] Andre, C. and Peraldi, D. M.-A. (1992). Hard real-time system implementation on a microcontroller. In *WRTP'92, in IBRA/BIRA International Workshop on real-time programming*.

[Assayad et al., 2004] Assayad, I., Girault, A., and Kalla, H. (2004). A bi-criteria scheduling heuristic for distiributed embedded systems under reliability and real -time. In *Conférence internationale sur des systèmes fiables et Réseaux, DSN' 04*.

[Assayad et al., 2012] Assayad, I., Girault, A., and Kalla, H. (2012). Scheduling of real embedded systems under reliability and power constraints. In *International Conference on Complex Systems, ICCS'12*.

[Awad et al., 1996] Awad, M., Kuusela, J., and Ziegler, J. (1996). *Object-oriented technology for real time systems : a practical approach using OMT and Fusion*. Prentice-Hall.

[Babau and Alkhodre, 2001] Babau, J. B. and Alkhodre, A. (2001). A development method for prototyping embedded systems by using uml and sdl (proseus). In *SIVOEES*.

[Barichard and Hao, 2003] Barichard, V. and Hao, J. K. (2003). Genetic tabu search for the multi-objective knapsack problem. *Tsinghua Science and Technology*, 8 :8–13.

[Bause and Buchholdz, 1990] Bause, F. and Buchholdz, P. (1990). Protocol analysis using a timed version of sdl. In *FORTE*.

[Belarbi, 2003] Belarbi, M. (2003). *Validation Temporelle des Applications MultitâchesTtemps Réel basée sur les Automates Temporisés Communicants*. PhD thesis, Institut National des Sciences Appliquées de Lyon.

[Bellman, 1986] Bellman, R. E. (1986). *The Bellman continum*. Robert S. Roth, Philadelphia (USA).

[Bendib et al., 2013] Bendib, S.-S., Kalla, H., Kalla, S., and Arar, C. (2013). A two-step bicriteria scheduling approach for distributed real time systems. In *International Conference on Electronics, Computer and Computation (ICECCO)*.

[Berry et al., 1987] Berry, G., Canavé, P., and Gauthier, G. (1987). Programmation synchrone des systèmes réactifs : le langage esterel. *Technique et Science Informatique*, 6 :305–316.

[Boeres et al., 2011] Boeres, C., Sardina, I. M., and Drummond, L. M. A. (2011). An efficient weighted bi-objective scheduling algorithm for heterogeneous systems. *Parallel Computing*, 37 :349–364.

[Bolognesi and Brinksma, 1989] Bolognesi, T. and Brinksma, E. (1989). *Introduction to the ISO specification language LOTOS*. van Eijket al.

[Bozga et al., 1999] Bozga, M., Fernandez, J.-C., Ghirvu, L., Krimm, J. P., Mouniera, L., and Sifakis, J. (1999). If : An intermediate representation of sdl and its application. In *Proc of the Ninth SDL Forum*.

[Bézivin, 2005] Bézivin, J. (2005). On the unification power of the models. *Software and System modeling*, 4 :171–188.

[Canon, 2010] Canon, L.-C. (2010). *Outils et algorithmes pour gérer l'incertitude lors de l'ordonnancement d'applications sur plateformes distribuées*. PhD thesis, Ecole doctorale IAEM Lorraine, Université Henri Poincaré.

[Canon et al., 2011] Canon, L. C., Essafi, A., Mounié, G., and Trystram, D. (2011). A bi-objective scheduling algorithm for desktop grids. In *European 2011*.

[Cardeira, 1994] Cardeira, C., M. Z. (1994). Ordonnancement des tâches dans les systèmes temps réel et répartis-algorihmes et critères de classification. *APII*, 28 :353–384.

[Carlson, 2002] Carlson, J. (2002). Languages and methods for specifying real-time systems. Technical report, Department of Computer Science and Engineering, Malardalen University.

[Cherif, 2014] Cherif, S. (2014). *Approche basée sur les modèles pour la conception des systèmes dynamiquement reconfigurables : de MARTE vers RecoMARTE*. PhD thesis, Université des Sciences et Technologies de Lille.

[Chevochot and Puaut, 1999a] Chevochot, P. and Puaut, I. (1999a). An approach for fault-tolerance in hard-time distributed systems. In *18th IEEE Symposium on Reliable Distributed Systems (SRDS'99)*.

