Modélisation de plate-forme avionique pour exploration de performance en avance de phase

Michaël Lafaye

▶ **To cite this version:**

Michaël Lafaye. Modélisation de plate-forme avionique pour exploration de performance en avance de phase. Autre. Télécom ParisTech, 2012. Français. NNT : 2012ENST0065 . pastel-01001760

HAL Id: pastel-01001760
https://pastel.hal.science/pastel-01001760

Submitted on 4 Jun 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2012-ENST-065

EDITE ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

Télécom ParisTech

présentée et soutenue publiquement par

Michaël LAFAYE

le 19 novembre 2012

Modélisation de Plate-Forme Avionique pour Exploration de Performance en Avance de Phase

Directeur de thèse : **Pr. Laurent Pautet, Telecom ParisTech**
Directeur de thèse : **Dr. Marc Gatti, Thales Avionics**

Jury

M. Elie NAJM,	Professeur, LTCI, Telecom ParisTech	Président du jury
M. Jean-Charles FABRE,	Professeur, LAAS, Institut National Polytechnique de Toulouse	Rapporteur
M. Paul LE GUERNIC,	Directeur de Recherche, IRISA, Université de Rennes	Rapporteur
M. Jean-Loup TERRAILLON,	Directeur du Service Logiciel Embarqué, Agence Spatiale Européenne	Examinateur
M François PECHEUX,	Enseignant chercheur, LIP6, Université Pierre et Marie Curie Paris 6	Examinateur
M Bernard LECUSSAN,	Professeur, DTIM, Office National d'Etudes et de Recherches Aérospatiales	Examinateur

T H È S E

Télécom ParisTech
Ecole de l'Institut Télécom – membre de ParisTech
46, rue Barrault – 75634 Paris Cedex 13 – Tél. + 33 (0)1 45 81 77 77 – www.telecom-paristech.fr

© 2012 TELECOM ParisTech

Remerciements

Il est temps d'écrire les derniers mots de ce manuscrit, et refermer le chapitre de ma thèse. *"L'important n'est pas la destination mais le voyage"*. Un voyage de trois années dont je retiens, au-delà de l'aboutissement, l'expérience accumulée dans les bons moments comme dans les moments plus compliqués. *"Les pierres font partie du chemin"*.

Je repense également aux différentes rencontres, aux personnes qui ont contribué, d'une manière ou d'une autre, à mener à bien ce voyage.

Je remercie mes encadrants, Laurent Pautet et Marc Gatti, pour m'avoir montré la voie. Leurs conseils m'ont permis de rester sur le bon chemin.*"Pour connaître le chemin, interroge celui qui en vient"*

Je remercie également Jean-Charles Fabre et Paul Le Guernic qui ont accepté d'être les rapporteurs de ce mémoire, ainsi que Bernard Lecussan, Elie Najm, François Pecheux et Jean-Loup Terraillon pour avoir accepté de faire partie de mon jury.

Mes remerciements vont également à mes collègues de Thales. Les nombreux échanges que nous avons eu ont contribué à ce travail, chacun m'apportant son expérience dans ou ou plusieurs domaines connexes à ma thèse.

Je pense bien évidemment à tous mes amis. Les doctorants, docteurs, post-docs et autres personnes de Thales et Telecom : Gilles, Simon, Cuauhtemoc, Fabien, Xavier J., Antoine, Sebastien T., Azin, Hoa, Nora, Asma, Nesrine, Elie, Sylvain, Imen, Joe, Marilena, Damien, Silviu, Greg, Thomas M., Khalil, Donguy, Xavier G., Thomas R., Sebastien G., Etienne, Olivier, Hayette...grâce aux bons moments passés ensemble, le quotidien n'aura pas été que labeur permanent (les sessions projet C vont me manquer !).

Je pense aussi à Tom, Mathilde, Céline et Seb, Elise et Dom, Hayder, David, Sophie...qui, n'étant ni à Thales ni à Telecom, ne m'ont pas forcément vu souvent (la thèse est une maitresse exigeante !) mais sont restés présents.

© 2012 TELECOM ParisTech

Une pensée particulière à ma compagne, Atoosa. Nous avons travaillé sur nos thèses respectives côte à côté, soutenant presque en même temps. Chacun a remonté le moral de l'autre dans les moments difficiles, nous permettant de mener à bien cette aventure, et entamer ensemble un nouveau voyage.

Enfin, je remercie ma famille pour leurs encouragements, et plus particulièrement ma soeur et mes parents. Toujours présents, soutien permanent, je leur dois la réussite de mes études, et cette thèse est aussi la leur.

© 2012 TELECOM ParisTech

Résumé

De nos jours, les systèmes embarqués temps-réels critiques intègrent de plus en plus de composants, et voient leur complexité augmenter. Les systèmes avioniques ont suivi cette évolution, voyant augmenter leurs processus de développement.
Dès lors, les développeurs de plates-formes avioniques se sont tournés vers les méthodes de modélisation en avance de phase (i.e. en tout début de cycle de développement), afin d'anticiper les performances de celles-ci et aider à leur dimensionnement. Particulièrement, l'exploration de l'utilisation des ressources matérielles de la plate-forme par la partie applicative (l'ensemble des applications) est le point central de cette exploration des performances. Si les méthodes de modélisation actuelles offrent la possibilité de modéliser une plate-forme depuis les exigences jusqu'au niveau architectural, elles ne sont pas encore adaptées à la modélisation comportementale. Elle ne permettent donc pas l'étude du comportement et la comparaison de différentes architectures d'une plate-forme en avance de phase.
Mes travaux de thèse ont pour but d'offrir un processus de modélisation et simulation de plate-forme avionique répondant à cette problématique. L'objectif est de compléter les méthodes de modélisation actuelles pour apporter une analyse plus fine des performances d'une plate-forme en avance de phase, et les comparer avec les exigences. Pour cela, nous proposons une approche en quatre étapes : i) une étape de modélisation des applications et d'extraction des stimuli applicatifs ; ii) une étape de modélisation architecturale du système basée sur AADL (Architecture Analysis and Design Language) et son annexe ARINC653 ; iii) une étape de génération d'un modèle comportemental de la partie matérielle et intergicielle du système en SystemC-TLM ; iv) une étape de simulation et d'analyses, où les stimuli applicatifs sont exécutés par le modèle comportemental, et les performances extraites comparées aux exigences système. Enfin, nous avons validé notre méthode sur un cas d'étude avionique que nous présenterons également.

Mots-clés : systèmes embarqués temps-réel critiques, systèmes avioniques, modélisation, simulation, AADL, SystemC-TLM

© 2012 TELECOM ParisTech

Abstract

Nowadays, real-time critical embedded systems are more and more complex due to an increase of the integrated components. Following that trend, avionic systems development complexity increases too.
So early modeling processes are more and more used in order to anticipate on platforms performance and help sizing them. Particularly, hardware resources usage exploration is a key aspect for performance exploration. Current processes allow to model avionic platform from requirements to architectural level of abstraction, but they do not allow to model a behavioral avionic platform. Thus, they do not allow to explore the hardware resources usage of the platform, neither to compare some alternatives of architectures at early phase of development cycle.
My PhD work presents our avionic platform modeling and simulation process that answer that problem. The goal is to complete current modeling processes to offer more accurate early performance analysis, and compare them with the system requirements. For that, we propose a for steps method : i) an application modeling and stimuli extraction step ; ii) an architectural modeling step, based on the AADL (Architecture Analysis and Design Language) and its ARINC653 annex ; iii) a behavioral execution platform model (hardware and middleware) generation step with SystemC-TLM ; iv) a simulation and analysis step, when performance are compared with system requirements. At last, we will present our validation part on an avionic case study.

Keywords : real-time critical embedded systems, avionics systems, modeling, simulation, AADL, SystemC-TLM

© 2012 TELECOM ParisTech

Table des matières

I Introduction Générale 1

1 Introduction 3

1.1 Contexte avionique 4

1.1.1 Application avionique 4

1.1.2 Architecture des systèmes avioniques 5

1.2 Problématique générale 11

1.2.1 Une nouvelle architecture avionique 11

1.2.2 Adapter les méthodes de conception d'une plate-forme . . 11

1.3 Objectifs et approche 12

1.3.1 Objectif 12

1.3.2 Approche 12

1.4 Plan du mémoire 13

II Enjeux Industriels

2 Problématique industrielle 17

2.1 Introduction 18

2.2 Exploration des performances en avance de phase 19

2.3 Modélisation de la plate-forme avionique 20

2.3.1 Niveaux d'abstraction pour la modélisation de plate-forme . 20

2.3.2 Modélisation de systèmes avioniques 22

2.3.3 Résumé 24

2.4 Modélisation d'une application 25

2.4.1 Niveau d'abstraction pour la modélisation 25

2.4.2 Organisation des stimuli 27
2.5 Evaluation des performances de la plate-forme en avance de phase 28
2.6 Synthèse 29

3 Etat de l'Art 31
3.1 Introduction 32
3.2 Langages de modélisation 33
3.2.1 Introduction 33
3.2.2 Langages synchrones et méthodes formelles 33
3.2.3 Langage de modélisation système 34
3.2.4 Langages de modélisation architecturale 34
3.2.5 Langages de description matérielle 36
3.2.6 Résumé 36
3.3 Méthodes de modélisation de plates-formes avioniques 37
3.3.1 Introduction 37
3.3.2 Modélisation d'une plate-forme globale 38
3.3.2.1 Modélisation architecturale 38
3.3.2.2 Modélisation physique 41
3.3.2.3 Modélisation mixte 41
3.3.3 Modélisation du réseau avionique 43
3.3.4 Modélisation des services ARINC653 pour les systèmes d'exploitation avioniques 44
3.3.5 Modélisation d'une application avionique 45
3.4 Langages et méthode de modélisation sélectionnés 46
3.4.1 AADL 47
3.4.1.1 Composants 47
3.4.1.2 Interfaces, connections et déploiement 48
3.4.1.3 Extensions et outils 49
3.4.2 SystemC 50
3.4.2.1 Structure du langage 50
3.4.2.2 Bibliothèque TLM 51
3.4.2.3 Communications 52
3.4.2.4 Outils 54
3.5 Synthèse 54

 © 2012 TELECOM ParisTech

III Approche adoptee

4 Methodologie **59**

4.1 Rappel des objectifs . . . 60

4.2 Structure globale . . . 62

4.3 Capture des paramètres de la partie applicative et génération des stimuli . . . 63

4.3.1 Objectifs . . . 63

4.3.2 Extraction de la configuration logicielle . . . 63

4.3.3 Génération des stimuli applicatifs . . . 64

4.3.4 Résumé . . . 66

4.4 Modélisation architecturale de la plate-forme avionique . . . 67

4.4.1 Objectifs . . . 67

4.4.2 Démarche . . . 67

4.4.3 Introduction des spécificités avioniques . . . 68

4.4.4 Ajout de propriétés . . . 70

4.4.5 Résumé . . . 70

4.5 Génération d'un modèle comportemental de la plate-forme d'exécution . . . 70

4.5.1 Objectifs . . . 70

4.5.2 Génération des composants comportementaux . . . 71

4.5.3 Résumé . . . 74

4.6 Exploration des performances de la plate-forme . . . 74

4.6.1 Objectifs . . . 74

4.6.2 Simulation . . . 74

4.6.3 Analyses . . . 75

4.6.4 Résumé . . . 76

4.7 Synthèse . . . 76

5 Mapping des éléments matériels AADL en SystemC **79**

5.1 Introduction . . . 81

5.2 Modélisation des spécificités avioniques au sein d'un modèle AADL 82

5.2.1 Modélisation du partitionnement temporel . . . 82

5.2.2 Modélisation du partitionnement spatial . . . 83

5.2.3 Modélisation de l'ordonnancement . . . 83

© 2012 TELECOM ParisTech

5.3 Extraction des paramètres du modèle AADL 84
5.3.1 Extraction des paramètres des composants matériels de la plate-forme 84
5.3.2 Extraction des paramètres de la partie applicative de la plate-forme 85
5.3.3 Extraction des paramètres de connexion de la plate-forme matérielle 85
5.4 Génération de la plate-forme matérielle SystemC 86
5.4.1 Description du comportement d'un composant 86
5.4.1.1 Base d'automates comportementaux 87
5.4.1.2 Extension des paramètres AADL pour l'adressage des automates SystemC 88
5.4.1.3 Configuration de l'automate 90
5.4.1.4 Modélisation du système d'exploitation 91
5.4.2 Génération d'une interface générique : le container matériel 92
5.4.2.1 Rappel du principe de communication TLM 92
5.4.2.2 Le Container : principe 94
5.4.2.3 Le Container : mécanisme d'abonnement et filtrage 96
5.4.3 Intégration 98
5.5 Synthèse 100

6 Méthode de génération des stimuli applicatifs en avance de phase 101
6.1 Introduction 102
6.2 Modélisation d'une application en avance de phase 102
6.2.1 Principe 102
6.2.2 Description des tâches applicatives 103
6.3 Génération des stimuli applicatifs 104
6.3.1 Principe 104
6.3.2 Adaptation aux variantes architecturales de la partie matérielle de la plate-forme 105
6.4 Synthèse 108

© 2012 TELECOM ParisTech

IV Validation

7 Expérimentations et Résultats 111
7.1 Introduction 112
7.2 Cas d'Etude 112
7.2.1 Calculateur avionique 112
7.2.2 Plate-forme avionique 113
7.3 Mise en oeuvre du processus sur un calculateur 114
7.3.1 Description des applications 115
7.3.2 Modélisation architecturale du calculateur 117
7.3.3 Génération de la plate-forme d'exécution en SystemC . . . 118
7.3.4 Simulation et analyse des résultats 120
7.3.5 Résumé 122
7.4 Mise en oeuvre du processus sur une plate-forme 123
7.4.1 Présentation de la plate-forme de test 123
7.4.2 Modélisation du réseau ARINC664 124
7.4.3 Modélisation des services de l'API ARINC653 125
7.4.4 Modélisation d'un calculateur au sein de la plate-forme . . . 126
7.4.4.1 Modélisation architecturale 126
7.4.4.2 Modélisation comportementale 127
7.4.5 Simulation et analyses des résultats 128
7.4.5.1 Analyse des temps d'exécution 128
7.4.5.2 Temps de simulation 129
7.4.5.3 Extraction de l'utilisation des ressources matérielles 130
7.5 Synthèse 132

V Conclusion Générale 135

8 Conclusions et Perspectives 137
8.1 Rappel des contributions 138
8.1.1 Génération d'un modèle comportemental de plate-forme d'exécution à partir d'un modèle architectural 138
8.1.2 Modélisation d'une application en avance de phase 139
8.2 Conclusion 139

© 2012 TELECOM ParisTech

8.2.1 Méthodologie 139
8.2.2 Résultats expérimentaux 140
8.2.3 Automatisation du processus 140
8.3 Limites et perspectives 141
8.3.1 Limites 141
8.3.2 Perspectives 142

Bibliographie **145**

© 2012 TELECOM ParisTech

Liste des illustrations

1.1 Architecture fédérée 5
1.2 Architecture intégrée, dite ”IMA” 6
1.3 Architecture logique d’un réseau avionique 7
1.4 Description d’un calculateur au sein d’une architecture IMA 8
1.5 Exemple d’un schéma d’ordonnancement fixe MAF 10
1.6 Acteurs de la conception d’un calculateur avionique 10

2.1 Cycle de développement d’un système, dit "cycle en V" 19
2.2 Niveaux d’abstractions adressés par les méthodes de modélisation actuelles 22
2.3 Exemple de réponse à un stimuli pour différents niveaux d’abstraction . 24
2.4 Décomposition d’une application à différents niveaux d’abstraction . . 26
2.5 Relation entre les niveaux de modélisation et les blocs applicatifs . . . 27

3.1 Aperçu des langages de modélisation de systèmes embarqués 37
3.2 Processus de modélisation mixte 42
3.3 Architecture d’un système avionique décrite par le standard ARINC653 44
3.4 Exemple de modélisation architecturale à l’aide des composants AADL 47
3.5 Structure d’un système modélisé en SystemC 50
3.6 Traitement d’une trame TLM 53

4.1 Méthode de modélisation : vue générale 62
4.2 Génération d’une séquence de stimuli à partir d’un bloc algorithmique . 65
4.3 Elaboration d’un scénario de simulation 66
4.4 Modèle architectural d’une plate-forme 69
4.5 Génération d’un modèle comportemental de composant matériel . . . 72
4.6 Exemple de représentation comportementale d’un service : *get partition status* 73
4.7 Phase de simulation de la plate-forme matérielle et système d’exploitation 75

5.1 Modélisation du partitionnement temporel en AADL 82
5.2 Modélisation du partitionnement spatial en AADL 83
5.3 Exemple d'ordonnancement de type ARINC-653 84
5.4 Exemple d'automate comportemental 88
5.5 Processus de sélection de l'automate comportemental 89
5.6 Configuration d'un composant SystemC au sein de notre méthode . . 90
5.7 Traitement des communications par le couple container-composant . . 97
5.8 Processus de transformation du modèle architectural AADL vers le niveau comportemental SystemC 99

6.1 Méthode de génération de stimuli par patrons de génération 105
6.2 Modification de l'architecture sans modification du jeu de stimuli 106
6.3 Modification de l'architecture avec modification du jeu de stimuli 107

7.1 Architecture du coeur du calculateur avec partition applicative 113
7.2 Architecture de la plate-forme de test 114
7.3 Représentation architecturale du calculateur de test 117
7.4 Exemple de caractérisation de l'automate comportemental SystemC de mémoire à partir du modèle AADL 119
7.5 Acheminement d'un stimulus vers le composant cible 120
7.6 Comparaison des temps d'exécution des trois applications 121
7.7 Représentation de l'écart moyen entre l'exécution réelle et la simulation sur modèle des trois applications tests 121
7.8 Comparaison des temps d'exécution minimum, moyen et maximum pour l'application calcul de Pi 122
7.9 Architecture simplifiée de la plate-forme de test 123
7.10 Représentation du modèle architectural AADL du calculateur de test au sein de la plate-forme 126
7.11 Schéma du modèle comportemental du calculateur de test généré en SystemC 128
7.12 Comparaison entre les temps d'exécution réels et simulés pour les différents codes exécutés 129
7.13 temps de simulation des différents codes exécutés 129
7.14 Comparaison entre les temps d'exécution réels et simulés pour les différents codes exécutés 130
7.15 Comparaison entre les temps d'exécution réels et simulés pour les différents codes exécutés 131

 © 2012 TELECOM ParisTech

Première partie

Introduction Générale

Chapitre 1

Introduction

SOMMAIRE

1.1 CONTEXTE AVIONIQUE 4
1.1.1 Application avionique 4
1.1.2 Architecture des systèmes avioniques 5
1.2 PROBLÉMATIQUE GÉNÉRALE 11
1.2.1 Une nouvelle architecture avionique 11
1.2.2 Adapter les méthodes de conception d'une plate-forme . 11
1.3 OBJECTIFS ET APPROCHE 12
1.3.1 Objectif 12
1.3.2 Approche 12
1.4 PLAN DU MÉMOIRE 13

1.1 Contexte avionique

Suivant la loi de Moore, la miniaturisation des composants électroniques a conduit à une augmentation du nombre de composants intégrés au sein d'une puce, augmentant par là ses performances. Le domaine avionique, qui constitue notre cadre de recherche, a été confronté comme tous les domaines industriels à la nécessité d'augmenter la puissance de calcul de ses systèmes, afin de pouvoir traiter un plus grand nombre d'applications. Toutefois, l'intégration de nouveaux composants ou nouvelles technologies ne peut se faire qu'en adaptant ceux-ci aux contraintes spécifiques du domaine, notamment les contraintes de sécurité et de consommation énergétique.

Plate-forme avionique : nos travaux portent sur les systèmes que l'on appelle plate-forme avionique. Il s'agit d'un système embarqué temps réel critique, composé de plusieurs calculateurs chargés d'exécuter les fonctions avioniques qui permettent le bon déroulement du vol. Comme le précise [Brunette et al., 2005], ces calculateurs sont eux-mêmes des sous-systèmes avioniques, composés d'éléments matériels, d'un système d'exploitation et hébergeant une ou plusieurs applications. En tant que système embarqué, la plate-forme doit pouvoir fonctionner avec des contraintes de ressources limitées, comme un espace mémoire limité ou encore une consommation énergétique autorisée plafonnée. La notion de temps réel implique que le système doit exécuter ses fonctions dans le respect de contraintes temporelles (deadlines), au risque de défaillances graves. Enfin, l'aspect critique induit que le non respect de ces contraintes temporelles peut avoir des conséquences dramatiques allant jusqu'au crash de l'avion et la perte de vies humaines.

1.1.1 Application avionique

Au niveau logiciel, le développement d'une application avionique se fait suivant le respect de la norme DO-178B, "Règlementation pour le développement de logiciels dans le secteur aéronautique" [RTCA, 1992]. Dans l'optique de la certification, chaque application doit, en plus de fournir certains documents de conception, être conforme à certains objectifs de tests et de couverture de tests. Il existe plusieurs niveaux de criticité au sein de la DO-178B, appelés niveaux DAL (Design Assurance Level), traduisant les conséquences d'une erreur dans le fonctionnement d'une application :

© 2012 TELECOM ParisTech

- Niveau DAL A : conséquences catastrophiques ;
- Niveau DAL B : conséquences dangereuses ;
- Niveau DAL C : conséquences majeures ;
- Niveau DAL D : conséquences mineures ;
- Niveau DAL E : sans effet sur la sécurité.

A titre d'exemple, une application chargée du contrôle des commandes de vol sera d'un niveau DAL A, tandis que les applications de gestion des services de détente des passagers (projection de films, etc.) seront DAL E. Ainsi, les objectifs de tests et de couverture de tests sont différents suivant le niveau DAL de l'application. Au niveau le plus élevé (niveau A), la couverture de code par les jeux de tests doit être complète, c'est-à-dire qu'elle doit couvrir toutes les conditions d'exécution et vérifier que le comportement dans telle condition est conforme au scénario prévu.

1.1.2 Architecture des systèmes avioniques

Architecture fédérée : auparavant, l'exécution des applications avioniques se faisait au travers d'une architecture dite fédérée. Comme le montre la figure 1.1, chaque calculateur a la charge d'exécuter une unique application. De plus, les communications entre applications de chaque calculateur se font via un canal dédié. Cette architecture implique que chaque calculateur est conçu et optimisé pour l'application qu'il doit exécuter, de même que les interfaces et liens de communications doivent être adaptés aux besoins des échanges à effectuer.

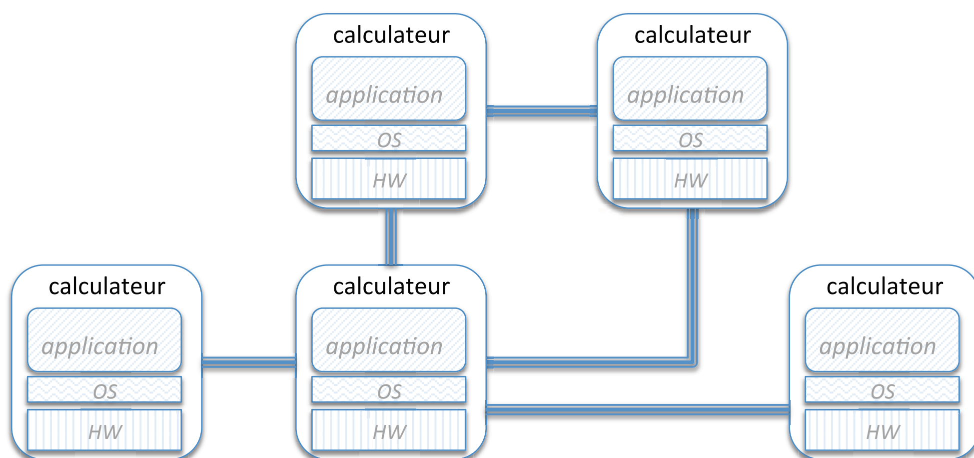

FIGURE 1.1 – Architecture fédérée

© 2012 TELECOM ParisTech

Architecture intégrée : lors de la dernière décennie, une nouvelle architecture a fait son apparition que l'on peut voir sur la figure 1.2. Cette architecture suit un nouveau principe dit d'architecture intégrée ou IMA (Integrated Modular Avionics). Les enjeux de cette nouvelle architecture sont de concevoir des calculateurs plus génériques, configurables, et pouvant exécuter plusieurs applications dont le niveau DAL est différent.
L'IMA a pour but de réduire l'empreinte environnemental de la plate-forme : moins de calculateurs implique une consommation d'énergie et un poids réduits, et moins d'espace occupé dans l'appareil [Li and Xiong, 2009]. Un autre aspect qui accompagne l'IMA concerne les communications entre les calculateurs. Auparavant entièrement dédiées, certaines communications se font désormais au travers d'un réseau standardisé, l'ARINC-664, que présentent [Alena et al., 2006] et [Alena et al., 2007].

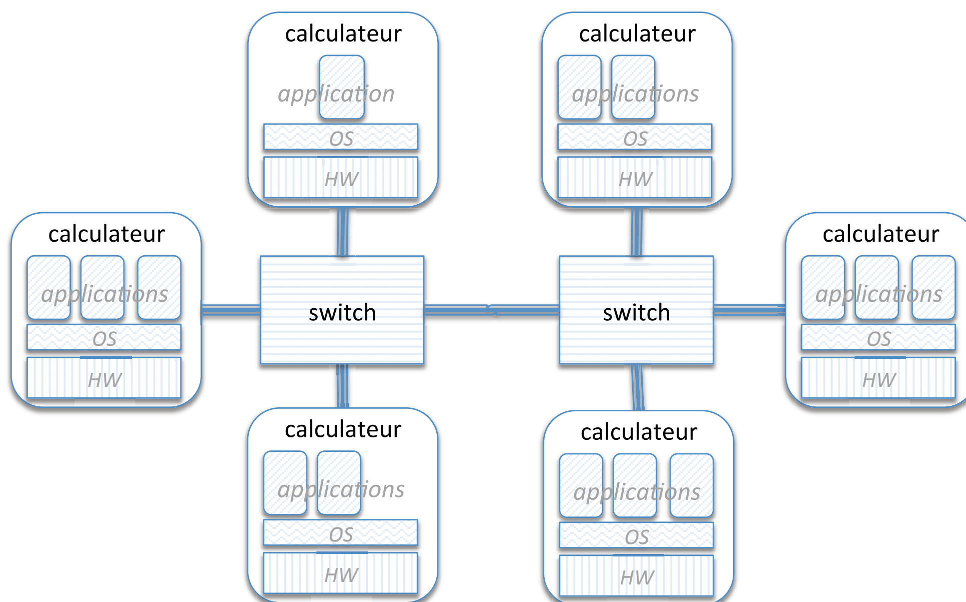

FIGURE 1.2 – Architecture intégrée, dite "IMA"

© 2012 TELECOM ParisTech

ARINC-664 : réseau avionique : l'ARINC-664, via le standard [Aeronautical-Radio-Inc, 2002c] définit un réseau déterministe basé sur Ethernet (norme 802.3). Comme le précise la partie 2 du standard [Aeronautical-Radio-Inc, 2002b], il assure la bonne transmission des données d'un calculateur à l'autre dans le respect de contraintes temporelles. Le fait de regrouper la majeure partie des communications au sein de ce réseau autorise là aussi une réduction de l'espace occupé, et sa réutilisation dans différentes architectures.
Concernant la description physique du réseau, celui-ci est composé de terminaux et de switchs. Les terminaux communiquent directement avec le calculateur et envoient les données sur le réseau. Les switchs sont chargés de router les messages vers le bon terminal. Concernant la partie logique, des "Virtual Links" (VL), ou chemins logiques, sont définis lors de la configuration du réseau. Ils représentent les différentes routes utilisées pour faire transiter les données d'un terminal à un autre. [Charara, 2007] nous donne plusieurs exemples de configurations d'un réseau ARINC664. La partie 1 du standard, [Aeronautical-Radio-Inc, 2002a], définit que chaque chemin logique reçoit une priorité et une bande passante, permettant d'acheminer les données en un temps maximum borné et connu. Le schéma 1.3 nous offre une vue logique du réseau, représenté par les chemins de communications autorisés entre les différents calculateurs.