[Chevochot and Puaut, 1999b] Chevochot, P. and Puaut, I. (1999b). Tolérance aux fautes dans les systèmes répartis temps réel strict. *Techniques et Sciences Informatiques (TSI)*, 18(8) :837–870.

[CNRS, 1988] CNRS (1988). Le temps réel. *TSI*, 7 :493–500.

[Collette and Siarry, 2002] Collette, Y. and Siarry, P. (2002). *Optimisation Multiobjectif*. Editions Eyrolles.

[Cormick et al., 2011] Cormick, J., Singhoff, F., and Hugues, J. (2011). Building parallel, embedded, and real-time applications with ada. Technical report, Cambridge University Press.

[Courtiat et al., 2000] Courtiat, J. P., Santos, C. A. S., Lohr, C., and Outtaj, B. (2000). Experience with rt-lotos, a temporal extension of the lotos formal description technique. *Computer Communications*, 23, No. 12 :1104–1123.

[Darwin, 1876] Darwin, C. (1876). *L'origine des espèces au moyen de la sélection naturelle ou la lutte pour l'existence dans la nature*. Reinwald, Paris.

[Dasgupta et al., 2008] Dasgupta, D., Hernandez, G., Garrett, D., and Vejandla, P. K. (2008). A comparaison of multiobjective evolutionary algorithms with informed initialization and kuhn-munkres algoritm for the sailor assignment problem. In *Genetic and Evolutionary Computation Conference (GECCO)*.

[Davare et al., 2007] Davare, A., Densmore, D., Meyerowitz, T., Pinto, A., Sangiovanni-Vincentelli, A., Hang, G., Zeng, H., and Zhu, D. (2007). A next-generation design framework for platform-based design. In *Design and Verification conference*.

[Deb et al., 2000] Deb, K., Agrawal, S., Pratab, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorith for multi-objective optimization :nsgaii. Technical report, Indian Institute of Technology.

[Deb and Jain, 2002] Deb, K. and Jain, S. (2002). Running performance metrics for evolutionary multi-objective opti. Technical report, Department of Mechanical Engineering. IIT Campur.

[Demarco, 1979] Demarco, T. (1979). *Structured Analysis and System Speci*. Englewood Cliffs.

[Dennis, 1975] Dennis, J. (1975). First version of a dataflow procedure language. *Springer-Verlag, Lecture Notes in Computer Science*, 19 :362–376.

[Dertouzos and Mok, 1989] Dertouzos, M. L. and Mok, A. k. (1989). Multiprocessor online scheduling of hard- real-time tasks. *IEEE Transactions on Software Engineering*, 15(12) :1497–1506.

[Dhaenens-Flipot, 2005] Dhaenens-Flipot, C. (2005). *Optimisation Combinatoire Multiobjective : Apport des Méthodes Coopératives et Contribution à l'Extraction de Connaissances*. PhD thesis, Université des Sciences et Technologies de Lille.

[Dhingra and Chandna, 2010] Dhingra, A. and Chandna, P. (2010). Bi-criteria m-machine sdst flow scheduling using modified heuristic genetic algorithm. *International Journal of Engineering Science and Technology*, 2 :216–225.

[Diefenbruch, 1997] Diefenbruch, M. (1997). Queuing sdl- a language for the functionnal and quantitative specification of distributed systems. Technical report, University Gesamthochschule Essen. Fachbereich Mathematik und Informatik.

[do Nascimento Francisco Assis et al., 2007] do Nascimento Francisco Assis, M., Oliveira, M. F. S., and wagner, F. R. (2007). Modes : Embedded systems design methodology and tools based on mde. In *Model-Based Methodologies for Pervasive and Embedded Software*.

[Dogan and Ozguner, 2000] Dogan, A. and Ozguner, F. (2000). Reliable matching and scheduling of precedence-constrained tasks in heterogeneous distibuted computing. In *The2000 International Workshops on Parallel processing (ICPP00)*.

[Dogan and Ozguner, 2002] Dogan, A. and Ozguner, F. (2002). Matching and scheduling algorithms for mimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Trans. Parallel and Distributed Systems*, 13 :308–323.

[Dogan and Ozguner, 2005] Dogan, A. and Ozguner, F. (2005). Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. *Comput.*, 48 :300–314.

[Dongarra et al., 2007] Dongarra, J., Jeannot, E., Saule, E., and Shi, Z. (2007). Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *SPAA'07*.