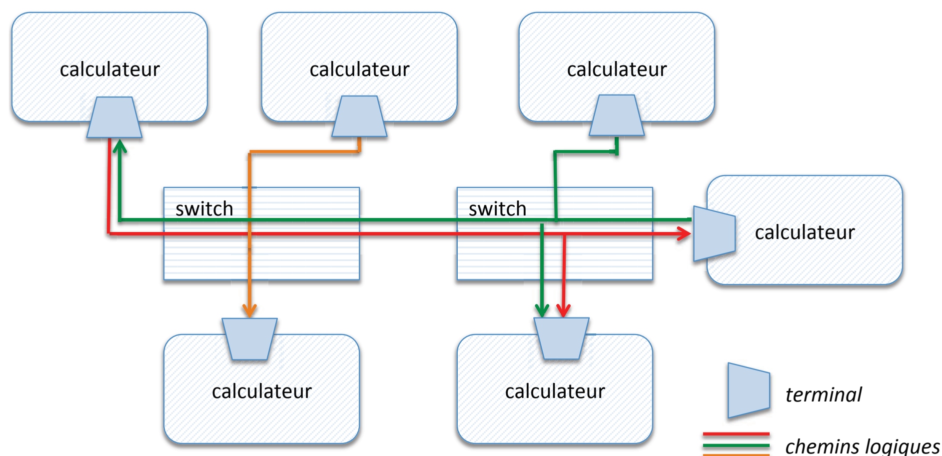

FIGURE 1.3 – Architecture logique d'un réseau avionique

© 2012 TELECOM ParisTech

ARINC-653 : interface de programmation standard pour architecture intégrée : le principe d'architecture intégrée permet d'assurer une indépendance entre le développement des applications et le développement de la plate-forme d'exécution (système d'exploitation et architecture matérielle). En effet, une application avionique comporte deux ensembles d'instructions : les instructions de calcul et les appels aux services du système d'exploitation. Ce dernier offre un ensemble de services standardisés par le standard ARINC-653 [Aeronautical-Radio-Inc, 2006], et permet, comme nous le voyons sur la figure 1.4, la séparation des applications vis-à-vis de la plate-forme d'exécution sous-jacente.

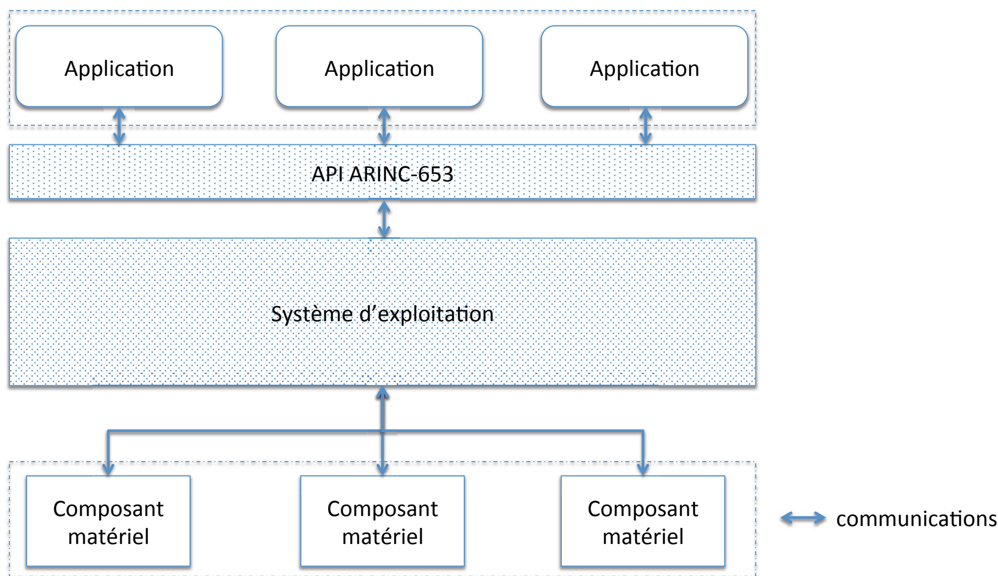

FIGURE 1.4 – Description d'un calculateur au sein d'une architecture IMA

Le standard ARINC-653 [Aeronautical-Radio-Inc, 1997] permet de faire cohabiter les différentes applications d'un même calculateur tout en respectant les principes de sûreté. Car si l'IMA permet de réduire le nombre de calculateurs en regroupant plusieurs applications sur un nombre réduit d'entre eux, cela ne doit pas se faire au détriment de la performances ni de la sûreté. Une application doit avoir accès aux ressources matérielles suffisamment souvent, et ce malgré la "concurrence" des autres applications sur le même calculateur. De même, une erreur lors de l'exécution de l'une de ces applications ne doit pas avoir de conséquences sur les autres applications hébergées sur le même calculateur. Afin de respecter ces exigences de sûreté, le standard ARINC-653 définit le principe de partitionnement spatial et temporel. Chaque application avionique est découpée en une ou plusieurs partitions, "boites" logicielles

 © 2012 TELECOM ParisTech

isolées les unes des autres. Cette isolation permet de contenir une erreur d'exécution au sein d'une partition, sans contagion aux autres applications. Ainsi pour résumer, nous avons deux types de partitionnement :

- Le partitionnement spatial : chaque partition logicielle possède son propre espace mémoire dans lequel elle seule peut aller écrire ;
- Le partitionnement temporel : chaque partition s'exécute au moins une fois pendant la MAF. Pendant cette période, la partition a accès à l'ensemble des ressources matérielles auxquelles elle est "abonnée" (processeur, zone mémoire dédiée et les entrées-sorties auxquelles elle a l'autorisation d'accéder).

Les services ARINC653 peuvent être classés en six catégories principales :

- Gestion des partitions : fixer / récupérer le mode de la partition (IDLE, COLD, NORMAL, etc.) ;
- Gestion des processus : récupérer les valeurs des propriétés (identifiant, priorité), suspendre le processus, le stopper, le relancer, etc. ;
- Gestion du temps : mettre en attente un processus, récupérer le temps absolu, etc. ;
- Gestion des communications inter-partitions : gestion des queues de messages inter-partition (queuing), gestion des tampons de messages inter-partition (sampling) ;
- Gestion des communications intra-partitions : gestion des queues de messages intra-partition (buffer), gestion des tampons de messages intra-partition (blackboard), gestion des sémaphores et évènements ;
- Gestion des pannes.

L'ordonnancement des partitions se fait suivant un schéma d'exécution fixe afin d'assurer le déterminisme lors de l'exécution. Ce schéma d'exécution est nommé MAF (Major Frame) dans le standard ARINC-653, dont nous donnons un exemple dans la figure 1.5. La MAF assure l'accès à l'ensemble des ressources matérielles à une partition pendant au moins une fenêtre temporelle. Au sein de chaque partition s'exécutent des processus, équivalents à des tâches logicielles, qui sont ordonnancées suivant un schéma adapté aux tâches périodiques comme RMS (Rate Monotonic Scheduling) [Liu and Layland,].

© 2012 TELECOM ParisTech

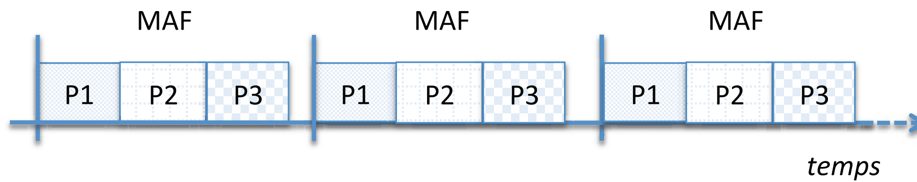

FIGURE 1.5 – Exemple d'un schéma d'ordonnancement fixe MAF

Nous avons vu que l'IMA permet de dissocier le développement des applications du développement du module (système d'exploitation et partie matérielle). Cette indépendance a renforcé la spécialisation des différents acteurs participant à la conception d'un calculateur. Nous distinguons trois catégories d'acteurs, comme le résume le schéma 1.6. Les applications sont généralement développées par les fournisseurs d'applications, ou développeurs d'applications ; la plate-forme matérielle (composants matériels et système d'exploitation) est développée par le fournisseur de module ou plate-formiste, tandis que l'intégration se fait par l'intégrateur, généralement l'avionneur (Aribus, Boeing, etc.). Nos travaux s'inscrivent dans le domaine du fournisseur de module ce qui signifie que nous avons accès aux informations concernant le module (système d'exploitation et composants matériels), mais pas forcément aux applications qui seront exécutées. Cette précision est utile car elle est une composante de notre problématique qui va être décrite dans le chapitre 2.

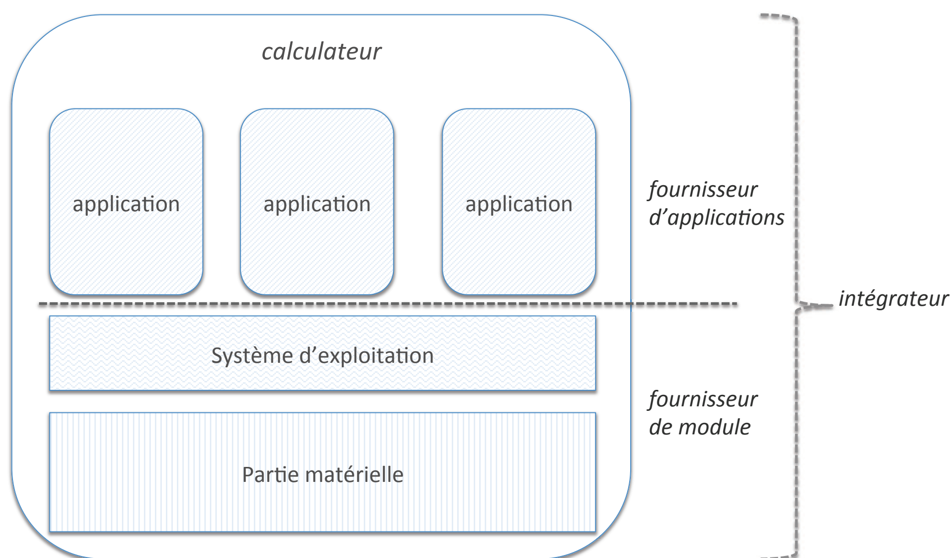

FIGURE 1.6 – Acteurs de la conception d'un calculateur avionique

© 2012 TELECOM ParisTech

1.2 Problématique générale

1.2.1 Une nouvelle architecture avionique

Nous venons de voir que le principe IMA permet de développer une plate-forme avionique à l'empreinte réduite (en terme de consommation d'énergie, d'espace occupé et de poids), tout en conservant les performances et les propriétés de sûreté en place dans les architectures fédérées. Toutefois, le fait de regrouper plusieurs applications de niveau DAL différents sur un même calculateur a entrainé une augmentation de la complexité de développement par rapport aux architectures fédérées : dans ces dernières, chaque calculateur exécute une seule application, la non propagation d'une faute est ainsi en grande partie assurée par l'architecture elle-même de par l'absence d'autres applications sur le même hôte. Dans le cas de l'IMA, la mise en place des mécanismes spécifiques a été nécessaire afin d'assurer le respect de ce type de contrainte.

1.2.2 Adapter les méthodes de conception d'une plate-forme

Pour faire face à cette augmentation de la complexité et tenir les délais de livraison, il a été nécessaire d'adapter les méthodes de conception actuelles, voire en développer de nouvelles [Ott, 2007]. Les phases de conception détaillée et d'implémentation se sont adaptées aux nouvelles contraintes induites par l'IMA, et n'offrent désormais que peu de marge de manoeuvre pour tenter de réduire les temps de développement. Les recherches en vue d'optimiser le processus de développement se sont donc tournées vers la phase amont du processus, constituée par les phases de spécification et pré-dimensionnement de la plate-forme. Dans ce domaine, de nouvelles méthodes liées à la modélisation de plate-forme ont vu le jour. Toutefois ces méthodes, qui permettent de modéliser des systèmes temps-réels, ne sont pas forcément adaptées aux spécificités de l'avionique (partitionnement, réseau commun déterministe, etc.), et ne sont souvent pas suffisamment robustes pour être utilisées en phase amont du processus, notamment au niveau de la modélisation de l'applicatif en avance de phase. Nous verrons dans le chapitre 2 quelles sont les spécificités d'une modélisation en avance de phase, et les points techniques à résoudre. Dans le chapitre 3 nous verrons plusieurs méthodes de modélisation de systèmes temps-réel et leurs limitations pour l'application au domaine avionique en avance de phase.

© 2012 TELECOM ParisTech

1.3 Objectifs et approche

1.3.1 Objectif

Notre objectif principal est d'aider au dimensionnement d'une architecture matérielle avionique en fonction d'applications qu'elle aura à exécuter. Plus spécifiquement, l'objectif est de fournir une méthode de modélisation offrant la possibilité d'anticiper les performances, notamment temporelles, de la plate-forme, afin de pré-valider l'architecture au regard des exigences qu'elle doit respecter. Ce modèle pourra être amené à servir de base de travail lors des discussions autour de la plate-forme entre le plate-formiste et l'avionneur. Pour répondre à cela, la modélisation doit prendre en compte les spécificités des architectures avioniques actuelles, et doit être capable de fournir des résultats suffisamment représentatifs en s'adaptant aux contraintes de la modélisation en toute phase amont du processus de développement.

1.3.2 Approche

Nous proposons une méthode de modélisation de la plate-forme avionique en plusieurs étapes [Lafaye et al., 2010a]. Une première étape d'extraction des caractéristiques dimensionnantes de la partie applicative en avance de phase, deux étapes de modélisation à différents niveaux d'abstraction de la plate-forme prenant en compte les spécificités avioniques, et enfin une étape d'analyse des performances de la plate-forme au regard des exigences. Cette approche s'appuie sur des méthodes existantes de modélisation architecturale, et vient enrichir celles-ci afin de descendre en abstraction et fournir un modèle comportemental de la plate-forme d'exécution (partie matérielle et système d'exploitation). Ce modèle comportemental autorise l'exécution de stimuli applicatifs et l'analyse dynamique de la réponse de la plate-forme à ces stimuli, via l'étude de la consommation de ses ressources. L'analyse des temps d'exécution suite à la réception d'un stimulus, ou encore l'analyse de la consommation d'énergie de la plate-forme sur une période donnée, sont quelques-unes des analyses que l'on souhaite pouvoir mener. Ces analyses sont ensuite mise en regard des spécifications afin de pré-valider ou non l'architecture proposée pour la plate-forme.

© 2012 TELECOM ParisTech

1.4 Plan du mémoire

Ce mémoire s'organise autour de quatre parties principales, dans lesquelles nous allons détailler notre approche, les choix effectués, son implémentation et la validation expérimentale. La partie II définit le cadre de nos travaux. Nous détaillerons dans le chapitre 2 les problèmes spécifiques liés à la modélisation d'un système avionique en avance de phase, en posant les objectifs à atteindre par notre approche. Dans le chapitre 3 nous présenterons les méthodes de modélisation de systèmes temps-réels, et plus spécifiquement celles touchant au domaine avionique. Nous y verrons les atouts et manques de chacune d'entre elles .

La partie III présentera notre approche en elle-même. Dans le chapitre 4, nous verrons l'approche proposée afin de répondre aux objectifs posés dans le chapitre précédent, ainsi que les points techniques auxquels il a fallu répondre. Nous verrons comment chaque partie du processus répond à l'un de ces points. Les chapitres 5 6 détailleront nos contributions qui mettent en oeuvre notre approche, ainsi que les choix techniques et solutions adoptés.

La partie IV via le chapitre 7 présentera les expérimentations effectuées afin de valider notre approche sur un cas d'étude avionique réel fourni par notre partenaire Thales Avionics, et mettra l'accent sur la mise en oeuvre des contributions. Enfin, conclurons dans le chapitre 8 de la partie V.

© 2012 TELECOM ParisTech

© 2012 TELECOM ParisTech

Deuxième partie

Enjeux Industriels

Chapitre 2

Problématique industrielle

Sommaire

2.1 Introduction . . . 18

2.2 Exploration des performances en avance de phase . . . 19

2.3 Modélisation de la plate-forme avionique . . . 20

- 2.3.1 Niveaux d'abstraction pour la modélisation de plate-forme 20
- 2.3.2 Modélisation de systèmes avioniques . . . 22
- 2.3.3 Résumé . . . 24

2.4 Modélisation d'une application . . . 25

- 2.4.1 Niveau d'abstraction pour la modélisation . . . 25
- 2.4.2 Organisation des stimuli . . . 27

2.5 Evaluation des performances de la plate-forme en avance de phase . . . 28

2.6 Synthèse . . . 29

2.1 Introduction

Nous venons de voir que la conception des plates-formes avioniques est une opération dont la complexité s'est accrue ces dernières années, du fait de l'IMA. Nous avons vu dans le chapitre 1 que les deux objectifs principaux de ce type d'architecture peuvent se résumer à :

- Permettre la réduction du nombre de calculateurs au sein d'une plate-forme, afin de réduire l'empreinte de cette dernière (diminution de la consommation, diminution de l'espace occupé et diminution du poids de la plate-forme) ;
- Faciliter la réutilisation des calculateurs lors de la conception de nouvelles plates-formes, en apportant plus de généricité.

Toutefois, ce changement d'architecture a engendré des changements dans les règles de conception. Par exemple, dans les architectures fédérées, le fait de n'autoriser qu'une application par calculateur permet, en cas de défaillance, de limiter relativement facilement l'impact de cette défaillance sur les autres applications hébergées sur les autres calculateurs. Sur les architectures IMA, où un calculateur peut héberger plusieurs applications, il est nécessaire de garantir que chaque application n'aura que des interactions très limitées avec les autres applications afin d'éviter la propagation d'erreurs. Le respect de ce type de contraintes dans les nouvelles architectures avioniques a engendré le développement de nouveaux mécanismes qui ont un impact sur la complexité du processus de conception et de vérification.

Parallèlement, les avionneurs demandent à pouvoir disposer des plates-formes de plus en plus tôt. Les plates-formistes ont donc cherché à mieux orienter leur processus de développement, afin de détecter au plus tôt certaines erreurs de conception et limiter leur impact sur la suite du développement. Notamment, afin de dimensionner au mieux la plate-forme en fonction des exigences, les plates-formistes se sont tournés vers la modélisation de plate-forme en avance de phase avec pour but d'offrir une meilleure représentation des spécifications de la plate-forme, et mieux rendre compte des choix architecturaux.

 © 2012 TELECOM ParisTech

2.2 Exploration des performances en avance de phase

Cette opération consiste en une exploration des performances de la plate-forme en phase amont du processus de conception. Les performances sont généralement des performances temporelles, mais peuvent également être d'un autre type comme la consommation d'énergie. Si l'on reprend le cycle de développement classique en V de la figure 2.1, l'avance de phase correspond à la phase de spécifications et de conception générale. A ce niveau là, l'architecture du système n'est pas encore concrète, mais décrite par un ensemble de documents qui définissent les spécifications matérielles et logicielles que doivent respecter la plate-forme.

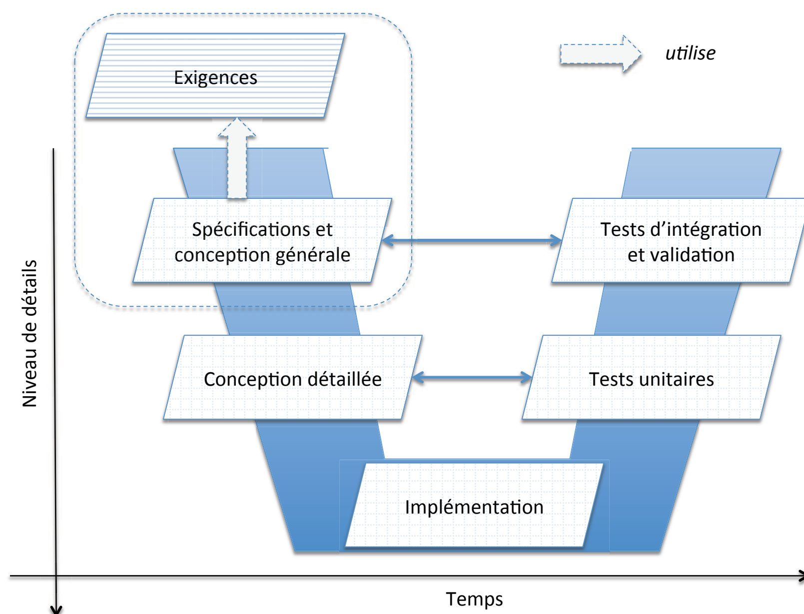

FIGURE 2.1 – Cycle de développement d'un système, dit "cycle en V"

Dans l'idéal, l'exploration des performances en avance de phase permet d'étudier la réponse de la plate-forme à l'exécution d'une ou plusieurs applications. Pour cela sont analysés i) l'utilisation des ressources matérielles de la plate-forme, ii) les limites d'utilisation de la plate-forme. Ces analyses permettent ensuite de vérifier le respect des exigences. Toutefois, à ce niveau très amont du processus de conception, l'ensemble des informations concernant la plate-forme n'est pas toujours fourni ou sont limitées. Nous avons vu par exemple dans le chapitre 1 que le code source des applications n'est que rarement accessible. Ceci exclut l'utilisation de méthodes de virtualisation pour la prédiction de performances, car elles nécessitent le binaire

© 2012 TELECOM ParisTech

d'une application pour l'exécuter sur un modèle détaillé d'architecture.
D'autre part, l'opération de modélisation et d'exploration des performances d'une plate-forme en avance de phase doit être relativement rapide. L'objectif étant de mieux orienter le processus de développement pour limiter la perte de temps causée par des problèmes de dimensionnement d'architecture, il est impératif de ne pas perdre plus de temps lors de la modélisation que l'on en gagne sur la suite du processus. Ainsi, les méthodes de modélisation fine de plate-forme basées sur des modèles très détaillés de type VHDL ou Verilog, sont eux aussi exclus.

Depuis quelques années, des méthodes de modélisation en avance de phase de systèmes embarqués ont fait leur apparition afin d'offrir un modèle de plate-forme accompagné d'outils d'analyses. Ces méthodes ont été enrichies afin de prendre en compte les aspects temps-réels. Par exemple, la méthode de modélisation décrite dans [Dissaux and Singhoff, 2008] permet de modéliser un système temps-réel, et d'effectuer de premières analyses sur le dimensionnement des ressources telle que l'allocation de mémoire. Toutefois, dès lors que l'on cherche à modéliser en avance de phase des systèmes plus spécifiques comme les systèmes avioniques, les méthodes actuelles souffrent de manques i) au niveau de la modélisation des différentes parties du système, du fait du manque d'informations, ii) de la prise en compte des spécificités du domaine avionique. Dans la suite, nous allons détailler les trois étapes clés de l'exploration en avance de phase des performances d'une plate-forme avionique. Nous donnerons les contraintes liées au domaine avionique et les différents problèmes auxquels se heurtent les méthodes actuelles.

2.3 Modélisation de la plate-forme avionique

2.3.1 Niveaux d'abstraction pour la modélisation de plate-forme

Les méthodes de modélisation de systèmes ont fait leur apparition dans le but, dans un premier temps, de rendre plus intelligible par les utilisateurs les spécifications logicielles et matérielles d'un système en les traduisant en un modèle architectural. Ces méthodes se sont peu à peu enrichies, offrant de plus en plus de possibilités de modélisation, pour finalement offrir de modéliser la plate-forme à plusieurs niveaux d'abstraction. Nous distinguons principalement cinq niveaux de modélisation :

© 2012 TELECOM ParisTech

- Niveau modélisation des exigences, où ces dernières sont exprimées de manière à pouvoir y associer les futures fonctionnalités logicielles ou matérielles ;
- Niveau modélisation fonctionnelle, où l'on décrit les fonctionnalités que devra remplir le système (calcul de trajectoire, calcul d'altitude, gestion de la lumière du cockpit etc.) ;
- Niveau modélisation logique, où les fonctionnalités définies ci-dessus sont découpées en une ou plusieurs fonctions, au sens informatique ;
- Niveau modélisation architecturale, où le système est modélisé sous forme de composants logiciels déployés sur des composants matériels ;
- Niveau modélisation comportementale, où le système est modélisé sous forme de composants comportementaux plus ou moins détaillés réagissant à des stimuli applicatifs.

Comme nous le voyons sur le schéma 2.2, seuls les deux derniers niveaux de modélisation, architecturale et comportementale, sont représentatifs d'une plate-forme. C'est-à-dire que ces niveaux permettent de distinguer les parties logicielles et matérielles, contrairement aux deux niveaux d'abstractions qui viennent juste avant.

Ainsi, dans l'optique de mener des analyses de dimensionnement et de respect des exigences système, seuls les niveaux de modélisation architecturale et comportementale nous intéressent. Ces analyses passent par des études ciblées comme l'ordonnançabilité des partitions logicielles par l'ordonnanceur (respect des deadlines), ou l'exploration de l'utilisation des ressources matérielles (utilisation mémoire, ou consommation énergétique du CPU entre autres exemples). Toutefois, pour le domaine avionique, l'utilisation de méthodes de modélisation de plate-formes nécessite l'adaptation des méthodes de modélisation actuelles.

© 2012 TELECOM ParisTech

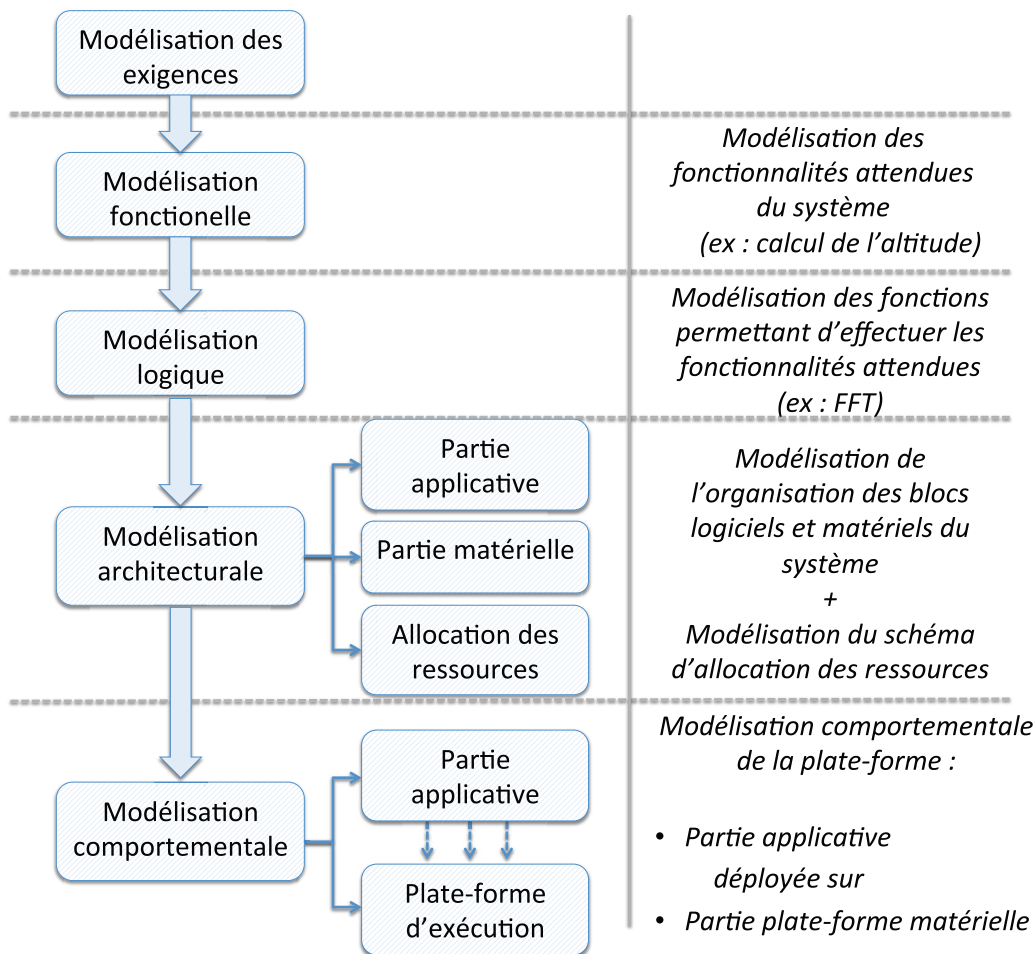

FIGURE 2.2 – Niveaux d'abstractions adressés par les méthodes de modélisation actuelles

2.3.2 Modélisation de systèmes avioniques

Si les méthodes existantes permettent de modéliser en avance de phase les systèmes à haut niveau d'abstraction, elles se limitent souvent à une description architecturale. Les méthodes offrant une modélisation comportementale en avance de phase du système restent marginales pour deux principales raisons. La première concerne les informations souvent limitées disponibles en avance de phase. Particulièrement, le code des applications n'est pas forcément accessible. En effet, le monde de l'avionique, nous l'avons vu dans le chapitre 1, se compose de plusieurs acteurs qui n'interagissent que ponctuellement, chacun développant de son côté une partie de la plate-forme. Les tests d'intégration et les analyses de dimensionnement se font par la

 © 2012 TELECOM ParisTech

suite, lorsque la plate-forme matérielle est développée, c'est-à-dire en aval du processus de conception du système. Nous avons donc ici une première contrainte : le non accès au code source des applications.

La seconde raison freinant la modélisation en avance de phase d'une plate-forme tient au fait que le modèle doit pouvoir être élaboré rapidement. Or, contrairement à une description architecturale où les éléments de la plate-forme sont caractérisés par leurs principales propriétés, la description comportementale de tous ces éléments est une opération plus longue, car elle nécessite de définir en plus leur comportement. De plus, une description comportementale du système d'exploitation, nécessaire si l'on souhaite étudier en détails l'utilisation des ressources matérielles d'une plate-forme, est une opération encore plus longue et compliquée que pour les autres éléments de la plate-forme du fait des mécanismes mis en oeuvre.

Or, si un modèle architectural permet d'effectuer des premières analyses de dimensionnement comme il est décrit dans [de Niz and Rajkumar, 2006] ou [Singhoff et al., 2004] par exemple, celui-ci ne permet pas de venir extraire les l'utilisation des ressources matérielles au cours du temps et suivant le contexte d'exécution. L'absence d'informations comportementales dans la description des composants du modèle empêche d'étudier la réponse d'un ou plusieurs composants à un ou plusieurs stimuli. L'exemple de la figure 2.3 montre qu'un modèle architectural de mémoire configuré par les latences d'accès en lecture et écriture rendra une réponse uniforme en cas de requête, tandis qu'un modèle comportemental de mémoire pourra faire intervenir des différences de latence pour un même stimuli, en fonction du contexte. Nous avons donc ici deux contraintes : temps de modélisation et exploration, et niveau de précision du modèle et des informations que l'on peut en retirer.

© 2012 TELECOM ParisTech

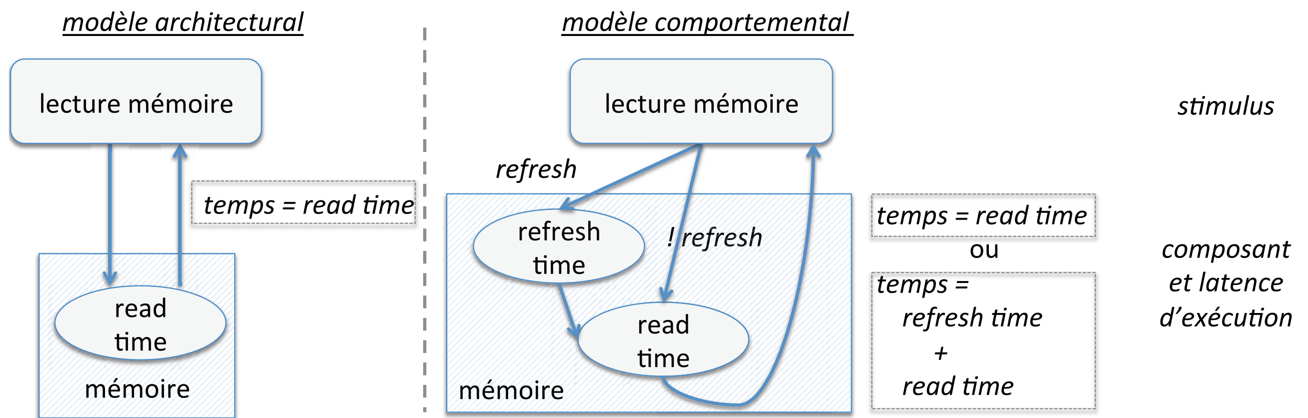

FIGURE 2.3 – Exemple de réponse à un stimuli pour différents niveaux d'abstraction

2.3.3 Résumé

Nous avons vu qu'il est nécessaire de disposer d'un modèle plus fin que le modèle architectural si l'on veut explorer les performances d'une plate-forme et les comparer avec les exigences. Un modèle de type comportemental permet d'extraire la réponse de la plate-forme en réaction aux différents stimuli applicatifs qu'elle devra traiter, et ainsi autorise l'analyse de l'utilisation des ressources de la plate-forme. Toutefois, l'élaboration d'un modèle comportemental précis du système, à l'aide de langages tels VHDL par exemple, est une opération complexe et longue si l'on ne possède pas déjà tous les modèles comportementaux des éléments de la plate-forme. La méthode d'exploration des performances en avance de phase doit donc répondre aux deux critères identifiés suivants :

- Critère 1 : proposer une méthode de modélisation de la plate-forme malgré le manque d'informations disponibles, particulièrement l'absence du code source des applications ;
- Critère 2 : proposer une méthode d'élaboration relativement rapide d'un modèle comportemental de la partie matérielle, qui vient affiner le modèle précédent, afin d'étudier dynamiquement la réponse de celle-ci aux stimuli applicatifs.

 © 2012 TELECOM ParisTech

2.4 Modélisation d'une application

2.4.1 Niveau d'abstraction pour la modélisation

L'objectif de cette étape est de modéliser une application avionique avec pour but d'en extraire ses principales caractéristiques et son comportement, pour ensuite élaborer un scénario de simulation pour le modèle de plate-forme. Nous l'avons vu à plusieurs reprises, les applications amenées à être exécutées par la future plate-forme ne sont ici pas toujours accessibles. Il n'est donc pas possible d'appliquer des méthodes de parcours de code afin d'en retirer leur comportement et les différents paramètres dimensionnants, tels que les appels aux fonctions de l'APEX, comme c'est le cas dans le projet [la Camara et al., 2007]. Or, si les informations extraites à ce stade ne sont pas suffisamment représentatives de l'application, les performances extraites par la suite seront trop imprécises. Il en va de même si l'écart entre le niveau de précision de ces informations et celui du modèle de la plate-forme d'exécution générée en aval du processus de modélisation est trop grand.

Afin de répondre à cette difficulté, il convient dans un premier temps de définir le niveau de granularité des stimuli que l'on souhaite générer. Ceux-ci sont à mettre en relation d'une part avec le modèle comportemental de la plate-forme, et d'autre part avec le niveau de granularité souhaité lors de l'analyse de performances, qui déterminera le niveau de modélisation de l'application. Le temps d'exécution du scénario de simulation est également un facteur à prendre en compte ; en effet, plus la granularité de la modélisation de la partie applicative est élevée (i.e. plus le nombre de stimuli généré dans le scénario est grand), plus l'exécution sera longue. La figure 2.4 nous montre à quels niveaux d'abstraction peut se décomposer une application.

© 2012 TELECOM ParisTech

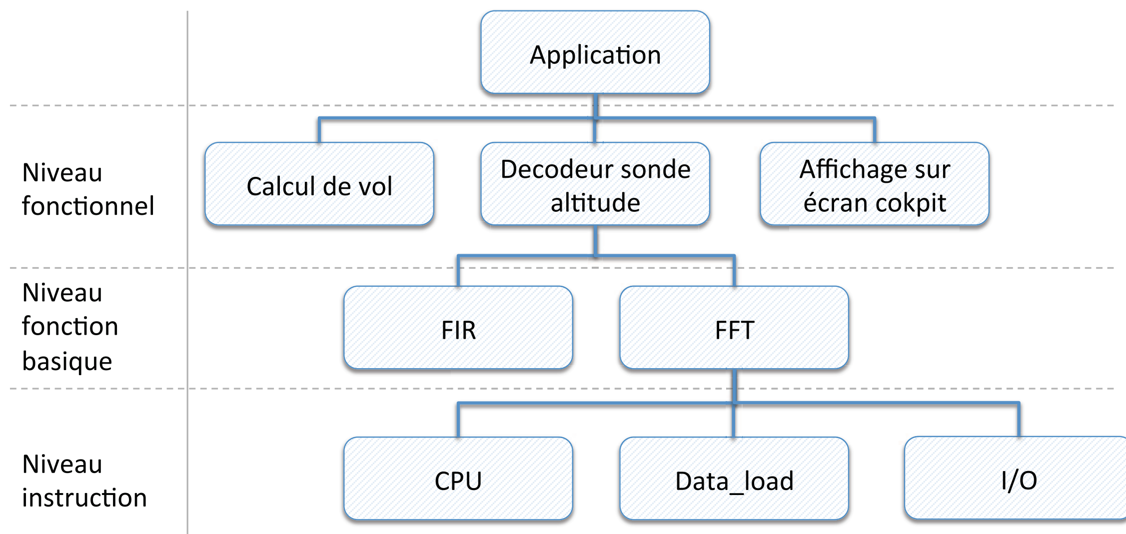

FIGURE 2.4 – Décomposition d'une application à différents niveaux d'abstraction

Ces niveaux rappellent les niveaux de modélisation que l'on a exposé dans la section précédente. Depuis le niveau de l'application elle-même, celle-ci est affinée en blocs fonctionnels, représentant les fonctionnalités attendues de l'application. Vient ensuite le niveau blocs logiques ou fonctions de base, représentant les fonctions -au sens informatique/algorithmique- permettant la réalisation des fonctionnalités précédentes. Enfin, le niveau instructions de bases représente l'accès aux différentes ressources matérielles de la plate-forme.

Or, nous avons vu dans la section précédente que pour mener à bien l'analyse des performances depuis le modèle de plate-forme, ce dernier devait être de niveau comportemental afin de pouvoir étudier les accès et la consommation des ressources matérielles. Ainsi, si l'on se réfère au schéma 2.5, nous voyons que le niveau de granularité instruction correspond le mieux au niveau modélisation comportementale de la plate-forme matérielle.

© 2012 TELECOM ParisTech

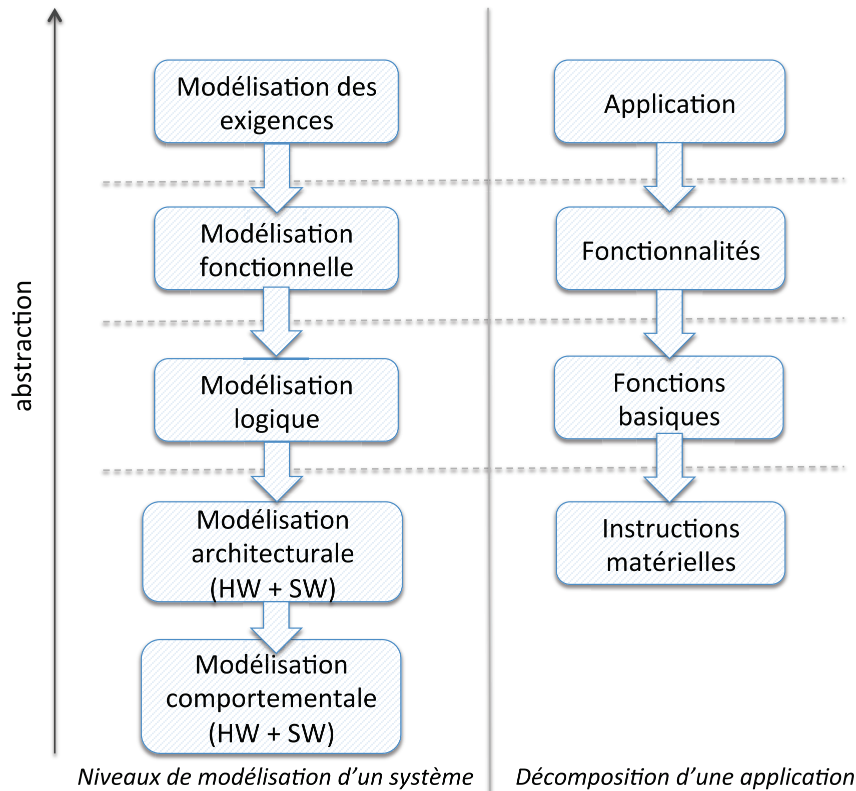

FIGURE 2.5 – Relation entre les niveaux de modélisation et les blocs applicatifs

2.4.2 Organisation des stimuli

La seconde partie de notre étape d'extraction des stimuli applicatifs concerne leur organisation en un scénario de simulation. L'exécution des partitions au sein d'un même calculateur étant effectuée suivant le principe d'isolation temporelle, via le schéma d'exécution fixe, la MAF, nous savons donc déjà comment seront organisées temporellement les partitions. Connaissant également le découpage de la partition en tâches, et le schéma d'ordonnancement associé, il est ainsi possible d'extraire un schéma d'exécution de ces tâches et du code applicatif associé. Les besoins pour la génération de ce scénario de simulation consistent en deux points : i) un ensemble de caractéristiques des tâches et ii) une description comportementale du code de cette tâche pour organiser les futures stimuli décrivant le comportement de la tâche (enchainement des séquences de calcul, d'appel services, etc.). Ces besoins nous ramènent au critère 1 identifié dans la section précédente : fournir une méthode de modélisation comportementale de toutes les parties de la plate-forme, dont la partie applicative à laquelle nous nous sommes particulièrement intéressés ici.

© 2012 TELECOM ParisTech

2.5 Evaluation des performances de la plate-forme en avance de phase

L'objectif de cette étape est double, puisqu'elle doit d'une part offrir certaines garanties quant à la représentativité des résultats obtenus, d'autre part établir si l'architecture proposée pour la plate-forme permet d'exécuter la partie applicative dans le respect des exigences fournies en entrée. Au début de ce chapitre nous avons vu que l'analyse des performances d'une plate-forme passe par l'analyse de l'utilisation de ses ressources. Dans notre cas, nous mettons l'accent sur l'utilisation des ressources matérielles de la plate-forme, afin d'être capable de déterminer la compatibilité entre la partie applicative et la plate-forme matérielle. Nous avons identifié précédemment que les composants de cette partie matérielle doivent être décrits avec leur partie comportementale, afin de pouvoir étudier la réponse d'un composant à un stimuli applicatif.

La validité d'une simulation est définie comme étant la représentativité du comportement du système et des résultats produits lors d'une simulation [Albert, 2009]. La représentativité de la simulation dépend ainsi en premier lieu de la méthode utilisée. S'appuyer sur un moteur de simulation existant et validé permet de s'assurer que l'exécution du scénario de simulation se fait sans erreurs. Outre la bonne exécution du scénario, il est nécessaire de s'assurer que les stimuli sont bien acheminés vers les composants cibles. D'autre part, le niveau de représentativité des résultats de simulation dépend du niveau d'abstraction du modèle de plate-forme, et du niveau de précision des stimuli. Une granularité faible, et donc un niveau d'abstraction élevé, peut entraîner des écarts relativement importants avec l'exécution des applications sur la plate-forme réelle. Enfin, une fois les performances extraites, il est nécessaire de mettre en regard celles-ci avec les exigences données en entrée afin de vérifier la compatibilité entre l'architecture de plate-forme proposée, ses performances, et ces exigences.

© 2012 TELECOM ParisTech

2.6 Synthèse

Nous avons vu que face à l'augmentation de la complexité dans la conception des plates-formes avioniques, l'exploration de leurs performances en avance de phase est un besoin croissant. Elle doit répondre aux critères identifiés dans ce chapitre :

- Critère 1 : l'approche doit fournir un modèle de la plate-forme décrivant les différentes parties du système : la partie applicative, la partie matérielle, le système d'exploitation et l'allocation des ressources. Particulièrement, elle doit fournir ou permettre de générer de manière relativement rapide un modèle comportemental de la partie matérielle et des services ARINC653 utilisés dans l'optique d'étudier la réponse de la plate-forme matérielle aux stimuli applicatifs (temps d'exécution) ;
- Critère 2 : l'approche doit offrir une méthode de modélisation comportementales de la partie applicative sans avoir accès au code source, et sans aller à l'encontre des contraintes de confidentialité. Elle doit également permettre d'extraire un ensemble de paramètres suffisamment représentatifs, afin de pouvoir générer un scénario de simulation du modèle comportemental précédent ;
- Critère 3 : l'approche doit permettre d'exécuter ce scénario sur la plate-forme d'exécution, afin de déterminer la compatibilité entre ses performances et les exigences qu'elle doit respecter.

Dans la suite, nous allons détailler comment notre méthode répond à chacun des points ci-dessus, au travers de sa mise en oeuvre sur deux cas d'études que nous présenterons dans la partie 7.

© 2012 TELECOM ParisTech

© 2012 TELECOM ParisTech

Chapitre 3

Etat de l'Art

Sommaire

3.1 Introduction 32
3.2 Langages de modélisation 33
3.2.1 Introduction 33
3.2.2 Langages synchrones et méthodes formelles 33
3.2.3 Langage de modélisation système 34
3.2.4 Langages de modélisation architecturale 34
3.2.5 Langages de description matérielle 36
3.2.6 Résumé 36
3.3 Méthodes de modélisation de plates-formes avioniques 37
3.3.1 Introduction 37
3.3.2 Modélisation d'une plate-forme globale 38
3.3.2.1 Modélisation architecturale 38
3.3.2.2 Modélisation physique 41
3.3.2.3 Modélisation mixte 41
3.3.3 Modélisation du réseau avionique 43
3.3.4 Modélisation des services ARINC653 pour les systèmes d'exploitation avioniques 44
3.3.5 Modélisation d'une application avionique 45
3.4 Langages et méthode de modélisation sélectionnés . . . 46
3.4.1 AADL 47
3.4.1.1 Composants 47
3.4.1.2 Interfaces, connections et déploiement 48
3.4.1.3 Extensions et outils 49

3.4.2 SystemC 50
3.4.2.1 Structure du langage 50
3.4.2.2 Bibliothèque TLM 51
3.4.2.3 Communications 52
3.4.2.4 Outils 54
3.5 SYNTHÈSE 54

3.1 Introduction

Contexte : un système embarqué ne se définit plus de nos jours comme un ensemble de composants, mais comme un ensemble de sous-systèmes, intégrant toujours plus d'éléments pour faire face au besoin croissant de puissance de calcul. Les plates-formes avioniques n'échappent pas à la règle et se définissent elles aussi comme un ensemble de sous-systèmes reliés entre eux, à savoir les calculateurs reliés par le réseau ARINC664. De plus, la rupture technologique induite par le changement de principe de conception de ces plates-formes, l'IMA, a nécessité le développement de nouvelles méthodes pour s'adapter aux contraintes toujours plus complexe de développement de ces architectures intégrées. Nous l'avons vu dans le chapitre 1, l'IMA autorise désormais un calculateur à exécuter plusieurs applications de niveaux de criticité différents, alors qu'auparavant chaque calculateur n'exécutait qu'une application.

Objectif : l'objectif de nos travaux est d'apporter une contribution au développement de ces nouvelles méthodes de conception. Plus particulièrement, nous nous sommes intéressés aux techniques de modélisation et d'exploration des performances d'un système embarqué en avance de phase, i.e. en tout début de cycle de développement. Ces méthodes permettent de modéliser l'architecture matérielle et/ou logicielle d'un système, avec plus ou moins de détails suivant le niveau d'abstraction choisi. S'appuyant notamment sur des langages de modélisation, l'objectif premier est de traduire les spécifications logicielles et matérielles en un modèle sur lequel il est possible d'effectuer diverses analyses pour la pré-validation de choix d'architecture et/ou le pré-dimensionnement du système en fonction de la partie applicative. Dans nos travaux, nous nous sommes appuyés sur ces outils et méthodes existants, que nous avons cherché à adapter aux spécificités de l'avionique.

 © 2012 TELECOM ParisTech

Organisation du chapitre : nous allons voir dans ce chapitre les méthodes de modélisation de systèmes embarqués temps-réel en avance de phase. Nous commencerons par un aperçu des langages de modélisation sur lesquelles celles-ci s'appuient. Nous verrons ensuite plus spécifiquement les travaux existants de modélisation de plate-forme avionique, et en quoi ils constituent une base qu'il a fallu enrichir pour s'adapter aux différentes problématiques liées à la modélisation en avance de phase.

3.2 Langages de modélisation

3.2.1 Introduction

Au delà des langages de programmation sont apparus dans les années 80 des langages de modélisation qui ont pour but de "fournir une représentation structurée d'un système logiciel" [Borde, 2009]. De plus en plus utilisés dans la conception de systèmes embarqués, ces langages offrent différents niveaux de description d'un système. Nous distinguerons par la suite 3 principaux niveaux d'abstraction : niveau système, niveau architectural et niveau comportemental ou description matérielle. Nous décrirons à part les langages synchrones destinés à faire de la vérification formelle.

3.2.2 Langages synchrones et méthodes formelles

Les langages synchrones sont à la base utilisés pour la programmation sûre de systèmes temps réels réactifs, i.e. qui interagissent en permanence avec leur environnement. Ils sont désormais utilisés pour la description de systèmes embarqués sous forme de modèles mathématiques simples, dont le but est l'application d'analyses formelles pour la vérification de propriétés. Parmi les langages synchrones, nous pouvons citer Esterel [Berry and Gonthier, 1992], SIGNAL ou encore Lustre [Halbwachs et al., 1996], très utilisé dans l'industrie aéronautique via le logiciel SCADE.
Ces langages permettent de modéliser une plate-forme avionique, notamment à l'aide du langage SIGNAL [Gamatié, 2004], [Gamatie and Gautier, 2002] ou encore [Gamatie et al., 2005], mais n'ont pas pour vocation première à décrire un système en vue de l'exploration de l'utilisation des ressources d'une plate-forme. Ils constituent toutefois une approche complémentaire dans l'optique de la validation du système, d'autant plus que certains projets visent à développer des passerelles entre des modèles décrits à l'aide d'un langage de description architecturale ou comportementale et des méthodes ou outils de vérification formelle comme UPPAAL. C'est le cas entre

© 2012 TELECOM ParisTech

une modélisation SystemC (que nous verrons plus loin) et UPPALL justement [Herbert et al., 2008], entre AADL (que nous verrons juste après) et le langage SIGNAL [Ma et al., 2007] ou encore AADL et UPPALL [Rolland, 2008]. Enfin, d'autres projets visent à intégrer la notion de contrat formel dans le langage SystemC [Bouhadiba et al., 2009], ou à appliquer des principes de vérification formelle sur des modèles architecturaux [Adballah et al., 2011].
Le projet [Kountouris and Guernic, 1991] permet, en s'appuyant sur l'environnement développé autour de SIGNAL, de pré-valider ou non en avance de phase un composant processeur (ASIC, ASIP) ou DSP, en vérifiant le respect de contraintes temporelles temps-réel. Ce genre d'approche s'avère intéressante pour notre objectif, sous réserve de posséder l'environnement SIGNAL et un minimum d'expérience. Or, lors du choix du/des langages et environnement à utiliser dans nos travaux, nous avons volontairement écarté les langages synchrones, principalement pour des raisons industrielles : d'une part, aucune expertise en langage synchrone et approche formelle n'est présente dans l'équipe, d'autre part les travaux doivent être intégrés à une chaîne de modélisation et simulation (critère 3 identifié dans le chapitre 2, ce qui a conditionné notre choix vers des approches à simulation.

3.2.3 Langage de modélisation système

Les langages de modélisation système décrivent ce dernier suivant les fonctionnalités (gestion de l'altitude, freinage, etc.) ou suivant les ressources logiques (FFT par exemple) qu'il doit offrir. Ce type de modélisation traduit les spécifications système en un modèle sur lequel peuvent être associées les exigences qui doivent être respectées. Nous pouvons citer UML [OMG, 2007] ou encore SysML [OMG, 2008] comme langages de description système. Les modèles exprimés ici ne permettent que peu d'analyses ; c'est pourquoi les langages sur lesquels ils sont basés ont été étendus afin d'intégrer plus d'informations. C'est notamment le cas de profils UML comme MARTE, que nous verrons un peu plus loin.

3.2.4 Langages de modélisation architecturale

Les langages de modélisation architecturale permettent de représenter l'architecture logicielle et matérielle du système, ainsi que le déploiement de la première partie sur la deuxième. Le plus souvent, ce sont des langages basés sur des composants représentant les ressources logicielles et matérielles, et caractérisés par des propriétés.