[Dorin, 2010] Dorin, F. (2010). *Contributions à l'ordonnancement et l'analyse des systèmes temps réel critiques*. PhD thesis, Ecole Nationale Supérieure de Mécanique et d'Aérotechnique, Poitiers, France.

[Dumitrescu et al., 2001] Dumitrescu, D., Grosan, C., and Oltean, M. (2001). A new evolutionary adaptive representation paradigm. *Studia Univ. Babes Bolyal, Informatica*, 2 :19–28.

[Dutot and Trystram, 2005] Dutot, P. F. and Trystram, D. (2005). A best-compromise bicriteria scheduling algorithm for parallel tasks. In *Workshop on Efficient Algorithms*.

[Edgeworth, 1881] Edgeworth, F. Y. (1881). *Mathematical Physics*. P. Keagan, London.

[Esquirol and Lopez, 1999] Esquirol, P. and Lopez, P. (1999). *L'ordonnancement*. Economica.

[Feiler et al., 2006] Feiler, P. H., Glush, D. P., and hudak J. J (2006). The architecture analysis and design language (aadl) : An introduction. Technical report, CMU/SEI.

[Fonseca and Flemming, 1993] Fonseca, C. M. and Flemming, P. J. (1993). Genetic algorithms for multiobjectives optimization : Formulation, discussion and generalization. In *Ffth International Conference on Genetic Algorithms*.

[frank, 2005] frank, A. (2005). On kuhn's hungarian method. *Naval Research Lo*, 52 :2–6.

[Friesz et al., 1993] Friesz, T., Anandalingam, G., Mehta, N., Nam, K., Shah, S., and Tobin, R. (1993). The multiobjective equilibrium network design problem revisited : A simulated annealing approach. *European Journal of Operational Research*, 65 :44–57.

[Funk et al., 2001] Funk, S., Goossens, J., and Baruah, S. (2001). On-line scheduling on uniform multiprocessors. In *22nd IEEE Real Time Systems Symposium (RTSS'01)*.

[Galand, 2008] Galand, L. (2008). *Méthodes exactes pour l'optimisation multicritère dans les graphes : recherches de solutions de compromis*. PhD thesis, Université Paris 6.

[Gamatié et al., 2008] Gamatié, A., Beux, S. L., Piel, E., Etien, A., Atitallah, R. B., Marquet, P., and Dekeyser, J. L. (2008). A model driven design framework for high performance embedded systems. Technical report, http ://hal.inria.fr/inria-00311115/en.

[Gandibleux et al., 1994] Gandibleux, X., Libert, G., Cartignies, E., and Millot, P. (1994). Smart : étude de la faisabilité d'un solveur de problèmes de mobilisation de réserve tertiaire. *Systèmes de Décision*, 3 :45–67.

[Garg and Singh, 2011] Garg, R. and Singh, A. K. (2011). Multi-objective optimization to workflow grid scheduling using reference point based evolutionary algorithm,. *Internanional Journal of Computer applications*, 22 :0975–8887.

[Gary and Johnson, 1979] Gary, M. R. and Johnson, D. S. (1979). *Computers and Intractability : A guide to the theory of NP-completeness*. W. F. Freeman and Co.

[Gaudel et al., 1988] Gaudel, M. C., Marre, B., Schlienger, F., and Bernot, G. (1988). *Précis de génie logiciel*. Masson.

[Girault and Kalla, 2009] Girault, A. and Kalla, H. (2009). A novel bicriteria scheduling heuristic providing a guaranteed global system rate. *IEEE Transactions in Dependable and Secure Computing*, 6 :241–254.

[Girault et al., 2009] Girault, A., Saule, E., and Trystram, D. (2009). Reliability versus performance for critical applications. *Parallel and Distributed Computing*, 69 :326–336.

[Glover, 1989] Glover, F. (1989). Tabu search, part i. *Computing*, 1 :190–206.

[Glover, 1995] Glover, F. (1995). Tabu thresholding : Improved search by nonmonotonic trajectoires. *Computing*, 7 :426–442.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.

[Grandpierre et al., 1999] Grandpierre, T., Lavarenne, C., and Sorel, Y. (1999). Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In *international workshop on Hardware/software codesign (CODES)*.

[Gérard et al., 2002] Gérard, S., Terrier, F., and Tanguy, Y. (2002). Using the model paradigm for real-time systems development : Accord/uml. In *Advances in object-Oriented information Systems*.