 © 2012 TELECOM ParisTech

Parmi les langages de description architecturale, nous avons poussé notre étude vers deux langages tournés vers la modélisation de systèmes avionique, AADL et le profil UML nommé MARTE, qui remplissent tous les deux le critère de sélection 1 évoqué dans le chapitre 2.

AADL est un ADL (Architecture Design Language), standardisé par la SAE [Aerospace, 2008]. Le langage est basé sur MetaH [Feiler et al., 2000], ADL dédié aux système avionique développé par Honeywell dans les années 90s. AADL a été développé pour la modélisation de systèmes avioniques d'abord, avant d'être étendu pour la description de tout type de systèmes embarqués temps-réel. AADL permet notamment une modélisation d'aspects non fonctionnels, comme les exigences temporelles ou le principe de partitionnement des architectures avioniques [Feiler et al., 2004].

MARTE [OMG, 2009] est une spécialisation du langage UML pour la description de systèmes embarqués temps réel [André, 2007]. Il offre la possibilité de modéliser les parties logicielles et matérielles d'un système via les bibliothèques SRM (Software Resource Modeling) et HRM (Hardware Resoucre Modeling). Il permet en outre de profiter d'outils existants pour UML et d'autres profils pour l'édition et l'analyse de modèles, en plus d'outils développés spécifiquement pour ce profil. Très proche d'AADL aussi bien dans la structure du langage -AADL a servi de base à sa standardisation-que dans les possibilités d'analyses, MARTE bénéficie d'une extension pour la description de systèmes ARINC653, basée elle aussi sur l'annexe AADL du même nom. Profitant de cette proximité, des travaux ont été développés afin de combiner les deux langages et tirer partie de leurs atouts respectifs, comme dans [Mallet et al., 2009] ou encore [Mallet and Simone, 2009].

Du fait de l'implication du laboratoire INFRES de Telecom ParisTech dans les travaux de développement du langage AADL et de ses annexes, nous avons opté pour ce dernier afin de profiter de l'accès aux sources du langage ainsi qu'aux outils de gestion de modèles AADL et d'analyses, comme l'outil OSATE. Nous détaillerons plus loin dans ce chapitre la structure du langage AADL.

© 2012 TELECOM ParisTech

3.2.5 Langages de description matérielle

Les langages de description matérielle sont apparus pour modéliser de manière très fine le comportement d'un composant électronique numérique ou analogique, à des fins de test voire de génération du schéma en portes logiques. Par opposition aux précédents langages ADL, ceux-ci sont appelés HDL (Hardware Design Language). Les langages HDL les plus connus sont VHDL [IEEE-Computer-Society, 2000] et Verilog [IEEE-Computer-Society, 2001] (ainsi que son extension SystemVerilog [Accelera, 2001]), les deux étant très proches. Ils permettent tous les deux de déterminer par simulation la validité du comportement d'un composant sans avoir à fondre le composant en lui-même, permettant ainsi de faciliter et d'accélérer le processus de développement. Cependant, ces langages demandent une description très précise des composants matériels, ce qui nécessite un certain temps et un effort de développement, allant à l'encontre du critère 1 que nous avons vu dans le chapitre 2.

Un autre langage de description matérielle s'impose de plus dans l'industrie, SystemC, standard IEEE [IEEE-Computer-Society, 2006]. A l'origine langage de description matérielle comme VHDL ou Verilog, SystemC a été étendu afin de couvrir plusieurs niveaux d'abstraction, et assurer une continuité dans la modélisation d'un système. Parmi ces niveaux, TLM (Transaction Level Modeling), permet de modéliser un système à un niveau comportemental intermédiaire, plus précis qu'une modélisation architecturale mais moins précis qu'une description de type VHDL. Ce niveau offre un bon compromis entre temps de modélisation et temps de simulation. Il remplit ainsi les critères 1 et 3 identifiés dans le chapitre 2, ce qui nous a amené à le sélectionner. Nous allons par conséquent décrire un peu plus en détail le langage un peu plus loin dans ce chapitre.

3.2.6 Résumé

Nous venons de voir que les langages de modélisation se sont développés afin de couvrir plusieurs niveaux d'abstraction répondant à des besoins différents. Ceci est rappelé sur la figure 3.1. Certains d'entre eux ont développé des extensions afin de cibler les plates-formes avioniques (AADL, MARTE), tandis que d'autres se sont spécialisés dans l'analyse comportementale (VHDL, Verilog) et/ou l'exploration de performances de systèmes temps-réel en avance de phase (SystemC).

© 2012 TELECOM ParisTech

Dans notre objectif de modélisation de plate-forme avionique en avance de phase et d'exploration de ses performances, nous pouvons déjà écarter plusieurs de ces langages. Le niveau de modélisation système (UML, SysML) ne convient pas à l'analyse de performances, et ne permet pas la prise en compte de propriétés non fonctionnelles comme les exigences temporelles. A l'opposé, les langages HDL de type VHDL ne sont pas non plus adaptés à la modélisation en avance de phase pour deux raisons. D'une part, l'effort de modélisation à fournir pour la description d'un composant et le temps de simulation peuvent être relativement longs. D'autre part, le manque de détails lorsque l'on travaille en avance de phase va à l'encontre de ce niveau d'abstraction qui modélise justement les composants avec un niveau de précision poussé.

Niveau de modélisation	Exemple de langages
système, fonctionnelle, logique	UML, SysML
architecturale	AADL, MARTE
comportementale	SystemC-TLM
physique	SystemC-CABA/RTL, VHDL, Verilog
synchrone	Lustre, Esterel, SIGNAL

FIGURE 3.1 – Aperçu des langages de modélisation de systèmes embarqués

Nous allons voir dans la section suivante différents travaux ayant porté sur la modélisation et l'analyse de plates-formes avioniques ou partie de plates-formes, notamment le réseau ARINC664.

3.3 Méthodes de modélisation de plates-formes avioniques

3.3.1 Introduction

S'appuyant sur ces langages, des méthodes de modélisation et d'analyses ont été développées pour décrire tout ou partie d'une plate-forme avionique, à différents ni-

© 2012 TELECOM ParisTech

veaux d'abstraction : depuis la description des exigences au niveau système, jusqu'à celle du niveau description physique, en passant par le niveau architectural, chaque niveau permettant différentes analyses. Nous allons voir dans la sous-section suivante plusieurs travaux de modélisation d'une plate-forme globale, avant de décrire certains travaux de modélisation de réseau avionique. Ces derniers visent à vérifier que les performances temporelles du réseau qui relie les calculateurs sont conformes aux exigences, ce qui revient à vérifier que les latences maximales de transfert de trames d'un calculateur à un autre sont inférieures à la borne supérieure fixée dans les exigences.

3.3.2 Modélisation d'une plate-forme globale

Comme pour les langages de modélisation, nous pouvons distinguer différents niveaux de modélisation pour différents types d'analyses. Nous avons identifié lors de la description des différents langages de modélisation que le niveau de description système de type SysML n'est pas approprié à nos objectifs. De même, nous avons vu que les modélisations de systèmes avioniques à l'aide de langages synchrones étaient utiles pour la vérification de propriétés [Brunette et al., 2005] comme les exigences de sûreté [Sagaspe, 2008]. Toutefois, nous avons privilégié l'approche par simulation ; nous laisserons donc de côté les approches synchrones.

3.3.2.1 Modélisation architecturale

Le premier type de modélisation auquel nous nous sommes intéressés concerne donc une description de niveau architectural. A ce niveau de description, le but est de constituer un modèle du système où les éléments logiciels et matériels sont connus dans leur ensemble (nom, type, propriétés), mais où les informations plus précises (comportement détaillé, description de chaque signal d'une trame, etc.) ne sont pas forcément connues (modélisation en avance de phase) ou requise. Par exemple, dans les travaux [Singhoff et al., 2004], l'objectif est de vérifier que la configuration logicielle définie (ensemble de processus et de tâches déployées sur un processeur) est ordonnaçable dans le respect des contraintes temporelles.

Certains travaux comme [Senn et al., 2008] visent à estimer rapidement la consommation électrique d'un modèle de plate-forme d'exécution avant que celle-ci ne soit développée. Pour cela, les auteurs ont étendu les propriétés d'AADL afin d'ajouter une

 © 2012 TELECOM ParisTech

bibliothèque contenant des modèles de composants matériels (PowerPC 405, ARM7, etc) avec leurs règles de consommation en utilisation. Une première phase d'estimation gros grain de la consommation est menée sur le modèle AADL en fonction de la configuration logicielle (ordonnancement et propriétés des tâches logicielles), et suivant le déploiement sur les éléments matériels. Ces premiers tests conduits donnent des estimations à 30% de précision. Afin d'augmenter la précision, les auteurs passent par la méthode développée dans le projet Spices que l'on verra plus loin. Un modèle SystemC de la plate-forme matérielle est généré, et le code applicatif est parcouru afin d'en extraire les stimuli. Les modèles SystemC, réagissent ensuite à ces stimuli suivant des règles de consommation associées à chacun d'entre eux. Si la méthode semble intéressante et à approfondir pour l'exploration de performances (pas seulement énergétique comme ici), quelques points de discussion subsistent : i) il n'y a aucune indication sur la partie génération des stimuli pour le modèle SystemC ; ii) cette partie nécessite de posséder le code source des applications, or nous verrons que dans le domaine avionique les applications ne sont pas forcément disponibles pour le développeur de la plate-forme d'exécution pour cause de confidentialité ; iii) le processus ne prend pas en compte les spécificités et contraintes avioniques et nécessiterait d'être étendu pour cela.

D'autres travaux visent à apporter une simulation à haut niveau d'abstraction du modèle de plate-forme. Par exemple, dans [Casteres and Ramaherirariny", 2009], les auteurs ont pour but de développer un simulateur pour plate-forme d'exécution avionique. Cette dernière est décrite à l'aide des composants matériels d'AADL, tandis que la partie applicative, consistant ici en un modèle de simulation, est décrite avec les composants logiciels d'AADL. Afin d'effectuer leur simulation, les auteurs se sont basés sur le paradigme producteur - consommateur : chaque tâche est décrite comme un ensemble de requêtes CPU, accès mémoire et accès I/O, qui sont envoyées au processeur principal. Celui-ci relaie ensuite les trames vers le composant cible (mémoire si requête mémoire en lecture / écriture, etc.), qui traite la requête dans un temps défini par les propriétés AADL du modèle (read_latency pour un accès mémoire par exemple). Ainsi, les auteurs obtiennent un diagramme des taux d'utilisation en terme de CPU, I/O et accès mémoire pour le modèle de simulation donné en entrée avec une précision de 15% selon leurs tests.
Les principaux points de discussion ici concernent la modélisation de la partie applicative et le mécanisme de simulation des modèles. Pour le premier point, il est là aussi nécessaire d'avoir le code source de l'application afin de le traduire, d'une manière

© 2012 TELECOM ParisTech

qui n'est pas mentionnée, en requêtes CPU, I/O ou mémoire. D'autre part, le mécanisme de simulation basé sur le principe producteur - consommateur (ou initiateur - cible selon la littérature), est le même que dans certaines méthodes de modélisation s'appuyant sur des noyaux de simulation existants, ce qui autorise l'utilisation de simulateurs existants.

 © 2012 TELECOM ParisTech

3.3.2.2 Modélisation physique

Les travaux de modélisation de niveau physique ou comportemental permettent d'explorer les performances d'un système temps-réel avec plus de précision que les travaux de niveau de modélisation architecturale. Toutefois, nous avons vu que les modélisations de niveau physique nécessitaient un niveau de détail dans la description des composants matériels beaucoup plus élevé, ce qui demande un effort de modélisation relativement grand qui ne correspond pas à l'exigence d'exploration rapide des performances en avance de phase. De plus, notre objectif est d'explorer la consommation des ressources d'une plate-forme, et non de voir ce qui se passe au bit près de telle trame. Les méthodes de modélisation physique se rapprochent souvent de la virtualisation, en exécutant le code source sur le modèle de plate-forme d'exécution [Buchmann, 2006], ce qui n'est pas notre objectif Or, nous avons vu précédemment que dans le domaine avionique, une application n'était pas toujours accessible pour raisons de confidentialité.

3.3.2.3 Modélisation mixte

Une modélisation de niveau architectural est utile pour décrire un système et faire des premières analyses en avance de phase, mais ne permet pas une analyse précise des performances temporelles. Partant de ce constat, plusieurs travaux ont été lancés afin de coupler ces modélisations architecturales avec un niveau d'abstraction moins élevé de type comportemental. Le langage SystemC est généralement choisi pour son niveau d'abstraction TLM permettant une modélisation intermédiaire entre le niveau architectural et le niveau physique.

Ainsi, dans le projet MARTES, l'idée est de coupler UML avec SystemC afin de profiter des deux niveaux d'abstraction, système et comportemental, et surtout utiliser le noyau de simulation de SystemC. Une première étape consiste à réaliser de manière séparée une modélisation fonctionnelle du système et un modèle architectural de la plate-forme d'exécution. Les deux étant ensuite intégrés dans un modèle réalisé en profil UML MARTE, avant qu'une transformation de modèle vers SystemC TLM ne soit opérée. Il existe d'autres projets similaires, dont Spices, qui définit un processus semblable au précédent en utilisant le langage AADL à la place d'UML MARTE. Ces projets sont intéressants dans leur volonté de coupler un langage de description haut niveau avec un langage de niveau d'abstraction moins élevé permettant de simuler les

© 2012 TELECOM ParisTech

modèles pour explorer les performances d'une plate-forme. La figure 3.2 présente le processus de modélisation d'un système en couplant une modélisation architecturale et une modélisation comportementale.

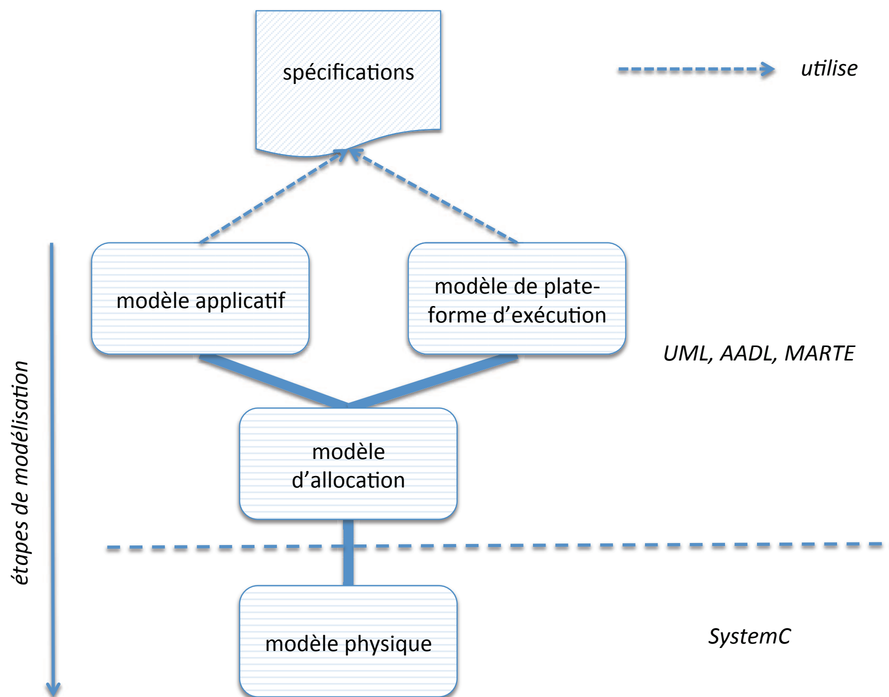

FIGURE 3.2 – Processus de modélisation mixte

Plusieurs points de réflexions ont émergés lors de leur étude. D'une part, les plates-formes ciblées n'étaient pas les mêmes que dans notre cas, les projets se concentrant surtout sur les plates-formes mobiles. Sachant que la conception de plates-formes avioniques est soumise à des contraintes spécifiques, il est nécessaire d'adapter ou d'enrichir le processus. D'autres part, il apparait à la vue des publications et travaux que ces derniers se sont surtout concentrés sur la partie modélisation architecturale AADL, MARTE voire CCM. Enfin, le processus de modélisation architecturale et de transformation vers SystemC de ces travaux demande beaucoup d'informations sur les parties applicatives et matérielles qui ne sont pas toujours disponibles en avance de phase.

 © 2012 TELECOM ParisTech

3.3.3 Modélisation du réseau avionique

Les travaux précédents visent à modéliser une plate-forme embarquée, avionique ou non suivant les projets, dans son ensemble à mener de premières analyses. Or, dans une plate-forme avionique IMA, les applications sont réparties sur plusieurs calculateurs reliés entre eux par le réseau ARINC664. Les latences de communication au sein du réseau peuvent varier selon le nombre de trames présentes à un instant donné, ce qui peut avoir un impact sur les performances temporelles de la plate-forme. Plusieurs travaux ont été développés afin de modéliser spécifiquement le réseau d'une plate-forme, et pratiquer certaines analyses comme le respect de la borne supérieure du temps de transmission d'une trame, ou encore la validité de la topologie du réseau. Pour rappel, nous avons vu dans le chapitre précédent qu'un réseau avionique se définit suivant son architecture logique, constitué de plusieurs chemins logiques par lesquels transitent les messages.

Dans l'optique de déterminer les performances temporelles du réseau et vérifier que les temps de transmission respectent les exigences de latence maximum, plusieurs méthodes ont été utilisées. La première consiste en une modélisation de type "network calculus", méthode visant à déterminer les bornes de latences de transmission : pour chaque chemin logique, la méthode détermine une borne supérieure de latence de transfert de bout en bout, comme dans [Grieu, 2004]. Les latences calculées sont très pessimistes par rapport à la réalité, ce qui fait que cette méthode est surtout utilisée dans les calculs de performances temporelles pire cas (WCET).
La seconde méthode consiste en une approche de type "model checking". Elle détermine par calculs mathématiques que les contraintes temporelles sont respectées. Bien que théoriquement plus précise que la précédente [Ermont et al., 2006], l'application à un réseau avionique réel est impossible car elle engendre un nombre de calculs trop importants, i.e. une explosion combinatoire.
Ainsi, afin d'affiner les méthodes d'exploration de performances de plates-formes avioniques vues dans les sections précédentes, il semble important d'enrichir celles-ci en affinant la partie modélisation du réseau avionique. Notamment, une solution peut consister en le couplage d'une méthode de modélisation existante et d'une modélisation spécifique du réseau comme dans [Charara, 2007] afin, dans un premier temps, d'obtenir les latences de transmission maximales au sein du réseau. Ces valeurs serviraient ensuite dans les analyses de performance temporelle de la plate-forme globale.

© 2012 TELECOM ParisTech

3.3.4 Modélisation des services ARINC653 pour les systèmes d'exploitation avioniques

Une autre partie des plates-formes avioniques est souvent soit négligée, soit représentée de manière très succincte dans les méthodes de modélisation architecturales : il s'agit de la partie système d'exploitation. Rappelons que dans le domaine avionique, les systèmes d'exploitation doivent être conformes au standard ARINC653 et fournir les services définis dans celui-ci (on parle de l'APEX, l'API des services ARINC). Toutefois, l'implémentation de ces services reste propre à chaque plate-formiste. Nous avons donc l'architecture décrite sur la figure 3.3 : des partitions au sein desquelles les tâches logicielles font appel aux services de l'APEX. Ces appels sont ensuite traités par le système d'exploitation, suivant l'implémentation qui est faite des services, avant d'être envoyés vers la partie matérielle.

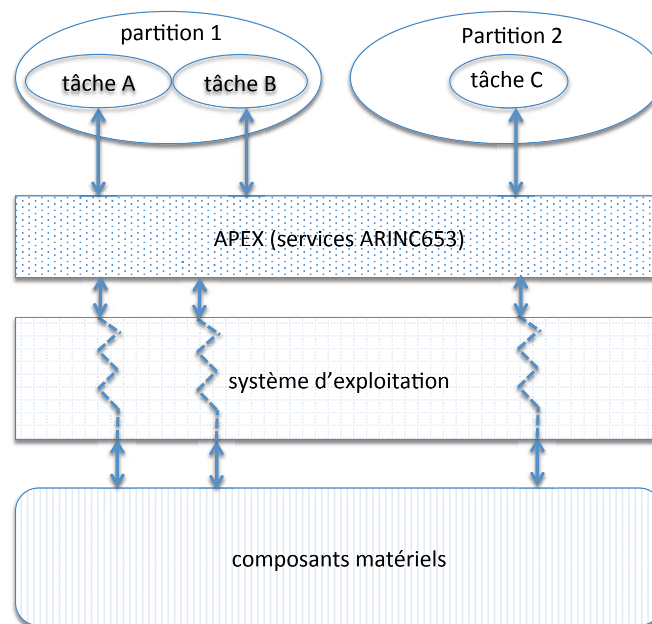

FIGURE 3.3 – Architecture d'un système avionique décrite par le standard ARINC653

Il y a ici deux enjeux : le premier consiste à modéliser les partitions et les tâches du système avec leurs propriétés ainsi que les politiques d'ordonnancement : ordonnancement fixe pour les partitions et ordonnancement classiques de type RMS pour les tâches. Cette modélisation a pour but de vérifier l'ordonnançabilité du système. Plusieurs travaux se sont intéressés à cette partie, comme [la Camara et al., 2007] ou [Easwaran et al., 2009].

© 2012 TELECOM ParisTech

L'autre enjeu consiste en la modélisation du comportement du système d'exploitation lors de l'appel à un service. Cette partie a été beaucoup moins adressée, car la modélisation d'un système d'exploitation est une opération coûteuse en terme d'effort et de temps de modélisation. De plus, comme il a été évoqué, chaque plate-formiste dispose de sa propre implémentation du standard ARINC653, ce qui rend difficile toute modélisation "générique" d'un système d'exploitation. Il est toutefois possible de s'appuyer sur deux éléments : le premier est que le standard ARINC653 impose pour chaque service un comportement global, que l'on peut traduire en macro-blocs algorithmiques. Pour exemple, le service "get_sampling_port_id" qui récupère l'identifiant du port en question et renvoie un code de retour OK ou ERROR suivant l'exécution. Le second élément tient au fait que le système d'exploitation étant propre au plate-formiste, plusieurs études de performances temporelles ont été menées sur différentes architectures matérielles, donnant autant d'informations temporelles sur l'exécution de chaque service. Une possibilité pour caractériser les temps de latences au sein du système d'exploitation serait donc d'extrapoler ces données pour les intégrer dans le modèle de plate-forme globale.

3.3.5 Modélisation d'une application avionique

Nous avons vu dans l'état de l'art que le développement d'une plate-forme avionique se faisait en parallèle par plusieurs acteurs : la partie plate-forme d'exécution, i.e. l'aspect matériel et le système d'exploitation, sont développés par le plate-formiste, tandis que les applications sont développées généralement par les "function suppliers". A ce titre, le code source des applications n'est pas toujours disponible pour des raisons de confidentialité, ce qui peut rendre difficile la modélisation de la plate-forme dans son ensemble en avance de phase. Ainsi il est nécessaire de développer un moyen permettant aux développeurs de ces applications de fournir un modèle de celles-ci sans pour autant dévoiler trop d'informations sur leur code.

L'application peut être modélisée à plusieurs niveaux d'abstraction suivant les besoins. L'application peut être modélisée suivant sa décomposition en partitions et tâches, avec leurs propriétés. Cette modélisation nous renvoie aux travaux précédents sur les différentes analyses d'ordonnançabilité de la configuration logicielle au sein de la plate-forme, que ce soit avec le profil UML MAST pour la modélisation d'applications temps-réel [Medina et al., 2001] ou l'outil Cheddar associé au langage AADL [Singhoff et al., 2004]. Si l'on souhaite effectuer des analyses de performances (temporelles,

© 2012 TELECOM ParisTech

consommation d'énergie, etc.), il est alors nécessaire d'avoir une modélisation plus fine de l'application. Nous l'avons vu, les méthodes de modélisation actuelles considèrent très généralement que le code applicatif est accessible et appliquent des outils de parcours de code pour en extraire les informations souhaitées, voire même mener des premières estimations, comme des analyses de consommation [Laurent, 2002].

Ainsi, le problème spécifique du besoin d'une modélisation de l'applicatif plus précise que dans les modèles logiques ou architecturaux, sans avoir accès aux applications elles-même, n'est pas réellement étudié dans la littérature car assez spécifique à des domaines critiques faisant intervenir de multiples acteurs, comme dans l'avionique. Une piste de réflexion est toutefois d'adapter les méthodes de modélisation d'application existantes, de les enrichir si besoin, et de se servir des modèles définis comme d'un modèle intermédiaire. Par exemple, l'ajout d'une description comportementale d'une tâche logicielle au sein d'un modèle architectural comme AADL, en utilisant l'annexe comportementale [Dissaux et al., 2006], peut être une solution envisageable.

3.4 Langages et méthode de modélisation sélectionnés

Il est ressorti de notre étude qu'une méthode de modélisation mixte est la plus adaptée à notre problématique. S'appuyer sur un langage de modélisation architecturale pour la prise en compte des spécifications de la plate-forme avionique globale, et sur un langage de description comportementale de la plate-forme d'exécution pour l'analyse de ses performances, permet d'effectuer des analyses à deux niveaux d'abstraction, et de tirer parti de chacun d'eux. Nous avons dans cette optique sélectionné le langage AADL pour la description architecturale, et le langage SystemC pour la description comportementale.

© 2012 TELECOM ParisTech

3.4.1 AADL

AADL, Architecture Analysis and Design Language, est un langage de modélisation architecturale orienté composants. Comme le montre la figure 3.4, il permet à l'utilisateur de décrire les composants logiciels et matériels d'un système.

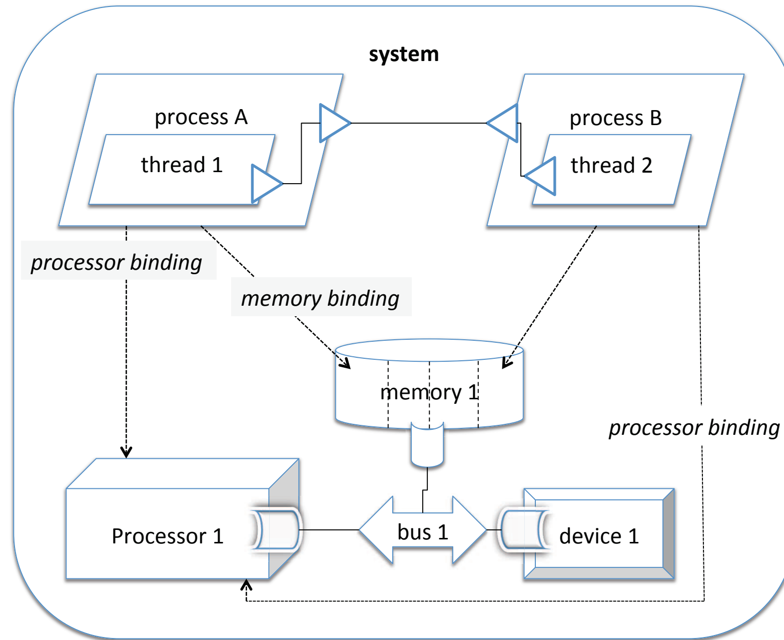

FIGURE 3.4 – Exemple de modélisation architecturale à l'aide des composants AADL

3.4.1.1 Composants

Chaque composant est caractérisé par un ensemble de caractéristiques dimensionnantes (nommées propriétés), comme la latence de lecture et d'écriture, la taille ou la largeur des données d'une mémoire par exemple. Chaque propriété est typée de manière à ce qu'il n'y ait aucune ambiguïté (la taille mémoire est de type *size*, les latences sont de type *time*, etc.). Pour représenter un système, AADL définit les composants logiciels suivants :

- Le composant **process** qui représente un processus dans lequel s'exécutent une ou plusieurs tâches. Dans le contexte avionique, le composant représente une partition avionique ;
- Le composant **thread** qui modélise l'une de ces tâches. Dans le contexte avionique, il modélise un processus, au sens ARINC653 du terme, qui s'exécute au sein de la partition ;

© 2012 TELECOM ParisTech

- Le composant **subprogram** qui représente une séquence de code associée à une tâche ;
- Le composant **data** qui représente une donnée ou un emplacement mémoire.

AADL définit également les composants matériels suivants :

- le composant **processor** qui représente une ressource de calcul principale, et associe quelques propriétés pour la définition du système d'exploitation qu'il embarque ;
- Le composant **virtual processor** qui représente une ressource logique associée à un processeur (ordonnanceur, partition, etc.) ;
- Le composant **memory** qui représente un composant de stockage (mémoire DDR, ROM, disque dur, etc.) ;
- Le composant **bus** qui représente un accès entre composants matériels.
- Le composant **virtual bus** qui représente généralement le protocole du bus physique auquel il est associé.

Enfin, AADL définit le composant **system** qui permet la définition du système en lui-même via la connexion des éléments et le déploiement des composants logiciels sur les composants matériels. Le langage précise les règles de composition de ces composants. Par exemple, un composant *process* peut contenir un ou des composants *thread*, mais ne peut pas contenir de composant *memory* par exemple.

3.4.1.2 Interfaces, connections et déploiement

La plupart des composants AADL peuvent se voir associés une interface, permettant leur connexion avec d'autres composants. Les connexions peuvent être physiques pour les éléments matériels (connexion d'un processeur avec un bus, lui-même connecté à une mémoire), ou logiques pour les échanges de données entre éléments logiciels (connexion d'un processus avec un autre). Ces échanges se font via des ports, sur lesquels des signaux, représentés par le type *event* sont envoyés.

Lors de la composition des éléments au sein du composant *system*, il est possible de décrire le déploiement des ressources logicielles sur les ressources matérielles. Par exemple, il est possible d'associer un processus avec un processeur et une mémoire ou zone mémoire (opération de "binding" sur la figure 3.4 précédente).

© 2012 TELECOM ParisTech

L'exemple qui suit nous donne un aperçu de l'implémentation d'un système en AADL, avec les parties instantiation des composants, connexions et déploiement :

```
system implementation syst.impl

subcomponents
   cpu1   : processor cpu.impl;
   bus1  : bus mbus.impl ;
   ram1  : memory mem.impl ;
   part1  : process m_process.impl1;
   part2  : process m_process.impl2;

connections
   bus access bus1 <-> cpu1.bus_connector ;
   bus access bus1 <-> ram1.bus_connector;

properties
  Actual_Processor_Binding => (reference (cpu1.part1)) applies to part1;
  Actual_Processor_Binding => (reference (cpu1.part2)) applies to part2;

  Actual_Memory_Binding => (reference(ram1.part1)) applies to part1;
  Actual_Memory_Binding => (reference(ram1.part1)) applies to part2;

end syst.impl;
```

3.4.1.3 Extensions et outils

Des extensions au langage ont été développées, comme "l'annexe de modélisation des erreurs" pour modéliser les aspects de sûreté de fonctionnement [Rugina, 2007], l'annexe ARINC653 pour la modélisation de systèmes IMA [Delange et al.,], ou encore l'annexe comportementale [Dissaux et al., 2006], [Lasnier et al., 2010], en développement au début de ma thèse.

© 2012 TELECOM ParisTech

Autour du langage et de ses extensions, de nombreux outils ont été développés pour effectuer des premières analyses statiques sur les modèles créés, dont la présentation [Hughes and Singhoff,] nous donne un bon aperçu. Nous citerons par exemple l'outil Ocarina pour la génération d'applications temps-réel réparties embarquées [Zalila et al., 2009] ou encore l'outil OSATE, basé sur éclipse, pour la gestion de modèles AADL et de ses annexes.

3.4.2 SystemC

3.4.2.1 Structure du langage

SystemC est un ensemble de classes C++ pour la description et la simulation de composants matériels. Contrairement à VHDL ou Verilog, il permet une modélisation d'un composant ou d'une plate-forme matérielle à plusieurs niveaux d'abstractions que nous avons vu précédemment dans le chapitre. S'appuyant sur le langage C++, il est possible d'ajouter à une description matérielle des fonctions logicielles, le rendant particulièrement utile pour la modélisation de SoC (System on Chip). Comme nous le voyons sur la figure 3.5, SystemC est constitué de briques de bases comportant :

- Les modules, l'enveloppe de base d'un composant ;
- Les ports, associés à un module, qui constituent les points d'entrée-sortie de ce dernier ;
- Les interfaces, qui contiennent les fonctions qui permettent la communication via les ports ;
- Les canaux, moyens de communication entre les modules et branchés au module via un port ;
- Les processus, qui décrivent les fonctionnalités ou comportement d'un module.

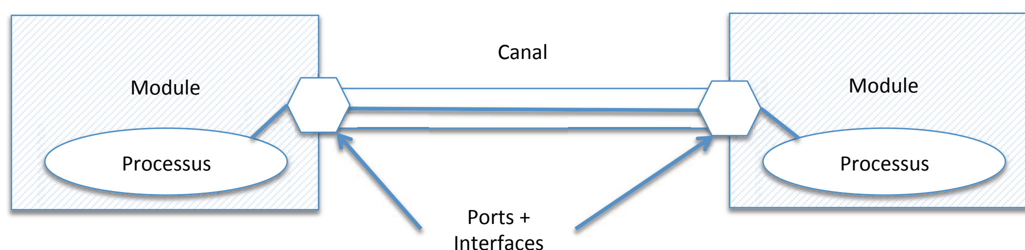

FIGURE 3.5 – Structure d'un système modélisé en SystemC

© 2012 TELECOM ParisTech

Il existe principalement quatre niveaux de modélisation, même si chacun peut également posséder des sous-niveaux :

- Le niveau système pour la représentation des spécifications ;
- Le niveau TLM (Transaction Level Modeling), permet une modélisation à un niveau comportemental intermédiaire. Ce niveau est plus précis qu'une modélisation architecturale [Cai and Gajski, 2008], mais moins précis qu'une modélisation VHDL par exemple. C'est actuellement le niveau le plus utilisé dans l'industrie et un standard de facto, nous y reviendrons par la suite ;
- Le niveau CABA (Cycle Accurate Bit Accurate), permet une modélisation comportementale précise au signal près d'un composant ;
- Enfin, le niveau RTL (Register Transaction Level), est le niveau le plus ”fin” de modélisation, au niveau portes logiques, proche des langages VHDL ou Verilog.

Chaque niveau de modélisation correspond à un besoin particulier en terme de précision de modélisation d'analyses à effectuer. Si la précision augmente en allant du niveau fonctionnel vers le niveau portes logiques, l'effort de modélisation et le temps de simulation augmentent eux aussi, même si des travaux visent à accélérer les temps de simulation au niveau CABA ou RTL comme [Buchmann, 2006]. SystemC possède un noyau de simulation événementiel, pour la simulation des modèles développés, ce qui en fait un langage de modélisation relativement complet.

3.4.2.2 Bibliothèque TLM

Le niveau d'abstraction TLM s'impose de plus en plus pour la modélisation et l'exploration des performances d'un système temps-réel, car il constitue un bon compromis entre précision, effort de modélisation et temps de simulation [Ghenassia, 2005]. Ce niveau décrit les échanges de trame entre composants maîtres et composants esclaves, typiquement un processeur et une mémoire. L'important ici est donc le comportement global de la plate-forme et l'utilisation de ses composants, plus que le message envoyé lui-même. L'utilisation de SystemC-TLM est ainsi privilégiée dans le cas d'étude de performances d'une plate-forme en début de cycle de développement. Le niveau TLM permet en outre d'obtenir des temps de simulation qui sont de 100 à 1000 fois plus rapides que les simulations de niveau CABA ou RTL.

© 2012 TELECOM ParisTech

3.4.2.3 Communications

Les communications au sein de la bibliothèque TLM se font via des *interfaces*, qui regroupent un ensemble de fonctions pour la communication. La notion de port et de canal vu précédemment est ici abstraite et remplacée par la notion de *socket*. Lorsqu'un composant veut communiquer avec un autre composant, il envoie un message sur la *socket* correspondante via l'une des fonctions de *l'interface* associée. Le message est ensuite envoyé vers le composant cible, et plus précisément vers une méthode spécifique qui peut être généralement *b_transport* (transmission bloquante) ou *nb_transport* (transmission non bloquante). Lorsque la fonction reçoit le message elle le décode, le traite, et renvoie une réponse (traitement ok, erreur, etc.). Lorsque le message est envoyé via une *socket*, il n'est pas envoyé tel quel mais inséré dans une trame, la *generic_payload*. Cette trame contient les informations utiles de la trame :

- La commande en elle-même (lecture, écriture, aucune) ;
- L'adresse où lire/écrire la donnée ;
- Un pointeur vers la donnée à lire/écrire ;
- La longueur de la donnée ;
- L'état de la réponse (ok, erreur, mauvaise adresse, etc.).

En plus de ces informations, la trame contient un champs *delay*, qui spécifie le temps d'acheminement de la trame vers le composant destinataire. Le traitement de la trame par le composant cible se fait via un automate. Suivant les éléments de la trame (commande, adresse, etc.), le composant traitera de manière différente la commande qui lui est envoyée. La figure 3.6 résume de manière simplifiée l'opération d'émission, traitement et réponse d'une trame : 1) le composant source envoie une commande en l'encapsulant dans une trame. La trame est ensuite envoyée sur la *socket* correspondante ; 2) la trame est relayée au composant destinataire ; 3) l'automate du composant cible décode la trame et traite la commande ; 4) le composant cible envoie la réponse correspondante au composant source.

© 2012 TELECOM ParisTech

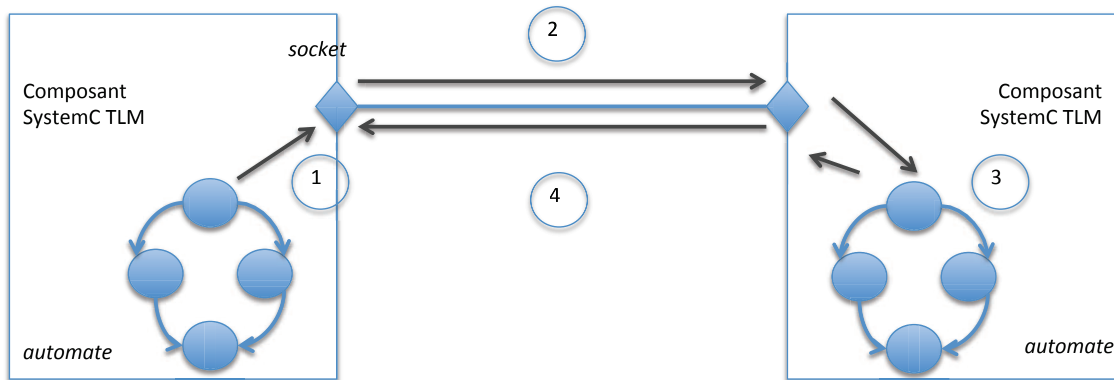

FIGURE 3.6 – Traitement d'une trame TLM

Pour résumer, une plate-forme matérielle décrite en SystemC peut se voir comme un ensemble d'automates communicants via des *interfaces* liées à des *sockets*. Cette notion d'automates permet d'introduire la notion de variabilité dans le comportement d'un composant suivant la commande reçue et l'état en cours du composant. Nous donnons ci-dessous un exemple de composant mémoire SystemC-TLM : en fonction de la commande reçue (lecture ou écriture), le composant applique un comportement spécifique (ici la mémoire exprime le temps de lecture ou d'écriture de l'opération en question).

```
switch(c->get_command())
    {
      //TLM_READ_COMMAND
    case READ:
      {
        delay += sc_time(m_read_time,SC_NS);
        gpayload->set_response_status(tlm::TLM_OK_RESPONSE);
        break;
      }

      //TLM_WRITE_COMMAND
    case WRITE:
      {
        delay += sc_time(m_write_time,SC_NS);
        gpayload->set_response_status(tlm::TLM_OK_RESPONSE);
        break;
      }
```

© 2012 TELECOM ParisTech

```
    default:
      {
        //traitement spécifique ou non du cas
        ....
        break;
      }
  }

return;
```

3.4.2.4 Outils

SystemC étant très présent dans le monde industriel pour la modélisation de plates-formes matérielles, surtout via la bibliothèque TLM, de nombreux outils ont été développés ou étendus afin de prendre en compte le langage. Parmi ceux-ci, il est possible de citer les outils de modélisation ModelSim et NCSim côté industriel. Côté académique, le front-end open-source PINAPA permet de faire de la vérification sur des modèles TLM. Cet outil est intégré au sein du projet GreenSocs qui regroupe tout un ensemble d'outils pour l'implémentation et la vérification de plates-formes SystemC. Il existe par ailleurs des bibliothèques de composants, TLM, CABA ou RTL, telle Soclib de l'Université Paris 6, développées afin de fournir une base de composants directement utilisables pour la composition de plates-formes. Cette liste est loin d'être exhaustive.

3.5 Synthèse

Nous venons de voir qu'il existe des méthodes de modélisation à plusieurs niveaux d'abstraction, permettant différents types d'analyses. Celles-ci s'appuient sur des langages de modélisation qui ont évolués et se sont enrichis afin de pouvoir modéliser des systèmes embarqués temps-réel.

En ce qui concerne plus particulièrement notre objectif de modélisation de plateforme avionique en avance de phase et d'exploration de performances, nous avons vu que des méthodes de modélisation architecturales, notamment AADL, avaient développé des extensions ou annexes afin de décrire les systèmes avioniques. Ces mé-

© 2012 TELECOM ParisTech

thodes autorisent des analyses statiques comme l'ordonnancement des partitions ou une exploration gros grain de la consommation du système, mais restent limitées pour des analyses plus poussées de type consommation des ressources matérielles. Pour tenter de répondre à cette limitation, certains travaux se sont penchés sur le couplage de méthodes de modélisation architecturale avec une modélisation de type comportementale, généralement à l'aide du langage SystemC et de son noyau de simulation. Toutefois, cette méthode doit être adaptée et enrichie sur certains points.

D'autre part, la méthode doit être adaptée au domaine de l'avionique au niveau de la génération de la plate-forme d'exécution SystemC, et prendre en compte les contraintes liées à la conception de plates-formes partitionnées. Il est également nécessaire d'affiner la modélisation de certains éléments de la plate-forme. Par exemple, la modélisation du réseau avionique au sein du modèle architectural AADL ne permet pas en l'état d'exprimer les latences de transmission sur les différents chemins logiques. Or, plusieurs travaux ont proposé une modélisation spécifique du réseau pour permettre de déterminer ces latences. D'autres proposent d'utiliser la possibilité d'étendre les jeux de propriétés d'AADL pour effectuer une modélisation du réseau logique, c'est-à-dire l'ensemble des chemins logiques. Un couplage de ces méthodes permettrait de gagner en précision pour l'exploration de performances. Il en va de même pour la modélisation de la partie système d'exploitation et la partie applicative, qui doivent être affinées.

Dans la partie suivante, nous allons préciser la problématique de nos travaux. Nous verrons les différents points techniques à résoudre pour notre objectif d'exploration des performances d'une plate-forme avionique en avance de phase.

© 2012 TELECOM ParisTech

© 2012 TELECOM ParisTech

Troisième partie

Approche adoptee

Chapitre 4

Methodologie

SOMMAIRE

4.1 RAPPEL DES OBJECTIFS 60
4.2 STRUCTURE GLOBALE 62
4.3 CAPTURE DES PARAMÈTRES DE LA PARTIE APPLICATIVE ET GÉNÉRATION DES STIMULI 63
4.3.1 Objectifs 63
4.3.2 Extraction de la configuration logicielle 63
4.3.3 Génération des stimuli applicatifs 64
4.3.4 Résumé 66
4.4 MODÉLISATION ARCHITECTURALE DE LA PLATE-FORME AVIONIQUE 67
4.4.1 Objectifs 67
4.4.2 Démarche 67
4.4.3 Introduction des spécificités avioniques 68
4.4.4 Ajout de propriétés 70
4.4.5 Résumé 70
4.5 GÉNÉRATION D'UN MODÈLE COMPORTEMENTAL DE LA PLATE-FORME D'EXÉCUTION 70
4.5.1 Objectifs 70
4.5.2 Génération des composants comportementaux 71
4.5.3 Résumé 74
4.6 EXPLORATION DES PERFORMANCES DE LA PLATE-FORME 74
4.6.1 Objectifs 74
4.6.2 Simulation 74

4.6.3 Analyses 75
4.6.4 Résumé 76
4.7 Synthèse 76

4.1 Rappel des objectifs

Nous avons vu dans l'état de l'art que les méthodes de modélisation de systèmes temps-réel, bien que de plus en plus matures, ne permettaient pas d'explorer les performances d'une plate-forme avionique en avance de phase. Soit la méthode n'est pas applicable en toute phase amont du processus de conception car elle nécessite de connaitre les applications amenées à être exécutées sur la plate-forme [Varona-Gomez et al., 2005] et [Casteres and Ramaherirariny", 2009], soit la méthode n'est pas adaptée pour l'avionique, i.e. ne prend pas en compte les contraintes avioniques [Varona-Gomez and Villar, 2009], soit la méthode n'est pas mature [Januzaj et al., 2009]. Pour cela, nous proposons une méthode de modélisation de plate-forme avionique en avance de phase en nous appuyant sur des méthodes existantes tout en les complétant. Nous allons ici présenter cette méthode de modélisation, dont les objectifs premiers sont :

- Fournir un modèle de la plate-forme dans son ensemble, via la modélisation de la partie applicative, de la partie matérielle et du système d'exploitation. Ce modèle décrira aussi l'allocation des ressources. En tant que traduction des spécifications logicielles et matérielles, ce modèle pourra servir de base lors de discussions sur la future architecture (contraintes, performances attendues, etc.) de la future plate-forme entre l'avionneur et le plate-formiste ;
- Fournir un modèle plus précis de la plate-forme d'exécution (système d'exploitation et partie matérielle) qui permettent d'anticiper sur les performances de l'architecture proposée afin de vérifier sa compatibilité avec les exigences fournies. Ce modèle doit aider à mieux orienter le processus de conception pour limiter l'impact d'erreur dans la définition de l'architecture.

Dans le chapitre précédent, nous avons vu qu'une méthode de modélisation de plate-forme et d'exploration de performances en avance de phase devait comporter certaines étapes, et répondre à plusieurs points techniques que nous rappelons ci-dessous :

© 2012 TELECOM ParisTech

Point technique 1 : une étape de modélisation de la partie applicative pour extraction d'un scénario de simulation de la plate-forme d'exécution : cette modélisation permet d'une part de modéliser la configuration applicative (ensemble des partitions et tâches logicielles, et leur politique d'ordonnancement), d'autre part la génération de stimuli pour venir stimuler le futur modèle de la plate-forme. Nous avons identifié que le principal point technique de cette étape réside dans le fait que les applications ne sont pas forcément accessibles. La modélisation de la partie applicative doit donc être suffisamment représentative malgré le non accès au code des applications, afin de générer un scénario de simulation pertinent.

Point technique 2 : une étape de modélisation architecturale de la plate-forme en avance de phase : en tant que traduction des spécifications logicielles et matérielles, le modèle architectural doit permettre une représentation plus "intelligible" de ces données. Il décrit la plate-forme dans son ensemble, à savoir partie applicative, partie matérielle et déploiement. Ce modèle sert de base à des premières analyses de dimensionnement du système. Pour cela, il doit prendre en compte les spécificités de conception d'un système propre au domaine de l'avionique, découlant principalement des recommandations du standard ARINC653 vu dans le chapitre 1.

Point technique 3 : un modèle comportemental de la partie matérielle de la plate-forme et du système d'exploitation : ce modèle affine la description matérielle du modèle architectural précédent. Il permet l'étude plus précise du comportement de la plate-forme matérielle en réponse à l'exécution de la partie applicative. L'étude du comportement se traduit par l'étude de la consommation des ressources matérielles : temps d'exécution suite à un sitmulus, consommation électrique, etc. Le point technique est ici lié à la représentativité des modèles comportementaux des éléments matériels de la plate-forme et de son système d'exploitation, qui doivent représenter le comportement réel des éléments modélisés sous peine de fausser les analyses futures des performances de la plate-forme.
Ce modèle doit également donner la possibilité d'exécuter le scénario élaboré dans l'étape 1, tout en assurant la validité de cette exécution (pas de perte et acheminement correct des stimuli) afin de vérifier la compatibilité entre les performances extraites et les exigences.

© 2012 TELECOM ParisTech

Nous nous proposons désormais de décrire notre approche. Nous commencerons par détailler les différentes étapes qui la composent, et nous verrons comment chacune d'elles répond à l'un des problèmes techniques que nous avons rappelé.

4.2 Structure globale

Le processus de modélisation [Lafaye et al., 2010b] que nous proposons vise à étendre les approche actuelles, en s'appuyant notamment sur quatre parties adressant les quatre étapes décrites précédemment. Comme nous le voyons sur la figure 4.1, notre processus de modélisation est une approche "top-down" ou descendante. Partant de fichiers de spécifications du système (partie applicative et partie matérielle), il modélise à plusieurs niveaux d'abstraction la plate-forme, afin d'affiner successivement ce modèle et permettre d'effectuer différents types d'analyses tirant parti de ces niveaux d'abstraction.

FIGURE 4.1 – Méthode de modélisation : vue générale

Nous allons désormais détailler chaque partie de ce processus. Pour chacune de ces parties, nous verrons :

- La réponse apportée aux objectifs ;
- La démarche proposée pour répondre à ces objectifs ;
- L'existant éventuel sur lequel s'appuie la démarche.

 © 2012 TELECOM ParisTech

4.3 Capture des paramètres de la partie applicative et génération des stimuli

4.3.1 Objectifs

Cette étape a pour objectif l'extraction des paramètres principaux de la partie applicative dans le but de générer un scénario de simulation pour le futur modèle comportemental de la plate-forme d'exécution. Pour assurer la représentativité du scénario de simulation, il est nécessaire d'extraire un minimum d'informations caractérisant la partie logicielle. Nous allons détailler ci-dessous les démarches d'extraction de ces informations, et l'utilisation qui en sera faite.

4.3.2 Extraction de la configuration logicielle

Nous l'avons vu dans le chapitre 1, la partie applicative d'un système avionique est constituée d'une ou plusieurs partitions qui hébergent chacune tout ou partie d'une application avionique. Ces partitions sont ordonnancées suivant un schéma d'exécution fixe, la MAF définie par le standard ARINC653, qui se répète tant que le système est en marche. Au sein de ces partitions, le code applicatif est exécuté par un ensemble de tâches effectuant principalement des accès aux ressources matérielles et des accès aux services du système d'exploitation embarqué. Ainsi la première opération consiste à récupérer une configuration logicielle. Par configuration, nous entendons :

- Le schéma d'ordonnancement des partitions, la MAF ;
- Pour chaque partition, la durée d'exécution (”Duration”), la période d'activation (”Period”), ainsi que le nombre de tâches associées ;
- Le schéma d'ordonnancement des tâches au sein de la partition ;
- Pour chaque tâche, la priorité de base (”Base Priority”) et la priorité en cours (”Current Priority”), la période d'activation (”Period”), et le temps alloué pour son exécution (”Time Capacity”).

Ces paramètres constituent le minimum des informations à renseigner concernant la configuration. Celle-ci nous permettra par la suite d'organiser les stimuli applicatifs créés. Mais d'autres informations peuvent être ajoutées comme par exemple l'identifiant de chaque partition ou tâche, afin de faciliter leur identification au sein du modèle

© 2012 TELECOM ParisTech

architectural. D'autre part, la partie applicative n'étant pas encore figée, la configuration donnée en entrée sera testée mais pourra être amenée à varier. Dans ce cas, une nouvelle itération de modélisation - simulation peut rapidement être lancée, permettant de déterminer si l'architecture est compatible avec la nouvelle configuration, et dans le cas contraire, permettra d'explorer quelle partie de la plate-forme d'exécution est à modifier.

4.3.3 Génération des stimuli applicatifs

Afin de générer ces stimuli en l'absence du code source, ce qui interdit toute méthode de parcours de code, il est nécessaire d'avoir en entrée une description algorithmique de chaque tâche. Par description algorithmique nous entendons une description comportementale de la tâche, représentée par l'enchainement de ses fonctions de base (type FFT). Ces fonctions de base sont elles-mêmes découpées en sous-blocs algorithmiques principaux (boucles, calcul matriciel etc.). Ces sous-blocs vont nous servir à générer une partie du scénario de simulation, grâce à une technique de patron de transformation. Par patron de transformation nous entendons une ou des règles de génération de stimuli : lorsque le processus détecte un bloc algorithmique connu, il exécute le patron correspondant, i.e. génère une séquence de stimuli représentant le comportement de ce bloc.

La figure 4.2 résume cette étape de production d'une partie du scénario de simulation par transformation de blocs algorithmiques via des patrons : partant de la description algorithmique de la tâche en sous-blocs, la méthode analyse chacun de ces sous-blocs, et éventuellement les paramètres associés, afin de déterminer quel patron leur appliquer. Une fois déterminé, le patron est appliqué et génère un ou plusieurs stimuli correspondants.