[Grolleau, 1999] Grolleau, E. (1999). *Ordonnancement temps réel hors ligne à l'aide de réseaux de Petri en environnement monoprocesseur et multiprocesseur*. PhD thesis, LISI-ENSMA.

[Grosan and Dumitrescu, 2002] Grosan, C. and Dumitrescu, D. A. (2002). A comparaison of multiobjective evolutionary algorithms. *Acta Matematica*, 4.

[Gumzej et al., 2006] Gumzej, R., Colranic, M., and Halang, W. A. (2006). Safe and timely scenario switching in uml real-tme projects. In *International Symposium on Object and Component-Oriented Real-Time Distributed Computing*.

[Gumzej et al., 2009] Gumzej, R., Colranic, M., and Halang, W. A. (2009). A reconfiguration pattern for distributed embedded systems. *Software and System modeling*, 8(1) :145–161.

[Habet, 2004] Habet, D. (2004). *Coopération entre recherches locale et exhaustive pour l'optimisation combinatoire*. PhD thesis, Université de Picardie Jules Verne.

[Hakem and Butelle, 2006] Hakem, M. and Butelle, F. (2006). A bi-objective algorithm for scheduling parallel applications on heterogeneous systems subject to failures. In *Renpar' 17*.

[Hansson et al., 2004] Hansson, I. H. C., Akerholm, M., and Torngren, M. (2004). Saveccm : a component model for safety-critical real-time systems. In *Euromicro Conference, Special Session Component Models for Dependable Systems*.

[Harel and Pnueli, 1985] Harel, D. and Pnueli, A. (1985). On the development of reactive systems. *Logics and Models of Concurrent Systems*, 13 :477–498.

[Hertz et al., 1994] Hertz, A., Jaumard, B., Ribeiro, C. C., and Formosinho Filho, W. P. (1994). A multi-criteria tabu search approach to cell formation problems in group technology with multiple objectives. *RAIRO Operations Research*, 28 :303–328.

[Hirsh, 1985] Hirsh, E. (1985). Evolutionary evolution of command and control systems. In *Program Manager*.

[Holland, 1975] Holland, J. H. (1975). Adaptation in natural and artificial systems : an introduction analysis with applications to biology. Technical report, University of Michigan.

[Hoogeveen, 2005] Hoogeveen, J. A. (2005). Multicriteria scheduling. *European Journal of Operational research*, 167 :592–623.

[Horn et al., 1994] Horn, J., Nafpliotis, N., and Goldberg, D. E. (1994). A niched pareto genetic algorithm for multiobjective optimization. *First IEEE Conference on Evolutionary Computation*, 1 :82–87.

[Hull et al., 2004] Hull, E., Jackson, K., and Dick, J. (2004). *Requirements Engineering*. Springer.

[Hyppolite et al., 2008] Hyppolite, J. L., Bloch, C., Chatonnay, P., Espanet, C., Chamagne, D., and Wimmer, G. (2008). Tuning an evolutionary algorithm with taguchi methods. In *Soft Computing as Transdisciplinary Science and Technology*.

[Islam and Suri, 2007] Islam, S. and Suri, N. (2007). A multi variable optimization approach for the design of integrated dependable real-time embedded systems. In *IFIP International Federation for Information Prosessing, LNCS*.

[ITU-T, 1996] ITU-T, I. (1996). Specification and design language (sdl). http ://www.iec.org.

[ITU-T, 1999] ITU-T, I. (1999). Sdl combined with uml. http ://www.itu.int/ITU-T/index.htm.

[Jadaan et al., 2009] Jadaan, O. A., Rajamani, L., and Rao, C. R. (2009). Non dominated ranked genetic algorithm for solving constrained multiobjective optimization problems. *Theoretical and Applied information Technology*, 5 :60–67.

[Jaszkiewicz, 2002a] Jaszkiewicz, A. (2002a). Genetic local search for multiple objective combinatorial optimization. *European Journal of Operational Research*, 137 :50–71.

[Jaszkiewicz, 2002b] Jaszkiewicz, A. (2002b). On the performance of multiple objective genetic local search on the 0/1 knapsack problem-a comparative experiment. *IEEE Transactions on Evolutionary Computation*, 6 :402–412.

[Jeannot et al., 2008] Jeannot, E., Saule, E., and Trystram, D. (2008). Bi-objective approximation scheme for makespan and reliability optimization on uniform parallel machines. In *14th International Euro-Par Conference on Parallel Processing*.