© 2012 TELECOM ParisTech

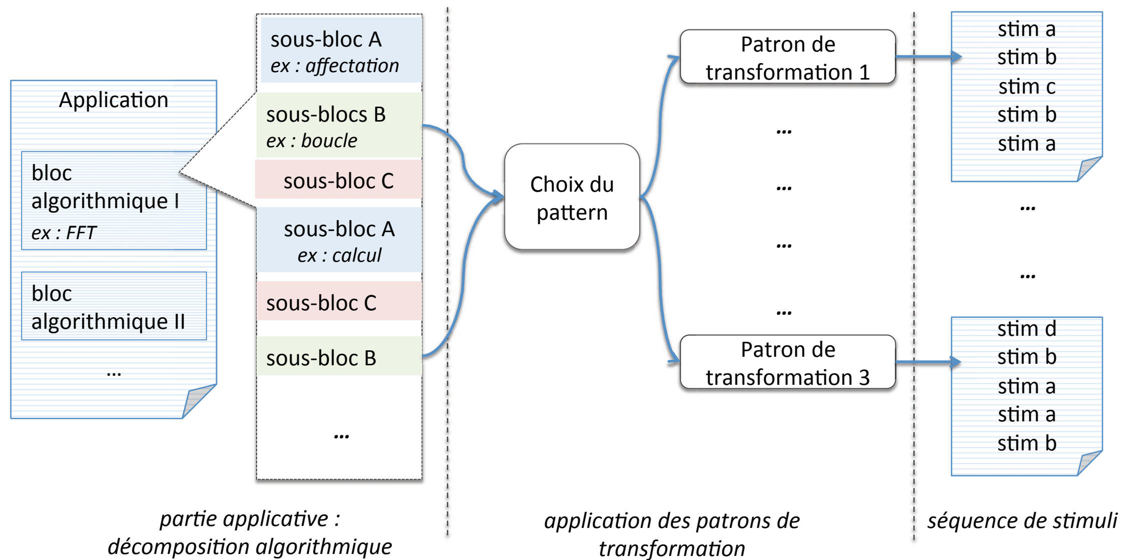

FIGURE 4.2 – Génération d'une séquence de stimuli à partir d'un bloc algorithmique

Cette méthode présente plusieurs avantages : elle permet d'une part au développeur d'applications de décrire la/les applications amenées à être exécutées sur l'architecture à tester sans décrire le code applicatif en lui-même. En effet, ce dernier n'est pas forcément défini en phase amont du cycle de développement, et même dans le cas où l'application existerait (développée pour une autre plate-forme par exemple), l'avionneur ne serait pas forcément disposé à livrer le code source. Toutefois, pour être efficace, cette méthode suppose de développer un ensemble de sous-blocs algorithmiques dans lequel l'utilisateur peut venir piocher pour décrire la/les applications. Cela suppose également de développer un ensemble de patrons de génération de stimuli associé à ces blocs.
D'autre part, la méthode reste relativement souple et modulaire, permettant de venir enrichir/modifier/affiner les patrons de génération de stimuli suivant i) les sous-blocs algorithmiques à cibler, ii) la plate-forme cible (spécialisation des stimuli). Ainsi, nous pouvons imaginer, suivant le niveau de précision voulu et des informations connues relatives aux éléments de la plate-forme :

- Un jeu de patrons "génériques", indépendant de la plate-forme matérielle cible ;
- Des jeux de patrons plus spécifiques aux différentes architectures cibles. Ces jeux profitent du fait qu'une architecture avionique est souvent une amélioration ou une évolution d'une architecture existante ; il est alors possible de réutiliser ces patrons dans de futures modélisations.

© 2012 TELECOM ParisTech

La figure 4.3 résume le processus de génération et organisation de ces stimuli applicatifs, via la méthode de transformation de sous-blocs algorithmiques en stimuli : pour chaque tâche d'une partition sont extraites la description algorithmique et les principales propriétés. Pour chaque bloc de la description comportementale, nous exécutons le patron correspondant qui génère une séquence de stimuli.

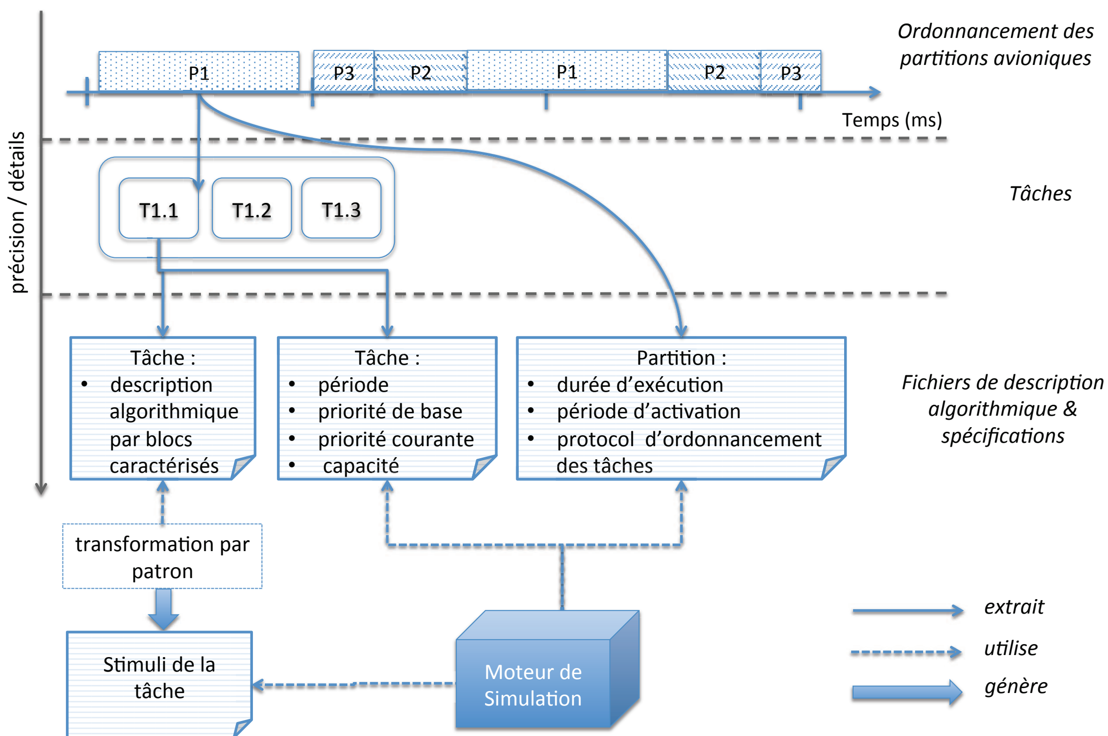

FIGURE 4.3 – Elaboration d'un scénario de simulation

4.3.4 Résumé

La démarche proposée, une fois mise en oeuvre, répond au point technique 1 : elle permet la génération des stimuli applicatifs sans avoir accès au code applicatif lui-même, grâce à une description à base de blocs comportementaux des tâches applicatives, ainsi qu'à des jeux de patrons de génération de stimuli associés.

© 2012 TELECOM ParisTech

4.4 Modélisation architecturale de la plate-forme avionique

4.4.1 Objectifs

Cette étape a pour but premier de fournir une représentation plus lisible du système dans son ensemble, à savoir la configuration de la partie applicative (partitions et tâches), l'architecture matérielle (composants et interconnections), et la modélisation gros grain de l'allocation des ressources matérielles aux ressources logicielles. Cette concrétisation doit permettre de faciliter les échanges entre les fournisseurs d'applications et le plate-formiste, en aidant à la définition ou modification de l'architecture de la plate-forme. Enfin, ce modèle architectural doit également permettre de prédimensionner le système, tel le dimensionnement de la mémoire suivant l'allocation aux partitions.

4.4.2 Démarche

Le modèle architectural développé ici est constitué des deux parties principales du système : la partie applicative et la partie matérielle. Ce modèle suit l'un des principes de l'IMA, à savoir la séparation de la partie applicative et de la plate-forme d'exécution (partie matérielle et système d'exploitation) ; ainsi la partie applicative peut être modélisée de manière séparée vis-à-vis de la plate-forme d'exécution, ces deux parties se retrouvant réunies au moment de l'allocation des ressources. Nous avons donc :

- La partie applicative de la plate-forme, modélisée via des composants logiciels qui représentent les partitions. Ces composants contiennent des sous composants logiciels représentant les tâches. Les composants partitions et tâches sont caractérisés par les paramètres extraits lors de la partie précédente (période d'activation de la partition, capacité de la tâche, etc.) ;
- La plate-forme d'exécution, traduction des spécifications architecturales, qui modélise les différents composants matériels et le système d'exploitation. Chaque composant matériel est caractérisé par ses paramètres principaux (capacité, latence de lecture / écriture et taille d'une mémoire ; fréquence d'un processeur, etc.). Ces modèles sont également connectés suivant les informations de connexions définies dans la proposition d'architecture matérielle en entrée. Concernant le système d'exploitation, celui-ci est également modélisé par ses

© 2012 TELECOM ParisTech

principales caractéristiques, à savoir le protocole d'ordonnancement des tâches, le quantum de temps entre chaque tic ou encore le nombre maximal de tâches autorisées ;

- Enfin, la partie déploiement du modèle architectural permet de représenter l'allocation des ressources matérielles aux éléments logiciels. Notamment, cette partie du modèle permet de représenter les principes de partitionnement spatial et chaque partition de zones mémoires spécifiques et de fenêtres temporelles au sein de la MAF.

4.4.3 Introduction des spécificités avioniques

La première spécificité d'une architecture avionique tient dans le fait qu'il s'agit d'une architecture partitionnée (chapitre 2). Certains projets se sont déjà penchés sur cette spécificité, notamment celui ayant débouché sur les annexes ARINC653 dans les modèles architecturaux AADL [Delange et al.,]. Ce projet ajoute au langage des propriétés propres au domaine de l'avionique, permettant notamment de modéliser le partitionnement temporel en définissant la place de chaque partition au sein de la MAF (on parle alors de fenêtre temporelle). Il offre également la possibilité de modéliser le partitionnement spatial, en modélisant la mémoire principale comme un composant matériel principal, lui-même composé de sous-composants mémoire représentants les segments mémoire. Ainsi, en attachant chaque partition non pas au composant mémoire, mais à un sous-composant (segment), il est possible de définir l'isolation de la partition en mémoire. La figure 4.4 donne un aperçu d'un modèle architectural : nous y voyons la partie applicative modélisée sous forme de partitions contenant les tâches applicatives. Chaque partition est associée à une fenêtre temporelle et à un segment mémoire, représentant les principes de partitionnement que l'on vient de voir.

L'autre caractéristique des plates-formes avioniques est d'être centrée autour d'un réseau commun, i.e. chaque calculateur communique avec les autres au travers d'un réseau. Or, chaque calculateur étant un modèle en soi (ensemble de partitions s'exécutant sur une architecture matérielle composée au moins d'un processeur, une mémoire principale et diverses entrées-sorties), il est nécessaire de disposer d'une approche permettant la composition de modèles de systèmes, i.e. des modèles eux-mêmes représentant déjà un système de composants logiciels et matériels. Ensuite, ces modèles de système doivent pouvoir être connectés via un composant représen-

 © 2012 TELECOM ParisTech

tant le réseau et caractérisé par les principales propriétés de celui-ci (principalement les temps de transfert de données d'un calculateur à un autre via les chemins logiques, voir chapitre 3).

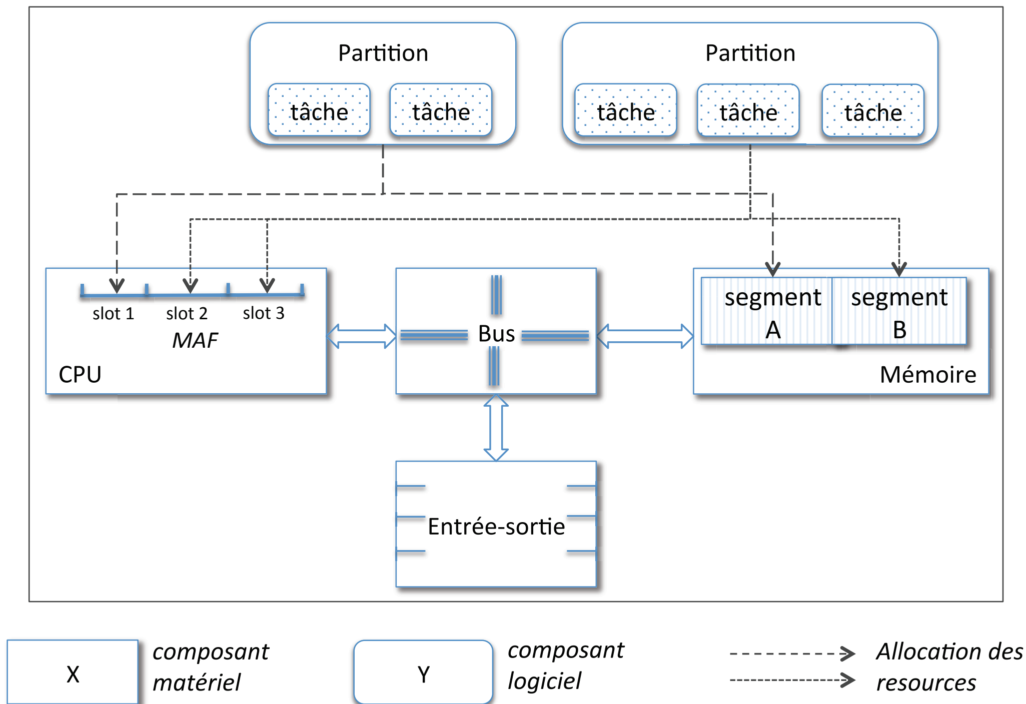

FIGURE 4.4 – Modèle architectural d'une plate-forme

Ce modèle peut également contenir des informations supplémentaires propres à l'ARINC653, comme une description des règles de communication définies dans le standard ARINC653. Nous l'avons vu dans le chapitre 1, le standard distingue les communications intra-partition (communications entre les tâches au sein d'une même partition) et inter-partition (communication entre les tâches de partitions différentes). Ce modèle permet, si les informations concernant la configuration de la partie applicative et la configuration matérielle sont suffisamment renseignées, de faire des premières analyses, comme des analyses d'ordonnançabilité [Singhoff et al., 2005], et/ou des analyses de pré-dimensionnement [Senn et al., 2008].

© 2012 TELECOM ParisTech

4.4.4 Ajout de propriétés

Nous avons vu dans le chapitre 3 que la plupart des langages de modélisation architecturale définissent, avec chaque composant, un ensemble de propriétés. Celles-ci constituent les propriétés de base, mais restent souvent très générales. Or nous avons besoin, pour la suite du processus, de pouvoir ajouter des propriétés afin d'affiner la description des composants, matériels principalement. Nous avons donc prévu dans notre processus la possibilité d'ajouter des propriétés aux composants de la plate-forme d'exécution. Ceci afin de pouvoir s'adapter à ces nouveaux composants et/ou nouvelles architectures de calculateurs.

4.4.5 Résumé

Dans les méthodes de modélisation actuelles, cette étape de modélisation architecturale est généralement déjà bien définie. Nous avons vu dans le chapitre 3 que certaines d'entre elles intègrent la prise en compte des spécificités des systèmes avioniques. Ainsi, dans notre approche nous nous appuierons sur ces méthodes existantes, et plus particulièrement sur la méthode définie dans [Delange et al.,] afin d'élaborer le modèle architectural de la plate-forme avionique. Ce modèle architectural doit permettre de répondre au point technique 2, en intégrant à ce modèle les particularités propres à l'avionique comme le partitionnement spatial et temporel.

4.5 Génération d'un modèle comportemental de la plate-forme d'exécution

4.5.1 Objectifs

Nous l'avons vu, le modèle développé précédemment donne une vision plus concrète des spécifications logicielles et architecturales, et offre un support pour mener certaines analyses de dimensionnement de la plate-forme. Toutefois, au sein de ce modèle les composants, qu'ils soient logiciels ou matériels, sont décrits sous forme de "boites noires". L'absence de description comportementale ne permet pas de venir examiner le comportement d'un composant, et par extension de la plate-forme, en réponse à un ou plusieurs stimuli applicatifs. Le but de cette étape est de venir affiner la description de la partie matérielle du système, en ajoutant aux composants concernés une description comportementale. L'objectif est de rendre possible, via un moteur de

© 2012 TELECOM ParisTech

simulation, i) l'exécution du scénario de simulation élaboré précédemment, ii) l'extraction dynamique des performances de la plate-forme, iii) l'extraction des limites de la plate-forme d'exécution en terme de puissance de calcul, utilisation mémoire, etc.

4.5.2 Génération des composants comportementaux

Le point d'entrée de cette étape est le modèle architectural élaboré précédemment à l'aide des diverses informations des parties applicatives et matérielles. Partant de ce dernier, nous allons affiner la description de chacun des composants matériels. La première partie de cette étape revient donc à extraire du modèle architectural chaque élément matériel, ainsi que ses paramètres. Une fois ces éléments extraits, nous ajoutons à chaque élément une partie comportementale, en allant chercher celle-ci dans une base de données. En effet, les modèles architecturaux étant sous forme de boites noires, nous n'avons pas suffisamment d'informations afin de générer les parties comportementales des composants. Ainsi l'approche proposée est de passer par une base de données qui contient les parties comportementales de plusieurs composants, et à différents niveaux de précision. Plus précisément, ces parties comportementales sont classés en 2 catégories :

- Les modèles comportementaux génériques, comme un modèle comportemental de mémoire DRAM, de bus, etc. ;
- Les modèles comportementaux spécifiques, correspondant à des classes de composants particulières (DDR3, bus PCI, etc.).

Les parties comportementales "génériques" servent lorsque le composant en cours de traitement ne possède pas de partie comportementale correspondante dans la base. Le niveau de précision dans la description de l'élément est moindre, mais permet toutefois de pouvoir adresser le composant au sein de la plate-forme, et ainsi ne pas interrompre l'étude du comportement global de la plate-forme. La configuration des composants se fait via les paramètres du composant en question renseignés dans le modèle architectural. Pour une mémoire par exemple sa latence d'accès en lecture, écriture, sa capacité ou encore la largeur des données acceptées sont quelques-unes des propriétés principales qui servent à configurer sa partie comportementale. Nous proposons d'autre part d'organiser ces paramètres de configuration par catégorie, ou "point de vue", afin de pouvoir orienter l'analyse des performances de la plate-forme sur un aspect particulier (consommation d'énergie, latence d'exécution, etc.).

© 2012 TELECOM ParisTech

Les avantages liés à cette méthode de base de données résident dans les propriétés intrinsèques de cette base : un composant mis en base est supposé conforme au composant réel, dans un certain intervalle de confiance qui doit être déterminé. Cela permet aux futurs utilisateurs d'utiliser ce composant en connaissant ce degré de représentativité. De plus, la réutilisabilité de ces composants permet un gain de temps, car une partie comportementale de composant mise en base évite de devoir développer à chaque modélisation cette partie. Ensuite, suivant le formalisme choisi, il pourrait être possible d'intégrer des modèles "extérieurs", i.e. développés en dehors de notre méthode de modélisation. Enfin, la base peut être enrichie afin de s'adapter à tout nouveau composant. Une fois les modèles comportementaux élaborés, il reste à les connecter. Les informations de connexion sont extraites du modèle architectural, et permettent de déterminer les composants connectés entre eux. Le schéma 4.5 permet de résumer cette approche de génération de modèles comportementaux via une base de données.

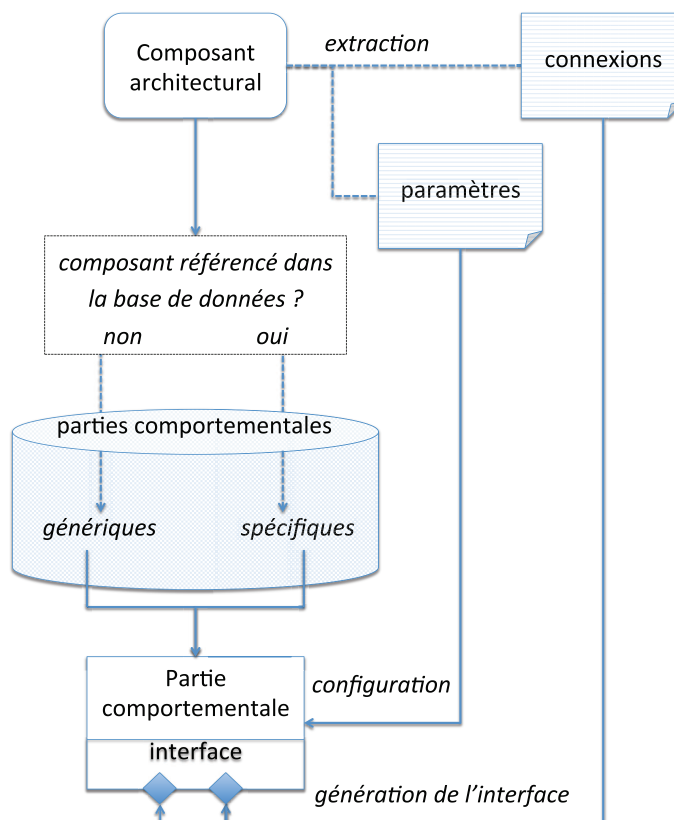

FIGURE 4.5 – Génération d'un modèle comportemental de composant matériel

© 2012 TELECOM ParisTech

Concernant le système d'exploitation, celui-ci n'est que très peu présent dans le modèle architectural précédent (essentiellement les informations d'ordonnancement), et doit être ajouté à cette étape. Pour rappel, nous avons vu qu'une application était principalement constituée de deux groupes d'instructions : les instructions qui adressent directement des éléments matériels (calcul), et des instructions qui font appel aux services du système d'exploitation. Ce système d'exploitation, dans le cas de systèmes avioniques, doit être conforme au standard ARINC653, et offrir un certain nombre de services prédéfinis aux applications. Ainsi, ayant vu comment étaient générés les composants matériels de la plate-forme, il reste à voir comment sont générés les services ARINC. Pour cela, une base de données est également développée, contenant une description comportementale de chacun de ces services, qui consiste en la description de l'enchainement des opérations exécutées. A titre d'exemple, le service *get_partition_status*, sur la figure 4.6, consiste en une succession de lecture en mémoire des différents paramètres de la partition (identifiant, période, durée, etc.). Ces opérations sont alors traduites en accès matériels, qui viennent stimuler les composants de la plate-forme d'exécution.

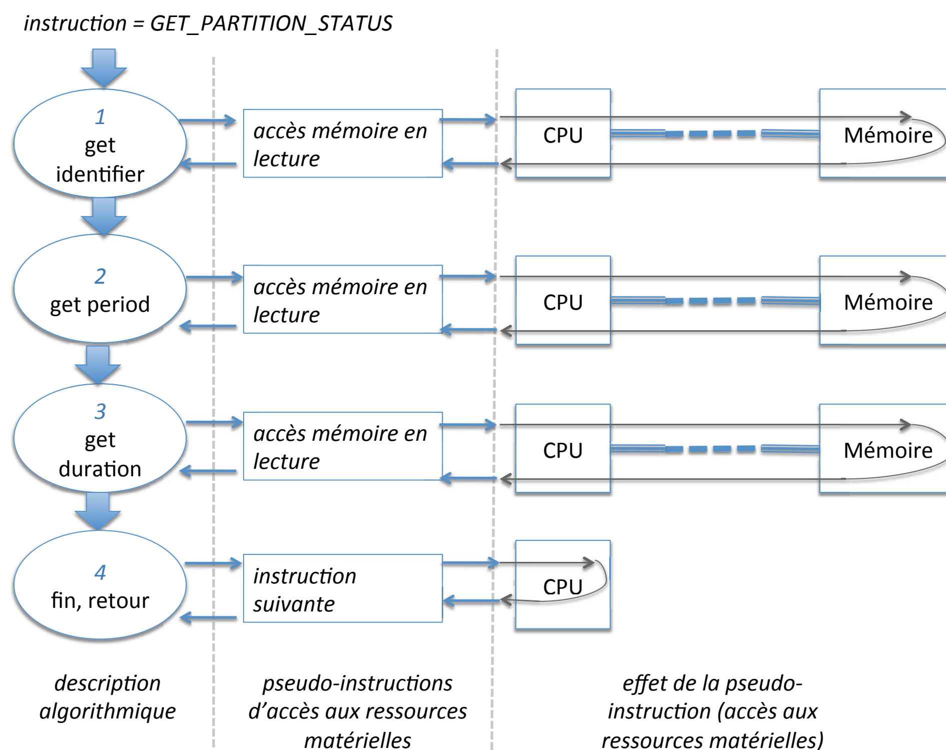

FIGURE 4.6 – Exemple de représentation comportementale d'un service : *get partition status*

© 2012 TELECOM ParisTech

4.5.3 Résumé

Cette étape d'enrichissement des informations contenues dans chacun des éléments de la plate-forme d'exécution par une partie comportementale et quelques propriétés supplémentaires, doit rendre possible l'étude des performances de la plate-forme d'exécution. Plus particulièrement, la partie comportementale ajoutée aux composants apporte un niveau de précision supplémentaire, et offre le moyen de venir étudier la réponse des composants et de la plate-forme d'exécution aux stimulus applicatifs que l'on exécute. La méthode répond au point technique 3 (représentativité des modèles comportementaux), en s'appuyant sur des parties comportementales validées avec un niveau de représentativité déterminé lors du développement du modèle. De plus, elle assure la conservation des propriétés renseignées dans le modèle architectural, en venant simplement ajouter la partie comportementale des composants.

4.6 Exploration des performances de la plate-forme

4.6.1 Objectifs

Cette dernière partie de notre processus de modélisation et d'exploration des performances d'une plate-forme avionique, consiste en une phase d'analyse des résultats extraits d'une simulation. En effet, dans notre méthode nous nous appuierons sur un moteur de simulation afin de répondre à l'un des objectifs de notre méthode, à savoir dépasser la précision des analyses statiques en effectuant des analyses dynamiques des performances de la plate-forme.

4.6.2 Simulation

Lors de la première étape de notre processus, nous avons généré un scénario de simulation de la partie matérielle de la plate-forme. L'exécution valide de ce scénario est soumise à deux points : i) une transmission des stimuli sans perte, ii) une exécution de ceux-ci par les modèles comportementaux. Concernant le point i), s'appuyer sur un moteur existant et validé permet de répondre à cet impératif.
Concernant le point ii), la bonne exécution des stimuli dépend des modèles comportementaux utilisés et du procédé permettant la communication entre le moteur de

© 2012 TELECOM ParisTech

simulation et ces modèles. Le procédé doit assurer qu'aucun stimulus n'est perdu lors de l'acheminement. L'approche utilisée est de définir une interface de communication standard moteur de simulation - composant de la plate-forme d'exécution, chargée de gérer les communications grâce à un protocole de communication simulateur - modèles. La figure 4.7 permet de présenter l'étape de simulation, en pointant les endroits où il est nécessaire d'assurer la bonne exécution des stimuli.

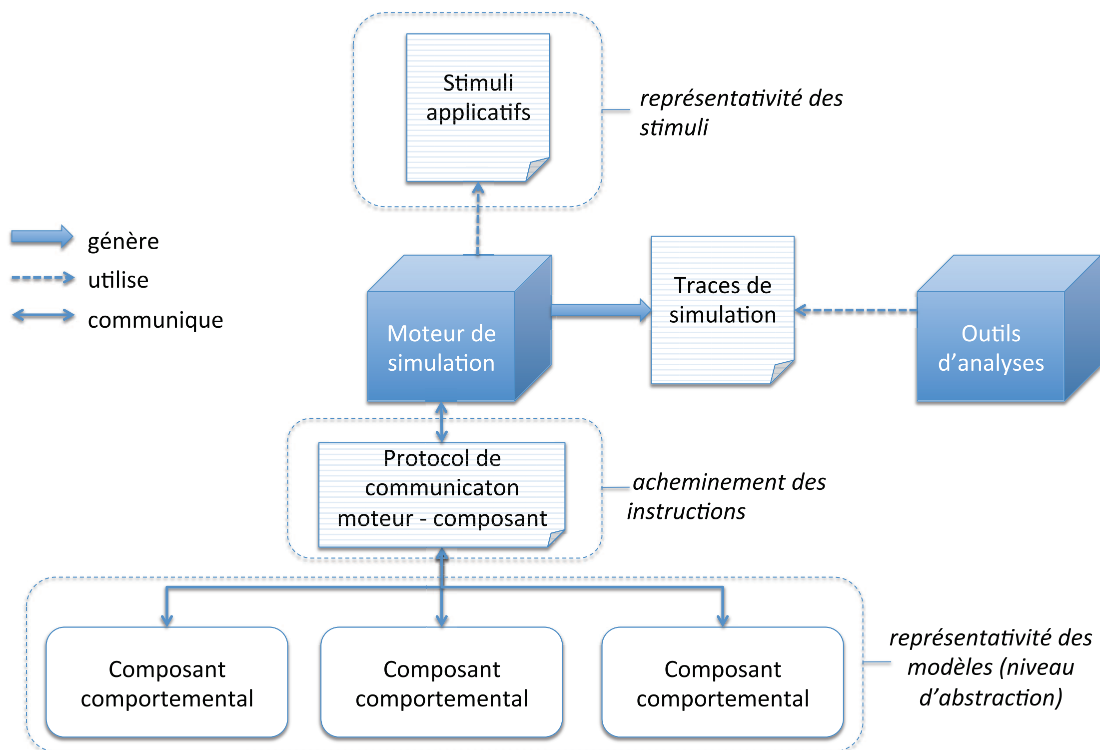

FIGURE 4.7 – Phase de simulation de la plate-forme matérielle et système d'exploitation

4.6.3 Analyses

L'approche choisie concernant l'analyse des résultats est d'extraire les traces de simulation au fur et à mesure, et d'effectuer des analyses à plusieurs niveaux : par exemple, analyser les réponses des différents éléments matériels à un stimuli donné, ou, plus largement, analyser l'exécution d'une partition d'un point de vue performance matérielle. Afin de conserver un aspect modulaire dans le processus de modélisation et analyses, les traces de simulation extraites sont traduites dans un formalisme connu (un format standard de type XML par exemple), qui servira de point d'entrée à l'outil /

© 2012 TELECOM ParisTech

aux outils d'analyses, et pourra faciliter le développement de passerelles vers des outils existants. La précision des analyses effectuées découle de la représentativité des stimuli applicatifs extraits ainsi que du niveau de précision des composants comportementaux. Rappelons que l'objectif est d'explorer les performances de la plate-forme matérielle vis-à-vis des applications qu'elle aura à exécuter.

4.6.4 Résumé

Cette étape de simulation s'appuie sur de l'existant via un moteur de simulation validé. Il prend en entrée les stimuli applicatifs, traduits dans un formalisme accepté par le moteur, et vient stimuler les modèles comportementaux, eux-aussi développés dans ce même formalisme. Un protocole de communication entre le moteur et les modèles sera proposé, afin d'assurer qu'aucun stimulus n'est perdu ou acheminé vers un mauvais composant en cours de simulation. Enfin, les méthodes proposées pour l'extraction des stimuli applicatifs et pour la modélisation comportementale des modèles matériels, associées à ce protocole, donnera une estimation de l'intervalle de confiance des performances extraites de la simulation. Cela permettra de répondre à la seconde partie du point technique 3.

4.7 Synthèse

Dans ce chapitre, nous avons présenté les principes de notre approche de modélisation - simulation en réponse aux objectifs fixés pour l'exploration des performances d'une plate-forme en avance de phase. Celle-ci répond aux différents points techniques identifiés dans le chapitre précédent, en s'appuyant sur quatre étapes :

- Extraction des stimuli applicatifs depuis les spécifications logicielles, en garantissant une certaine représentativité des stimuli ;
- Modélisation de la plate-forme complète, afin de traduire les différentes spécifications (logicielles et matérielles) en un modèle concret. Ce modèle prend en compte les spécificités avioniques ;
- Transformation du modèle architectural de la plate-forme d'exécution en un modèle comportemental ;
- Simulation et analyse des performances de la plate-forme d'exécution en réponse aux stimuli applicatifs, en garantissant une exécution valide du scénario

© 2012 TELECOM ParisTech

de simulation sur le modèle comportemental.

Dans le chapitre suivant, nous allons détailler chaque partie de notre processus d'un point de vue technique, qui fera l'objet d'une section.

© 2012 TELECOM ParisTech

© 2012 TELECOM ParisTech

Chapitre 5

Mapping des éléments matériels AADL en SystemC

Sommaire

5.1 Introduction 81
5.2 Modélisation des spécificités avioniques au sein d'un modèle AADL 82
5.2.1 Modélisation du partitionnement temporel 82
5.2.2 Modélisation du partitionnement spatial 83
5.2.3 Modélisation de l'ordonnancement 83
5.3 Extraction des paramètres du modèle AADL 84
5.3.1 Extraction des paramètres des composants matériels de la plate-forme 84
5.3.2 Extraction des paramètres de la partie applicative de la plate-forme 85
5.3.3 Extraction des paramètres de connexion de la plate-forme matérielle 85
5.4 Génération de la plate-forme matérielle SystemC . . . 86
5.4.1 Description du comportement d'un composant 86
5.4.1.1 Base d'automates comportementaux 87
5.4.1.2 Extension des paramètres AADL pour l'adressage des automates SystemC 88
5.4.1.3 Configuration de l'automate 90
5.4.1.4 Modélisation du système d'exploitation . . . 91
5.4.2 Génération d'une interface générique : le container matériel 92

5.4.2.1 Rappel du principe de communication TLM 92
5.4.2.2 Le Container : principe 94
5.4.2.3 Le Container : mécanisme d'abonnement et filtrage 96
5.4.3 Intégration 98
5.5 SYNTHÈSE 100

© 2012 TELECOM ParisTech

5.1 Introduction

Le choix du langage pour la modélisation architecturale s'est porté sur AADL et son annexe ARINC-653. AADL permet en effet de représenter une plate-forme avionique dans sa vision architecturale (partie applicative, partie matérielle et ordonnancement), et offre également la possibilité, grâce à l'annexe ARINC-653 utilisée dans ces travaux [Delange et al.,], de modéliser les spécificités avioniques telles que le partitionnement spatial et temporel. Toutefois, au moment où ces travaux de thèse ont commencé, le langage AADL ne permettait pas de représenter la partie comportementale des éléments matériels de la plate-forme mais se limitait à une description ”boite noire” des composants. Or, nous avons vu dans le chapitre 2 qu'il était nécessaire d'affiner les modèles des éléments de la plate-forme d'exécution (composants matériels et système d'exploitation) si l'on souhaitait mener diverses études le comportement de cette dernière en réponse à des stimuli applicatifs.

Ainsi, partant du modèle de plate-forme décrite en AADL, notre méthode consiste à venir compléter les composants matériels en leur ajoutant une description comportementale. Le langage cible de cet enrichissement de modèle est SystemC. Standard IEEE très largement utilisé dans l'industrie [IEEE-Computer-Society, 2006], il permet la description comportementale d'éléments matériels et leur simulation grâce à son noyau de simulation intégré (ref). Plus précisément, nous ciblons le niveau d'abstraction TLM [OSCI, 2007] que nous avons vu dans l'état de l'art. Ce niveau d'abstraction, qui représente les échanges de niveau trames de données, est le meilleur compromis dans notre cas entre précision de modélisation et vitesse de simulation. Notre méthode s'appliquant en avance de phase, elle doit offrir un processus de modélisation et d'exploration des performances rapide et itératif. Ainsi, il doit être possible de générer et modifier un modèle comportemental de plate-forme matérielle relativement rapidement, de plus sans imposer à l'utilisateur de connaître le langage et devoir entrer les spécificités de la modélisation SystemC-TLM.

Nous allons voir dans cette section comment nous générons un modèle comportemental de plate-forme d'exécution en SystemC, en partant d'un modèle AADL. Nous verrons quels sont les mécanismes mis en oeuvre afin de faciliter cette génération et la rendre quasi transparente à l'utilisateur.

© 2012 TELECOM ParisTech

5.2 Modélisation des spécificités avioniques au sein d'un modèle AADL

Le langage AADL permet, grâce à son annexe ARINC-653, de concevoir un modèle architectural d'une plate-forme avionique. Comme nous avons vu dans le chapitre 3, cette modélisation architecturale se base sur des composants logiciels et matériels, sorte de "boites noires" configurées par des propriétés propres à chacun des éléments. Un modèle AADL se décompose en trois parties principales, à savoir la partie logicielle, la partie matérielle et la partie déploiement. La partie système d'exploitation est quant à elle représentée par des propriétés d'ordonnancement associées au processeur. Les spécificités avionique principales à prendre en compte dans ce modèle concernent le principe de partionnement spatial et temporel défini par l'ARINC-653.

5.2.1 Modélisation du partitionnement temporel

Comme il a été décrit dans le chapitre 1, le partitionnement temporel permet de garantir à chaque partition au moins une fenêtre d'exécution au sein de la MAF. Pour cela, l'annexe ARINC-653 utilise le composant "virtual processor", associé au processeur principal du modèle. Ce dernier se voit également définir les propriétés suivantes concernant la MAF : "Major Frame" pour définir la durée de la MAF, et "Slots Allocation" qui définit la position de chaque partition au sein de la MAF. Ensuite, chaque composant logiciel de type "process", qui modélise une partition, est associé à un "virtual processor" et définit les propriétés de la partition (période, durée, etc.). La figure 5.1 donne un aperçu de cette modélisation :

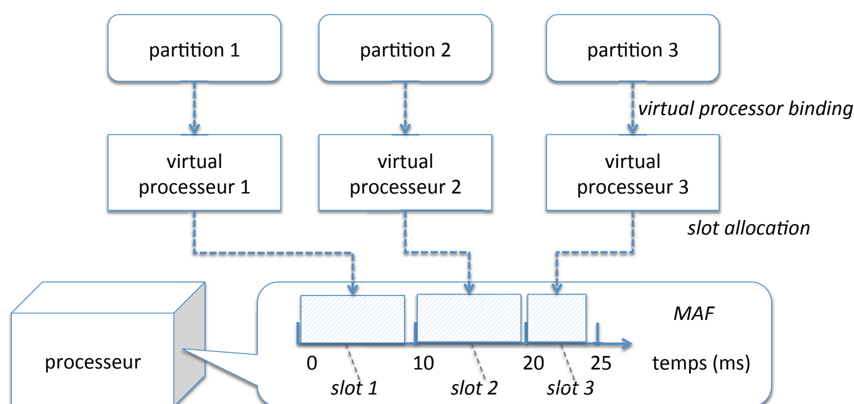

FIGURE 5.1 – Modélisation du partitionnement temporel en AADL

© 2012 TELECOM ParisTech

5.2.2 Modélisation du partitionnement spatial

Comme nous l'avons vu dans le chapitre 1, le partitionnement spatial permet d'isoler chaque partition en mémoire, i.e. chaque partition possède sa propre zone mémoire dédiée dans laquelle aucune autre partition ne peut venir écrire. Du point de vue de la modélisation, nous voyons sur la figure 5.2 que cela se traduit par le découpage du composant mémoire principal en sous-composants mémoire, lesquels seront dédiés à une unique partition. Concrètement, lors de l'intégration, chaque composant "virtual processor" vu ci-dessus se verra alloué un sous-composant mémoire par la propriété "Actual Memory Binding". Les propriétés de cette zone mémoire comme par exemple l'espace alloué, seront définies via les propriétés du sous-composant mémoire lui-même.

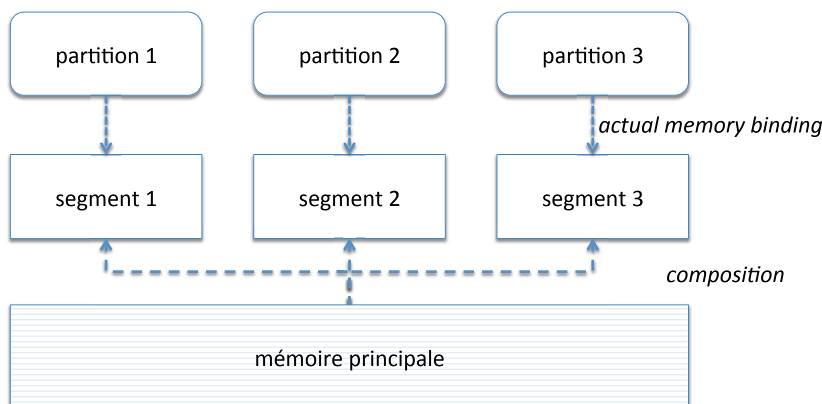

FIGURE 5.2 – Modélisation du partitionnement spatial en AADL

5.2.3 Modélisation de l'ordonnancement

Nous avons vu que dans un système avionique, l'ordonnancement se fait à deux niveaux : l'ordonnancement fixe des partitions via la MAF, et l'ordonnancement des tâches au sein de chaque partition. Comme le montre le schéma 5.3, la politique d'ordonnancement d'une partition pouvant varier d'une partition à l'autre, nous utilisons là encore les propriétés du composant "virtual processor" vu juste précédemment. La propriété "Scheduling Protocol" permet de définir la politique d'ordonnancement spécifique à la partition.

© 2012 TELECOM ParisTech

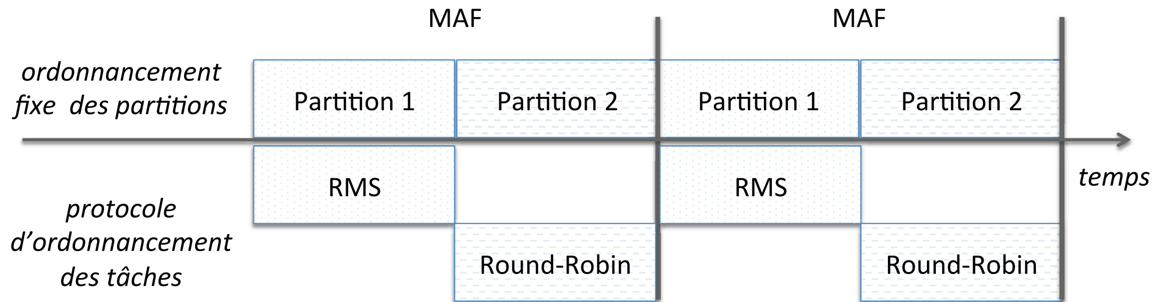

FIGURE 5.3 – Exemple d'ordonnancement de type ARINC-653

Nous considérons donc ici principalement les propriétés concernant l'ordonnancement (MAF et partitionnement des partitions). Nous allons voir un peu plus loin que nous récupérons également via le modèle AADL les propriétés de chaque partition et tâche, propriétés définies dans le standard ARINC653.

5.3 Extraction des paramètres du modèle AADL

Nous allons voir dans cette partie quelles informations sont extraites du modèle AADL pour la suite du processus.

5.3.1 Extraction des paramètres des composants matériels de la plate-forme

Une fois le modèle de notre plate-forme élaboré à l'aide d'AADL et son annexe ARINC-653, l'opération suivante consiste en l'extraction de ses caractéristiques principales. Nous avons vu que chacune des parties était modélisée via des composants configurés avec des paramètres. Nous parcourons le modèle AADL et stockons dans une liste les paramètres dimensionnants de chaque composant matériel du modèle, afin de les utiliser dans la partie génération du modèle correspondant SystemC. Par exemple, pour un composant mémoire nous conservons les propriétés suivantes : "memory protocol" qui définit les droits d'accès de la mémoire, "word size" qui indique la largeur des données acceptées, "base address" qui permet de définir la place d'un sous-composant mémoire au sein de la mémoire principale, "read time" et ""write time". Cette liste n'est pas exhaustive, mais donne un bon aperçu du type de propriétés que l'on conserve pour la suite de notre processus.

 © 2012 TELECOM ParisTech

L'un des avantages du langage AADL est qu'il permet l'ajout de propriétés au sein de fichiers de configuration propre à l'utilisateur. Toujours dans notre exemple de composant mémoire, les propriétés "refresh time" et "refresh period" ne sont pas définies par le standard. Or, ces propriétés peuvent être utiles car elles jouent sur le temps de réponse d'une mémoire vive telle qu'une DRAM. Si l'on veut ajouter ces propriétés dans le modèle de mémoire AADL, il est ainsi possible de définir un nouveau "property set", en plus du fichier de paramètres inclus au langage et contenant les propriétés que l'on a vu précédemment. Le modèle peut donc s'adapter aux besoins de modélisation, ce qui sera important là aussi pour la suite du processus.

5.3.2 Extraction des paramètres de la partie applicative de la plate-forme

Pour ce qui est de la partie applicative du modèle AADL, nous récupérons les informations d'ordonnancement et de configuration. Pour les informations d'ordonnancement nous récupérons le schéma d'ordonnancement, la MAF. Pour les informations de configuration, nous récupérons les propriétés suivantes :

- La période, la durée et l'identifiant de chaque partition ;
- La politique d'ordonnancement des tâches au sein de chaque partition ;
- Pour chaque tâche, nous récupérons son identifiant, sa priorité et sa période.

5.3.3 Extraction des paramètres de connexion de la plate-forme matérielle

Le second ensemble de paramètres extraits du modèle de plate-forme AADL concerne les connexions entre composants matériels, afin de conserver la configuration de la plate-forme matérielle lors de la génération de la plate-forme matérielle SystemC. Ces informations sont disponibles à deux endroits du modèle : au niveau de la partie intégration, la section "connections" spécifie les connexions entre les composants. D'autre part, les interfaces de chaque composant sont définies dans la partie déclaration des composants.
Nous allons maintenant voir comment nous générons et configurons la plate-forme matérielle (système d'exploitation et partie matérielle) à l'aide de ces informations.

© 2012 TELECOM ParisTech

5.4 Génération de la plate-forme matérielle SystemC

Le processus de modélisation de la plate-forme en AADL décrit jusque-là se base sur des travaux existants, [Delange et al.,], et sont le point de départ de notre contribution concernant la génération d'un modèle comportemental de la plate-forme, que nous avons développé. Nous allons décrire comment nous générons un calculateur, composé d'un système d'exploitation et de sa partie matérielle. L'objectif de cette étape est de permettre une génération du modèle comportemental de la plate-forme d'exécution (système d'exploitation et partie matérielle) quasi transparente à l'utilisateur. La génération d'un composant dans notre processus de modélisation se fait en deux parties : d'une part, nous décrivons le comportement du composant en lui-même, d'autre part nous générons l'interface de communication. Nous avons volontairement séparé ces deux parties, afin de rendre notre processus plus modulaire et générique. Nous verrons comment dans la partie génération de l'interface. Avant cela, nous allons commencer par détailler comment nous attachons une description comportementale à chaque composant.

5.4.1 Description du comportement d'un composant

Les éléments matériels à modéliser en SystemC au sein d'une plate-forme d'exécution concernent mémoires, processeurs, bus, etc. Or, nous avons vu qu'en AADL, les composants permettant de décrire les éléments matériels sont au nombre de quatre ("processor", "memory", "bus" et "device"). Ils correspondent aux principaux types de composants que l'on rencontre dans un système embarqué (unité de calcul, unité de stockage, unité de communication et unité d'interface d'entrée-sortie). Ces composants possèdent des propriétés considérées comme caractéristiques qui permettent ainsi de décrire les éléments matériels comme une boite caractérisée par ses principales propriétés. Notre but est ici d'affiner ces descriptions générales en adressant plusieurs composants d'un même type (mémoire DRAM, mémoire ROM, mémoire flash, etc.). Nous avons pour cela développé une méthode autorisant la génération de composants suivant la règle 1 vers N : à partir d'un composant type AADL (ex : "memory"), nous ciblons plusieurs éléments matériels de ce même type (ex : mémoire DDR, mémoire Flash, etc.). Dans cette optique, nous avons i) basé notre méthode sur une base d'automates comportementaux correspondant aux différents éléments de la plate-forme d'exécution, ii) utilisé le mécanisme d'extension de propriétés d'AADL.

© 2012 TELECOM ParisTech

5.4.1.1 Base d'automates comportementaux

Pour ajouter une description comportementale aux composants de la plate-forme décrits en AADL, nous nous appuyons sur une base de données comportant des automates comportementaux. Ces automates SystemC-TLM sont développés soit par les fabricants, soit par des ingénieurs qualifiés. Nous nous appuyons ici sur le fait qu'une architecture avionique est souvent l'évolution d'une architecture existante ; les composants précédemment développés restent donc disponibles et réutilisables, moyennant quelques modifications éventuelles. Dans cette base nous distinguons deux types d'automates en base :

- Les automates des composants déjà développés pour de précédentes plates-formes ou pour la plate-forme à modéliser ;
- Les automates dits "génériques". Ils modélisent des classes de composants (mémoire vive, bus interne, etc.). Ces automates sont utilisés pour modéliser les composants dont la description n'est pas disponible dans la base de données. Ces modèles sont certes moins représentatifs que les automates précédents, mais permettent de modéliser de manière gros grain le composant en question et ainsi ne pas interrompre le processus de modélisation.

Ces automates décrivent le comportement du composant à modéliser en fonction de la / des valeurs en entrée. Chaque automate est composé d'au moins deux états dans le cas le plus simple : un état d'attente où le composant ne fait rien ou presque, et un état d'exécution. Le plus souvent, la phase d'exécution est constituée de plusieurs états, avec branchements sur conditions à l'état suivant lorsque cette transition dépend de l'état interne du composant. Dans l'exemple simple de la figure 5.4, l'automate possède trois états. L'automate reste dans l'état d'attente tant qu'il ne reçoit aucun stimulus. Dès qu'il en reçoit un, l'automate passe dans un état d'exécution A ou B suivant le stimulus et/ou suivant son état interne. Une fois l'exécution terminée, il renvoie une réponse et retourne dans l'état d'attente. Sur notre modèle de mémoire vive générique, une part de la variabilité vient du rafraichissement qui peut survenir au moment d'une lecture ou écriture et retarder celle-ci, ou d'une trame invalide (taille de donnée à écrire trop grande). Ainsi, nous introduisons dans les modèles la variabilité qui manquait au niveau des modèles AADL.

© 2012 TELECOM ParisTech

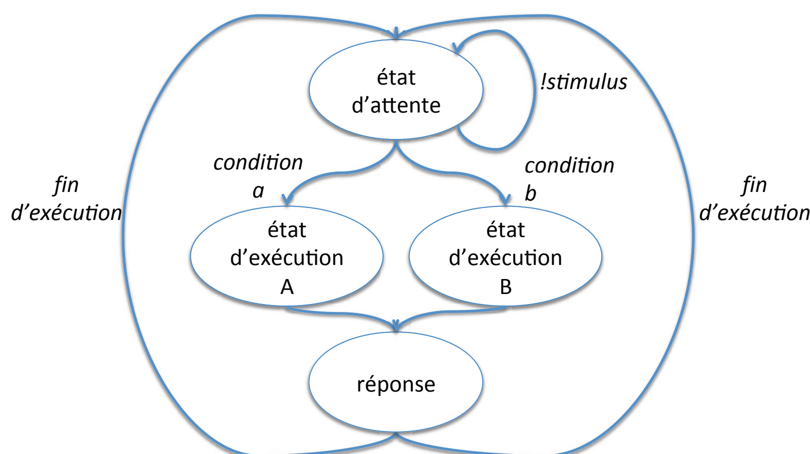

FIGURE 5.4 – Exemple d'automate comportemental

5.4.1.2 Extension des paramètres AADL pour l'adressage des automates SystemC

Afin de cibler les automates comportementaux depuis les composants AADL, nous avons profité de la possibilité d'étendre les jeux de paramètres de chaque composant. En effet, AADL définit pour chaque composant un jeu de paramètres "standard", mais parfois limité. Dans notre exemple de mémoire DRAM, les paramètres de rafraichissement ne sont par exemple pas définis. Toutefois, nous permettons d'ajouter ses propres jeux de paramètres (property set AADL) qu'il suffit d'inclure dans la définition du composant en question.

Ainsi, nous avons ajouté pour chaque composant matériel AADL deux paramètres principaux :

- Un type énuméré "*type*_supported_kind". Ce paramètre, défini dans un fichier de propriétés associées, permet de préciser les composants comportementaux dont on dispose en base (en plus des automates génériques) que l'on peut générer. Pour exemple, le paramètre "memory_supported_kind", s'il est défini avec la liste "(DDR2,DDR3,Flash)" permet de définir les types de mémoire non génériques que l'on est capable de cibler, à savoir les automates DDR2, DDR3 et Flash ;
- Une propriété "*type*_kind" qui défini le type du composant en cours. Par exemple, "memory_kind => DDR2" permet de définir que le composant en question est une mémoire de type DDR2. Grâce à cela, nous pouvons regarder si le com-

© 2012 TELECOM ParisTech

posant à générer (ici DDR2) est en base, en regardant dans la liste du type énuméré précédent. Dans le cas contraire le composant générique correspondant est utilisé. Dans l'exemple, il s'agit du composant générique mémoire.

Le schéma 5.5 permet de résumer cette opération de sélection de l'automate comportemental : dans le cas de gauche, on souhaite générer un composant DDR2. Celui-ci est présent dans la liste "memory_supported_kind" et peut donc être sélectionné en base. Dans le cas de droite, nous souhaitons générer un composant DDR3, qui lui n'est pas présent en base. Afin de ne pas interrompre le processus, nous utilisons le composant mémoire générique. Naturellement, dans ce cas, la précision sera impactée.