[Kalla et al., 2004] Kalla, H., Girault, A., and Sorel, Y. (2004). A scheduling heuristics for distributed real time embedded systems tolerant to processor and commnuincation media failures. *IJPR*, 42 :2877–2898.

[Kamiura et al., 2002] Kamiura, J., Hiroyasu, T., Miki, M., and Watanabe, S. (2002). Mogades : multi-objective genetic algorithm with distributed environment scheme. In *Second international Workshop on Intelligent systems design and application*.

[Kermia, 2009] Kermia, O. (2009). *Ordonnancement temps réel multiprocesseur de tâches non-préemptives avec contraintes de précédence, de périodicité stricte et de latence*. PhD thesis, Université de Paris Sud, France.

[Kim and Naghibdadeh, 1980] Kim, K. and Naghibdadeh, M. (1980). Prevention of task overruns in real-time non-preemptive multiprogramming systems. In *Perf*.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Jr, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220 :671–680.

[Knowles and Corne, 1999] Knowles, J. and Corne, D. (1999). The pareto archived evolution strategy : A new baseline algorithm for pareto multiobjective optimization. *Evolutionary Computation (CEC99)*, 1 :98–105.

[Kocik, 2000] Kocik, R. (2000). *Sur l'optimisation des systèmes distribués temps réel embarqués : application au prototype rapide d'un véhicule électrique semi-autonome*. PhD thesis, Université de Rouen.

[Krichen, 2013] Krichen, f. (2013). *Architectures logicielles à composants reconfigurables pour les systèmes temps réel répartis*. PhD thesis, Université Toulouse II.

[Kuhn, 1955] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *naval research logistics quarterly*, 2 :83–97.

[Kuhn, 2005] Kuhn, H. W. (2005). The hungarian method for the assignment problem. *Naval*, 52 :7–21.

[Kuusela and Kettunen, 1993] Kuusela, J. and Kettunen, E. (1993). Integrating sdl and object-oriented analysis through omt/sdl. In *6th SDL Forum in Darmstadt*.

[Langendoerfer and koenig, 1999] Langendoerfer, P. and koenig, H. (1999). Automated protocol implementations based on activity threads. In *Seventh Annual International Conference on Networks protocols*.

[Lavarenne and Sorel, 1997] Lavarenne, C. and Sorel, Y. (1997). Modèle unifié pour la conception conjointe logiciel-matériel. *Traitement de signal*, 6 :14.

[Le Pape, 1995] Le Pape, C. (1995). Three mechanisms for managing ressource constraints in the librairy for constraint-based scheduling. In *INRIA/IEEE Conference on Emerging Technologies and Factory Automation*.

[Lecomte, 2011] Lecomte, S. (2011). *Méthodologie de conception basée sur des modèles de haut niveau pour les systèmes de radio logicielle*. PhD thesis, Université de Rennes 1, France.

[Lee and Neuendorffer, 2007] Lee, E. A. and Neuendorffer, S. (2007). Tutorial : Buil ptolemy ii models graphically. Technical report, Electrical Engineering and Computer Science, University of California, Berkeley, USA.

[Lenoir, 1999] Lenoir, J. (1999). *Les outils de conception système du logiciel enfoui*. Electronique, 89.

[Leung and Whitehead, 1982] Leung, J. Y.-T. and Whitehead, J. (1982). On the complexity of fixed-priority scheduling of periodic, real time tasks. *Performance Evaluation*, 2(4) :237–250.

[Li et al., 2010] Li, X., Amodeo, I., Yalaoui, F., and Chehade, H. (2010). Métaheuristiques multiobjectif pour un problème d'ordonnancement de machines parallèles. In *10ème conférence internationale de MOdélisation et SIMulation- MOSIM'10*.

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

[Mammeri, 2000] Mammeri, Z. (2000). *SDL Modélisation de protocoles et systèmes réactifs*. Hermes.

[Meumeu Yomsi, 2009] Meumeu Yomsi, P. (2009). *Prise en compte du coût exact de la préemption dans l'ordonnancement temps réel monoprocesseur avec contraintes multiples*. PhD thesis, Université Paris-Sud 11.

[Mok, 1983] Mok, A. (1983). *Fundamental design problems for the hard real time environments*. PhD thesis, MIT.

[Muller and Stich, 2001] Muller, C. Z. P. and Stich, C. (2001). Components@work : Component technology for embedded systems. In *Component-based Software Engineering*.

[Muller et al., 2005] Muller, P. A., Fleurey, F., Vojtisek, D., Drey, Z., Pollet, D., Fondement, Fand Studer, P., and Jézéquel, J. M. (2005). On executable meta-languages applied to model transformations. In *Model Transformations*.

[Nagar et al., 1995] Nagar, A., Haddock, J., and Heragu, S. (1995). Multiple and bicriteria scheduling : A litterature survey. *European Journal of Operational Research*, 81 :88–104.

[Nakechbandi et al., 2006] Nakechbandi, M., Colin, J.-Y., and Gashumba, J. B. (2006). An efficient fault tolerant scheduling algorithm for precedence constrained tasks in heterogeneous distributed systems. In *Int. Joint Conf. on Computer, Information, and Systems sciences, and Engineering CIS2E 06*.

[Ober and Graf, 2003] Ober, I. and Graf, S. (2003). Validating timed uml models by simulation and verification. In *Proc. of the SVERTS'03 Workshop, San Fransisco, California*.

[Object Management Group, 2001] Object Management Group, I. (2001). Norme sysml. version 1.0.

[Object Management Group, 2002] Object Management Group, I. (2002). Uml-spt (uml for scheduling performance and time). http ://www.omg.org/.

[Object Management Group, 2003] Object Management Group, I. (2003). Mda guide version 1.0.1.

[Object Management Group, 2007a] Object Management Group, I. (2007a). Omg systems modeling language (omg sysml).

[Object Management Group, 2007b] Object Management Group, I. (2007b). Uml profile for modeling and analysis of real time embedded systems (marte).

[Object Management Group, 2011a] Object Management Group, I. (2011a). Uml profile for marte : Modeling and analysis of real-time embedded systems.

[Object Management Group, 2011b] Object Management Group, I. (2011b). Unified modeling language.

[Object Mannagement Group, 2006] Object Mannagement Group, I. (2006). Meta object facility (mof) core specification.

[Orozco et al., 1997] Orozco, J., Cayssials, R., Santos, J., and Ferro, E. (1997). Precedence constraints in hard real-time distributed systems. In *iceccs*.

[Oyetunji, 2011] Oyetunji, E. O. (2011). Assessment of the level of agreement among metrics comparing non-dominated sets. *Emerging Trends in Engineering and Applied Sciences (JETEAS)*, 2 :429–434.

[Oyetunji and Oluleye, 2010] Oyetunji, E. O. and Oluleye, A. E. (2010). Multicriteria scheduling : The challenges of comparing non-dominated sets. In *International Conference of Nigerian Institute of Industrial Engineers*.

[Pareto, 1896] Pareto, V. (1896). *Cours d'économie politique*. Rouge, Lausanne.

[Pentico, 2007] Pentico, D. W. (2007). Assignment problems : A golden anniversary survey. *European Journal of Operationnal Research*, 176 :774–793.

[.Perez, 1990] .Perez, J. (1990). *Systèmes Temps Réels : Méthodes de spécification et de conception*. Dunod.

[Perny, 2000] Perny, P. (2000). *Modélisation des préférences, agrégation multicritère et systèmes d'aide à la décision*. PhD thesis, Habilitation à diriger des recherches, Université Paris 6.

[Phan et al., 2004] Phan, T. H., Gerard, S., and Terrier, F. (2004). Real time system modeling wilth accord/uml methodology : illustration throw an automobile case study. In *Languages for system specification*.

[Pop et al., 2007] Pop, P., Poulsen, K., and Isosimov, V. (2007). Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *International Conference on Hardware-Software Codesign and System Synthesis CODES+ISSS' 07*.

[Qin and Jiang, 2006] Qin, X. and Jiang, H. (2006). A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. *Parallel Computing*, 32 :331–356.

[Qin et al., 2002] Qin, X., Jiang, H., and Swanson, D. R. (2002). An efficient fault-tolerant scheduling algorithm for real time tasks with precedence constraints in heterogeneous systems. In *International Conference on parallel Processing*.

[Rath and Dehuri, 2006] Rath, A. K. and Dehuri, S. N. (2006). Non-dominated sorted genetic algorithms for heterogeneous embedded system design. *Computer Science*, 2 :288–291.

[Roubtsova et al., 2002] Roubtsova, E., Katwijk, J., and Toetenel, J. (2002). Specification of real time systems in uml. *Elsevier, Electronic Notes in Theoretical computer science*, 39 no 2 :12p.

[Roy, 1990] Roy, B. (1990). Science de la décision ou science de l'aide à la décision. Technical report, Université Paris Dauphine, France.

[Saksena and Karvelas, 2000] Saksena, M. and Karvelas, P. (2000). Designing for schedullability : Integrating schedullability analysis with object-oriented design. In *Euromicro Conference on Real-Time Systems*.

[Samur and Bulkan, 2010] Samur, S. and Bulkan, S. (2010). An evolutionary solution to a multi-objective scheduling problem,. In *Proceeding of the World Congress on Engineering*.

[Sandgren, 1994] Sandgren, E. (1994). *Advances in design optimization, chapter Multicriteria design optimization by goal programming*. Chapman and Hall.

[Sandstrom and Fredriksson, 2004] Sandstrom, M. A. K. and Fredriksson, J. (2004). Introducing a component technology for safety critical embedded real-time systems. In *Intern. Symp. on Component-based Softaware Engineering (CBSE7)*.

[Sangiovanni-Vincentelli et al., 2004] Sangiovanni-Vincentelli, A., L, C., de Bernadinis, F., and Segroi, M. (2004). Benefits and challenges for platform-based design. In *Annual Design Automation Conference*.

[Schaffer, 1984] Schaffer, J. D. (1984). *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt university, Nashville Tennessee.

[Schaffer, 1985] Schaffer, J. D. (1985). Multiple objective optimization witn vector evaluated genetic algorithms. In *ICGA International Conference on Genetic Algorithms*.

[Schwarz, 1992] Schwarz, J. J. (1992). Lacatre : Langage d'aide à la conception d'applications multitâches temps réel. *Productique et informatique industrielle*, 26 :355–385.

[Schwarz et al., 1991] Schwarz, J.-J., Skubich, J., and Aubry, R. (1991). Lacatre : The basis of a real time software engineering. In *PEARL*, pages 20–40.

[Selik et al., 1994] Selik, B., Gullekson, G., and Ward, P. T. (1994). *Real-Time Object-Oriented Modeling*. Wiley.

[Sen et al., 1988] Sen, T., Raiszadeh, M., and Dileepan, P. (1988). A branch and bound approach to the bicriterion scheduling problem involving total flowtime and range of lateness. *Management Science*, 34 :254–260.

[Serafini, 1992] Serafini, P. (1992). Simulated annealing for multiple objective optimization problems. In *Tenth Int. Conf. on Multiple Criteria Decision Making*.

[Shatz et al., 1992] Shatz, S., Wang, J. P., and Goto, M. (1992). Task allocation for maximizing reliability of distributed computer systems. *IEEE Transactions on Computers*, 41 :1156–1169.

[Sih and Lee, 1993] Sih, G. C. and Lee, E. A. (1993). A compile-time scheduling heuristic for interconnexion constraint heterogeneous processor architectures. *IEEE. Trans. parallel and Distributed Systems*, 4 :175–187.

[Sommerville, 1988] Sommerville, I. (1988). *Le génie logiciel et ses applications*. Dunod.

[Sorel, 1994] Sorel, Y. (1994). The algorithm architecture adequation methodology. In *Massively Parallel Computing*.

[Sorel, 2004] Sorel, Y. (2004). Syndex : System-level cad software for optimizing distributed real-time embedded systems. *ERCIM News*, 59 :68–69.

[Srinivas and Deb, 1994] Srinivas, N. and Deb, K. (1994). Multiobjective optimization using non dominated sorting in genetic algorithms. *Evolutionary Computation*, 2 :221–248.

[Srinivas and Deb, 1995] Srinivas, N. and Deb, K. (1995). Multiobjective optimization using non-dominated sorting in genetic algorithms. *Evolutionary Computation*, 2 :221–248.

[Stankovic, 1988] Stankovic, J. A. (1988). Misconceptions about real time computing : A serious problem for next-generation systems. *Computer*, 21 :10–19.

[Stein and Wein, 1997] Stein, C. and Wein, J. (1997). On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operations Research Letters*, 21 :115–122.

[Steinberg et al., 2008] Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2008). Emf : Eclipse modeling framework, 2nd edition. In *Collection Eclipses Series*. Addison-Wesley Professional.

[Stewart and White, 1991] Stewart, B. and White, C. (1991). Multiobjective a*. *The ACM*, 38 :775–814.

[Taha, 2008] Taha, S. (2008). *Modélisation conjointe Logiciel/Matériel de Systèmes Temps réel*. PhD thesis, Université des Sciences et Technologies de Lille.

[Talbi, 2002] Talbi, E. G. (2002). A taxonomy of hybrid metaheuristics. *Heuristics*, 8 :541–564.

[Tesanovic et al., 2003] Tesanovic, J. A., Nystrom, D., and Norstrom, C. (2003). Towards aspectual component-based development of real-time systems. In *Real-Time and Embedded Computing Systems and Applications*.

[Topcuouglu et al., 2002] Topcuouglu, H., Hariri, S., and wu, M. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on paralleland Distributed Systems*, 13 :260–274.

[Ulungu et al., 1998] Ulungu, E., Teghem, J., Fortemps, P., and Tuyttens, D. (1998). Mosa method : A tool for solving multi-objective combinatorial optimization problems. Technical report, Laboratory of Mathematic and Operational Research, Faculté Polytechnique de Mons.

[Van Ommering and Van der Linden, 2000] Van Ommering, J. K. R. and Van der Linden, F. (2000). The koala component model for consumer electronics software. *IEEE Computer*, 33 No. 3 :78–85.

[Vincke, 1989] Vincke, P. (1989). *L'Aide Multicritère à la Décision*. Editions Ellipses, Bruxelles.

[Votaw and Orden, 1952] Votaw, D. F. and Orden, A. (1952). The personnel assignment problem. In *Linear inequalities and programming SCOOP*.

[White, 1982] White, D. (1982). The set of efficient solutions for multiple-objectives shortest path problems. *Computers an Operations Research*, 9 :101–107.

[XU and Parnas, 1991] XU, J. and Parnas, D. (1991). On satisfying timing constraints in hard real time systems. *Software Engineering Notes*, 16(4) :132–146.

[Zaffalon, 2007] Zaffalon, L. (2007). Programmation concurrente et temps réel. Technical report, Presses Potytechniques et Universitaires Romandes (PPUR).

[Zhu et al., 1995] Zhu, J., Lewis, T. J., Jackson, W., and Wilson, R. L. (1995). Scheduling in hard real-time applications. *IEEE Software*, 12 :54–63.

[Zitzler et al., 2001a] Zitzler, E., Laumanns, M., and Thiele, L. (, 2001a). Improving the pareto evolutionary algorithm. Technical report, Computer Engineering and Networking Laboratory (TIK) Department of Electical Angineering Swiss Federal Institute of Technology (ETH) Zurich.

[Zitzler et al., 2001b] Zitzler, E., Laummans, M., and Thiele, L. (2001b). Spea2 : Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*.

[Zitzler and Thiele, 1998] Zitzler, E. and Thiele, L. (1998). An evolutionary algorithm for multiobjective optimization : The strength pareto approach. Technical report, Swiss Federal institute of Technology (ETH) Zurich, Switzerland.

Thèse de Doctorat en Sciences en Informatique

Méthodologie de Conception de Systèmes Embarqués Temps Réel

Sonia-Sabrina BENDIB

Résumé

Les systèmes embarqués temps réel sont des systèmes dont les exigences constituent un défi pour les développeurs. En plus d'offrir les fonctionnalités requises, ces systèmes doivent impérativement respecter les contraintes temps réel qui leur sont imposées. Notre travail concerne le développement des systèmes embarqués temps réel où l'intérêt est porté sur une étape non des moindres du développement à savoir l'étape d'ordonnancement. Plus particulièrement, nous considérons l'ordonnancement statique bi-objectifs avec la fiabilité et la longueur d'ordonnancement comme objectifs à optimiser. Deux approches sont proposées, la première est hiérarchique optimisant alternativement les deux objectifs tandis que la seconde est une approche adaptative. L'adaptation consiste à étendre l'espace de recherche assurant de ce fait une meilleure diversité des solutions. Les deux approches sont respectivement intégrées dans la méthodologie 'Adéquation Algorithme Architecture' (AAA) pour en donner deux versions que nous avons appelées AAA-Bi-Hiérarchique pour la première et AAA-Bi-Adaptative pour la seconde.

Mots-clés: Systèmes embarqués temps réel, Méthodologie AAA, Ordonnancement bi-objectifs, GSFR (Global System Failure Rate), Longueur d'ordonnancement, Adaptation, Optimisation multi-objectifs.