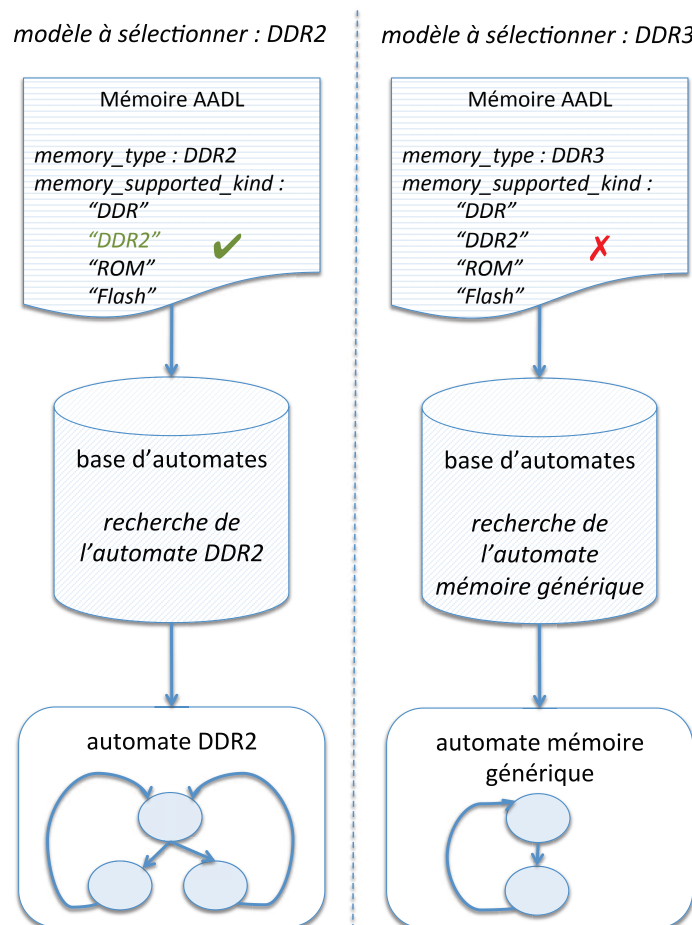

FIGURE 5.5 – Processus de sélection de l'automate comportemental

© 2012 TELECOM ParisTech

5.4.1.3 Configuration de l'automate

Chaque automate possède un certain nombre de variables de configuration dont il faut préciser la valeur. Afin d'optimiser notre processus de modélisation, nous offrons la possibilité de changer dynamiquement cette configuration, i.e. sans recompilation du modèle SystemC, et donc sans recompilation du modèle de plate-forme SystemC. Cela permet une exploration rapide du comportement du composant avec différentes valeurs de configuration. La figure 5.6 représente cette opération : les couples variables-valeur(s) extraits des composants matériels du modèle architectural AADL sont écrits dans des fichiers XML dont le nom correspond au nom du composant. Chaque fichier étant nommé suivant le nom du composant, et chaque composant étant unique en base, il n'y a pas de doublon de fichier de configuration. Lors de la phase de création du modèle comportemental du système, chaque composant matériel est instancié dans une phase d'initialisation : les fichiers XML sont alors lus par un module de notre processus de génération des modules SystemC, et les propriétés extraites sont utilisées pour configurer le composant.

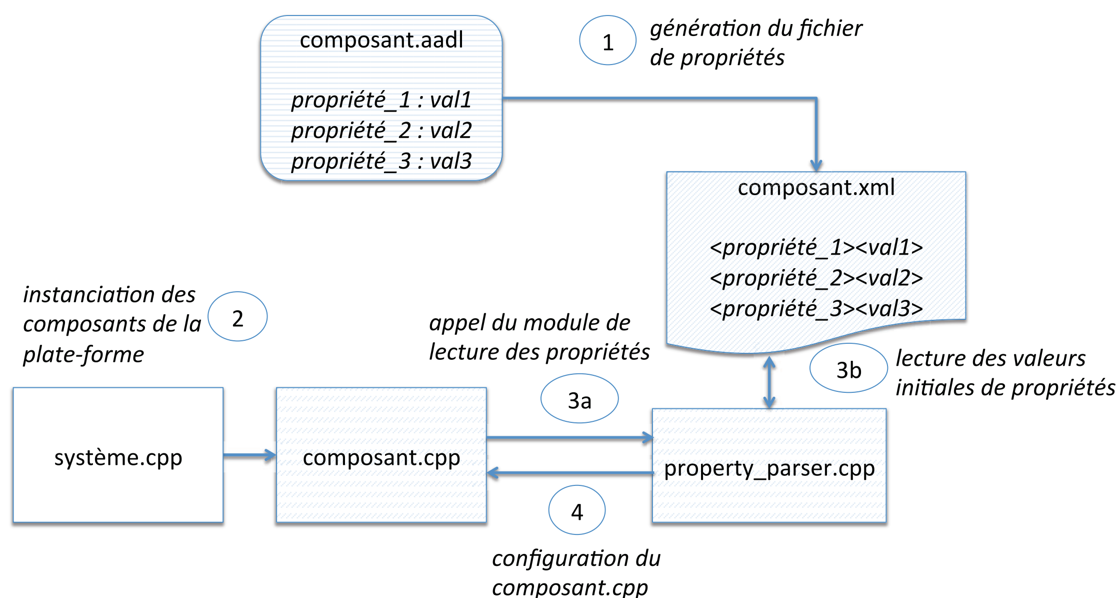

FIGURE 5.6 – Configuration d'un composant SystemC au sein de notre méthode

© 2012 TELECOM ParisTech

Si l'on souhaite tester une variante de l'architecture en cours, dont seules certaines propriétés de composants matériels varient (taille de la mémoire ou fréquence du processeur par exemple), il suffit de modifier les valeurs des propriétés soit dans le modèle AADL, soit directement dans les fichiers XML, puis de relancer la simulation, sans nouvelle compilation du/des modèles concernés par les modifications.

5.4.1.4 Modélisation du système d'exploitation

Dans le domaine avionique, les systèmes d'exploitation utilisés au sein des calculateurs doivent être conformes au standard ARINC653. Le standard défini les services que doit offrir le système d'exploitation aux applications. Celui-ci peut donc être vu comme une agrégation de services, qui constituent l'API ARINC653.

Afin de modéliser les services, nous nous sommes basés sur une représentation comportementale de ces derniers, que nous avons modélisés en SystemC. Nous adoptons alors le même mécanisme que pour les composants matériels. La description comportementale du service consiste en deux parties. La première décrit le service sous une forme gros grain d'appels aux ressources matérielles, ce qui permet, lors de l'appel à ce service, de venir stimuler les ressources matérielles de la plate-forme. Comme nous l'avons vu dans le chapitre précédent, le service *get_partition_status* retourne les valeurs des différents paramètres d'une partition (identifiant, période, durée, etc.). Ceci peut être traduit comme une suite d'accès à la hiérarchie mémoire dont le but est de charger ces valeurs.
La seconde partie de cette description consiste en un paramétrage des composants sélectionnés à l'aide des temps de latences, lorsque ceux-ci sont connus. Par temps de latences, nous entendons par exemple le temps de traversée du système d'exploitation. Pour finir, ces blocs services sont également mis en base.

Nous venons de voir comment nous sélectionnons la partie comportementale d'un composant SystemC et la configurons avec les propriétés entrées dans le modèle architectural par l'utilisateur. Nous allons maintenant voir comment nous générons l'interface de communication.

© 2012 TELECOM ParisTech

5.4.2 Génération d'une interface générique : le container matériel

Afin de rendre générique et ainsi simplifier la génération de l'interface de chaque composant, nous avons développé un composant générique que nous appelons le container matériel. Ce composant a deux objectifs : assurer la communication entre simulateur et le composant en se basant sur le protocole défini dans TLM, et définir la politique d'abonnement de chaque composant de la plate-forme aux stimulus applicatifs.

5.4.2.1 Rappel du principe de communication TLM

La bibliothèque TLM de SystemC offre, comme nous l'avons vu dans le chapitre 3, des fonctions permettant la communication entre composants SystemC par échange de trames. Ces échanges se font suivant le principe de communication initiateur-cible et reposent principalement sur les éléments suivants : deux types d'interface de communication, une structure contenant les informations de la trame et un protocole de communication qui met en relation les deux premiers éléments. Lorsqu'un composant souhaite envoyer une trame à une cible, il remplit la structure avec diverses informations que nous allons voir et fait appel à l'interface de communication de ce composant cible pour envoyer cette structure. Cette interface reçoit la trame et vient ensuite stimuler l'automate du composant.

Les interfaces de communications sont au nombre de deux : les interfaces *nb_transport* et *b_transport*. La première, *nb_transport*, autorise des communications suivant un protocole en plusieurs phases de type "requête - acquittement - réponse - acquittement". La seconde, *b_transport*, résume chaque transaction à une communication simple de type "requête-réponse". Ce type de communication est généralement utilisé lors de modélisation de plate-forme d'exécution à haut niveau d'abstraction, où l'implémentation détaillée des échanges n'est pas la priorité. Nous utilisons ce dernier type d'interface dans notre méthode de modélisation.

TLM introduit également une structure regroupant des propriétés relatives à chaque transaction, la *generic_payload*. Initialement, cette structure a été développée pour définir les propriétés spécifiques aux échanges initiateur-mémoire via le bus mémoire, avec des propriétés comme l'adresse mémoire ou la commande (lecture/écriture).

© 2012 TELECOM ParisTech

Dans notre cas, parmi toutes les propriétés présentes, nous utilisons les suivantes :

- La commande envoyée via la trame. TLM définit les commandes suivantes : *TLM_READ_COMMAND, TLM_WRITE_COMMAND, IGNORE*. Toutefois, nous verrons que nous avons étendu le jeu de commandes en profitant du mécanisme d'extension de la trame TLM ;
- L'adresse du composant cible ;
- Le statut de la requête (requête bien traitée, en attente, erreur lors du traitement etc.). Là encore, TLM définit plusieurs statuts, parmi lesquels : *TLM_OK_RESPONSE, TLM_ERROR_RESPONSE ou encore TLM_INCOMPLETE_RESPONSE*.

A cela s'ajoute un mécanisme d'extension de la structure, qui autorise l'ajout de paramètres. Nous avons profité de cette possibilité afin d'étendre les propriétés existantes, comme les commandes envoyées, ou en ajouter d'autres. Au final, la structure embarque les propriétés nouvelles suivantes :

- *src* qui contient l'adresse du composant initiateur ;
- *stimulus* qui contient le nom du stimulus envoyé au composant cible ;
- *type* qui permet de différencier les trames "services" des trames "matérielles". Cette distinction est utile lors de l'acheminement des trames au sein du modèle, et lors de la phase de routage ;
- *service_addr* qui contient, dans le cas d'une trame de type "service", l'identifiant du service cible.

Ces paramètres sont utilisés par notre méthode de simulation des composants SystemC, mais sont invisibles à l'utilisateur. En effet, l'un de nos objectifs est de générer et simuler la plate-forme d'exécution SystemC en toute transparence pour l'utilisateur. La seule exception est dans le cas où l'utilisateur souhaite ajouter un composant comportemental à la base de données. Dans ce cas, lors du développement du composant, celui-ci doit utiliser certaines fonctions de communication que nous avons définies dans le container matériel. Chaque composant de la plate-forme d'exécution étant rattaché à un container, toutes les communications se font via ce composant.

© 2012 TELECOM ParisTech

5.4.2.2 Le Container : principe

Le container fonctionne comme une enveloppe attachée à chaque composant de la plate-forme d'exécution. Il contient les fonctions de communication nécessaires à l'envoi et réception de trames par les parties comportementales. Afin d'éviter à l'utilisateur de rentrer dans les spécificités des communications sous TLM, nous avons développé un ensemble de fonctions comme une sur-couche à TLM, afin d'en abstraire les mécanismes de communication. Ainsi, lorsqu'il développe un composant comportemental, il suffit à l'utilisateur d'utiliser les fonctions de cette sur-couche. Nous donnons ci-après une liste de ces fonctions :

- *get_frame_properties* qui retourne les propriétés de la structure *generic_payload*, extensions comprises ;
- *set_frame_properties* qui permet de définir les valeurs de ces propriétés avant d'envoyer une trame ;
- *send_ok_response* qui retourne au composant initiateur un acquittement de transaction ;
- *send_error_response* qui avertit le composant initiateur d'une erreur lors de l'exécution de la trame reçue ;
- *send_frame_to_component* qui envoie la trame au composant cible ;
- *forward_frame* qui, dans le cas d'un bus par exemple, relaye la trame reçue vers le composant cible via le bon port ;
- Et toutes les fonctions d'accès aux différentes propriétés de la trame, comme *get_dest_addr, get_type, set_command, set_status,* etc.

Afin de simplifier également l'encapsulation de la partie comportementale dans le container, nous avons développé un module permettant de générer les contours de cette partie comportementale (déclaration de fonctions et variables internes), afin qu'il ne reste à l'utilisateur qu'à décrire le comportement en lui-même. La seule obligation pour lui est de décrire cela dans le corps de la fonction *b_transport*, qui reçoit directement les trames depuis le container. Le pseudo code suivant donne un aperçu des fonctions générées lors de la création du composant. Celle-ci initialise la configuration du composant (récupération des valeurs des paramètres), connecte le composant au container, puis génère l'interface de la fonction permettant les communications et la description du comportement.

 © 2012 TELECOM ParisTech

```
/*constructeur*/
component::component(module_name, container mon_container)
{

        //initialisation des variables
        get_property(property_name1); -> récupère la valeur de property_name1
        get_property(property_name2); -> récupère la valeur de property_name2


        //connexion avec le container
        connect(mon_container);
}

/*partie comportementale*/
component::b_transport(trame,temps_courant)
{

        //récupération des propriétés et données de la trame
        c->get_properties(trame); -> stimuli, type de la trame, etc.

        //comportement du composant

                /* code utilisateur */
                        ...
                        ...
                        ...

                /*******************/

}
```

© 2012 TELECOM ParisTech

5.4.2.3 Le Container : mécanisme d'abonnement et filtrage

D'autre part, nous avons développé dans notre modèle de communication un mécanisme d'abonnement aux stimuli. Celui-ci est également intégré au sein du container. Il indique quels stimuli sont traités par le composant attaché au container. L'objectif de ce mécanisme est double. D'une part, une phase de création des tables de routage est effectuée au terme de la phase d'initialisation, avant que la simulation ne soit lancée. Cette phase s'appuie sur les stimuli définis dans chaque container. Cette phase de routage s'assure qu'aucune trame n'est envoyée vers un mauvais composant.

D'autre part, le mécanisme d'abonnement offre un moyen de découpler la définition des stimuli de la définition des composants. Prenons l'exemple d'une mémoire DRAM : admettons que celle-ci puisse traiter les stimuli "data_load", "data_read" et "logbook_write". Le container attaché à l'automate de la mémoire se verra abonné à ces trois stimuli. Supposons maintenant que nous ajoutions une mémoire annexe pour le stimulus "logbook_write". Le stimulus sera enlevé du container attaché à la mémoire DRAM, et associé au container attaché à la mémoire annexe. La partie comportementale de la mémoire principale n'est pas modifiée. Une nouvelle phase de création des tables de routage est simplement effectuée afin de prendre en compte cette modification.

La figure 5.7 représente le mécanisme d'abonnement aux trames : lorsque le container reçoit une trame, celui-ci extrait les propriétés de la trame et vérifie que le stimulus associé à la trame est présent au sein de la liste des stimuli acceptés par le composant. Dans ce cas, il relaye la trame vers la fonction *b_transport* de la partie comportementale, qui la traite. Si le composant est le composant cible de la trame, il exécute le stimulus et renvoie une réponse. S'il n'est pas le composant cible, mais un composant transitoire, il relaie la trame vers le bon port à destination du composant cible.

© 2012 TELECOM ParisTech

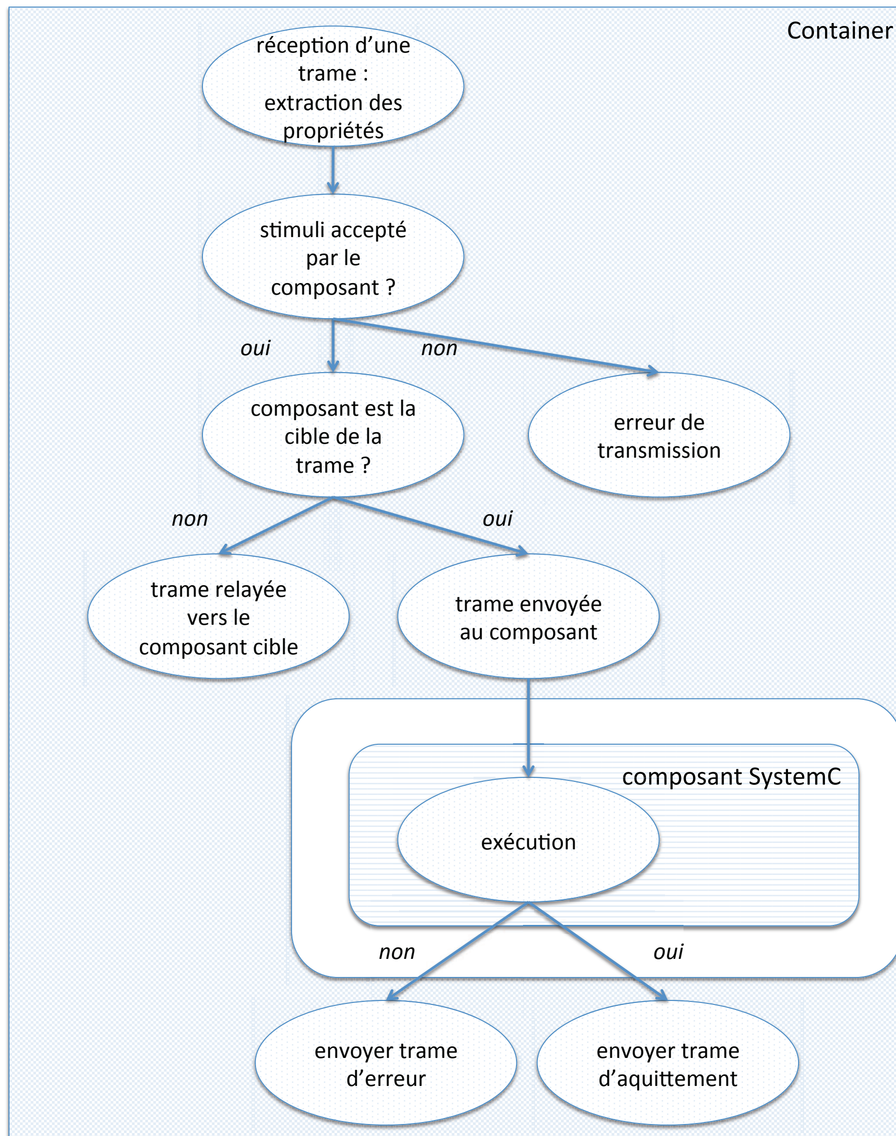

FIGURE 5.7 – Traitement des communications par le couple container-composant

Pour résumer, nous avons un composant générique qui :

- Encapsule la partie comportementale définie par l'utilisateur ;
- Offre des fonctions de communications simplifiées, venant abstraire celles fournies par TLM. L'utilisateur ne doit donc pas étudier en profondeur TLM et ses spécificités ;
- Offre un mécanisme de routage et filtrage automatique des trames, afin d'assurer la validité de la communication entre le simulateur et les composants.

© 2012 TELECOM ParisTech

5.4.3 Intégration

Nous l'avons vu dans le chapitre 2, l'objectif de notre méthode est de permettre une représentation des spécifications de la plate-forme qui aille au delà des représentations brutes sous forme de valeurs stockées dans un tableur. Dans notre cas, le modèle AADL permet à l'utilisateur de décrire l'architecture de la plate-forme, et d'extraire les configurations matérielles et logicielles en vue de la génération du modèle SystemC. En effet, le modèle architectural contient les éléments suivant, que nous utilisons :

- Les caractéristiques principales des éléments logiciels (partitions et tâches) ;
- Les propriétés d'ordonnancement de ces éléments ;
- La liste des services API ARINC653 (ou APEX) utilisés ;
- Les caractéristiques des éléments matériels ;
- Les informations de connexion de ces éléments ;
- Le déploiement des éléments logiciels sur les éléments matériels.

Le passage du niveau architectural vers le niveau comportemental pour la description de la plate-forme d'exécution (services APEX et partie matérielle) se fait ainsi en utilisant deux sources d'informations : celles du modèle architectural que l'on vient de voir, et la base d'automates comportementaux SystemC. Pour les éléments matériels et services APEX du modèle AADL, la règle du "un pour un" est appliquée : à chacun d'entre eux, un container est généré auquel sont associées leurs propriétés. Ces containers sont connectés suivant les connexions extraites du modèle AADL pour les éléments matériels, tandis que les services APEX sont connectés au processeur principal. Par cette opération, nous générons un équivalent du modèle architectural.

Ce modèle est ensuite enrichi afin d'atteindre une description comportementale. Les automates correspondant aux éléments de la plate-forme d'exécution sont connectés à leur container, avant d'être automatiquement configurés par les propriétés extraites du modèle AADL. La figure 5.8 résume ce processus de transformation.

 © 2012 TELECOM ParisTech

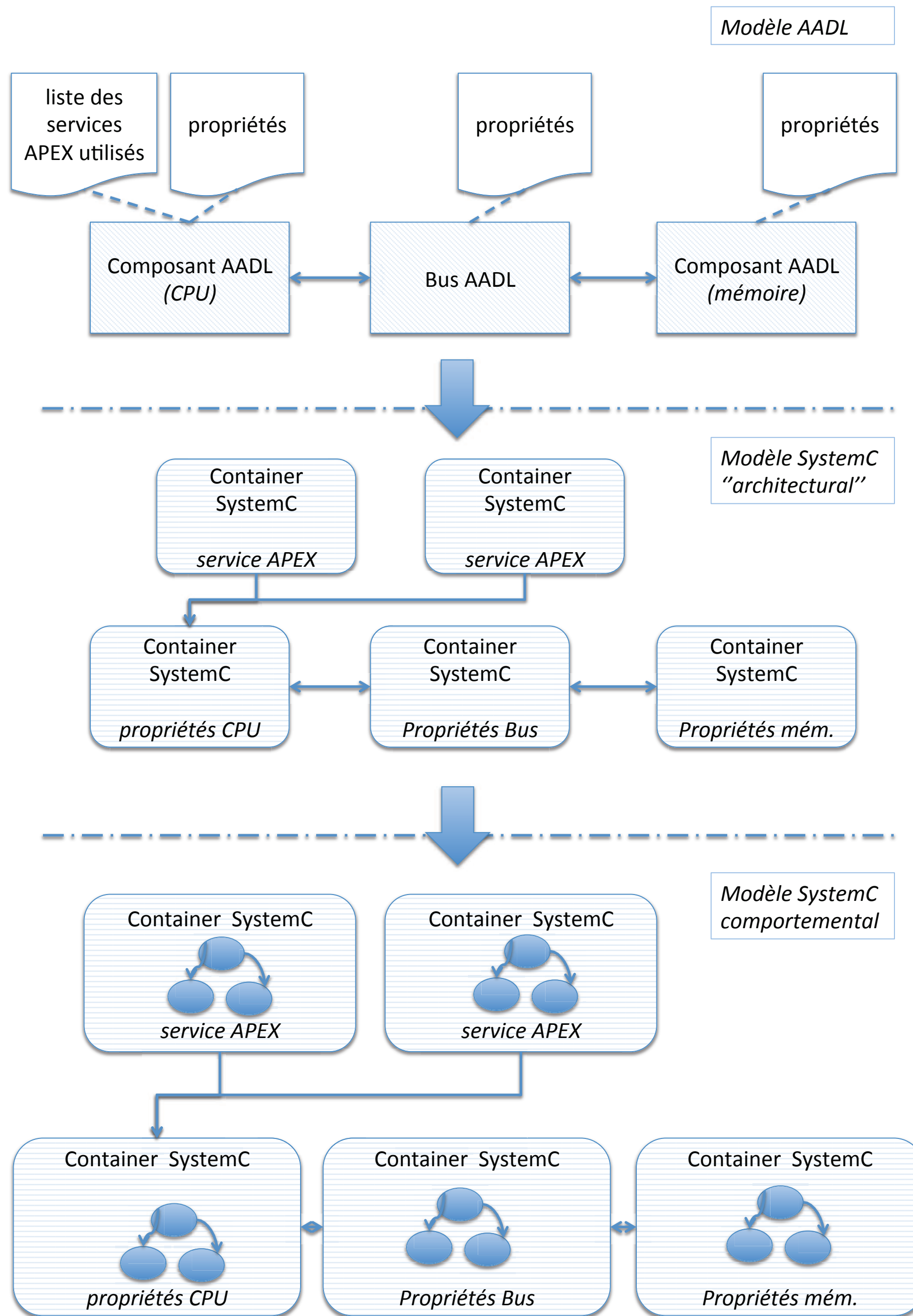

FIGURE 5.8 – Processus de transformation du modèle architectural AADL vers le niveau comportemental SystemC

© 2012 TELECOM ParisTech

5.5 Synthèse

Nous avons vu dans cette section comment nous améliorons la description d'une plate-forme, allant plus loin que la vue "tableur" ou que les premiers modèles architecturaux proposés dans d'autres travaux. Nous générons un modèle de plate-forme comportementale SystemC-TLM à partir d'une description architecturale AADL, en préservant ses propriétés. Dans un premier temps, nous nous sommes appuyés sur l'annexe ARINC653 du langage AADL, qui permet la modélisation de plates-formes avioniques avec ses spécificités. Partant de cela, nous avons ajouté des propriétés dans les composants matériels AADL afin d'aider à la sélection du modèle comportemental SystemC correspondant en base de données. Une phase d'extraction des propriétés de chaque composant matériel du modèle permet d'effectuer cette sélection, puis de configurer la description comportementale avec les propriétés du composant. La phase d'extraction des paramètres de connexions s'appuie quant à elle sur la partie suivante. Nous avons développé un composant standard générique en SystemC-TLM, le Container, qui vient encapsuler chaque partie comportementale générée. Ce composant offre des fonctions de communications, sur-couche aux fonctions spécifiques à TLM. Il sert ainsi d'interface de communication, et assure le bon déroulement des échanges de trames entre les composants du modèles, mais également entre le simulateur et le processeur, point d'entrée pour les trames envoyées à la plate-forme d'exécution.
En résumé, ces travaux ont permis de résoudre les points techniques suivants :

- Enrichir le modèle architectural AADL d'une plate-forme d'exécution vers un modèle comportemental SystemC, qui autorise des analyses dynamiques via sa simulation ;
- Intégrer les spécificités avioniques, principalement le partionnement spatial et temporel. Ceci est fait en se basant sur les travaux de l'annexe ARINC653 ;

 © 2012 TELECOM ParisTech

Chapitre 6

Méthode de génération des stimuli applicatifs en avance de phase

Sommaire

6.1 Introduction **102**

6.2 Modélisation d'une application en avance de phase . . . **102**

6.2.1 Principe 102

6.2.2 Description des tâches applicatives 103

6.3 Génération des stimuli applicatifs **104**

6.3.1 Principe 104

6.3.2 Adaptation aux variantes architecturales de la partie matérielle de la plate-forme 105

6.4 Synthèse **108**

6.1 Introduction

Notre première contribution concerne la modélisation de la partie applicative en avance de phase, et la génération d'un scénario de simulation. Le principal point technique vient, nous l'avons vu précédemment, de la pauvreté de l'information présente. Soit l'application n'est pas encore totalement développée, soit le code source de l'application n'est pas accessible car propriété du développeur d'applications. Dans ce contexte, il nous a fallu réfléchir à une solution de modélisation de l'application malgré l'absence du code source, qui interdit les méthodes actuelles de parcours de code pour en extraire les informations pertinentes. Dans un second temps, il nous a fallu développer une méthode permettant de générer des stimuli applicatifs depuis cette représentation de la partie applicative, pour venir stimuler la plate-forme d'exécution SystemC.

6.2 Modélisation d'une application en avance de phase

6.2.1 Principe

Une application avionique est, au sein de la plate-forme d'exécution, découpée en plusieurs partitions, elle-même contenant les tâches applicatives. C'est ce que nous avons appelé dans le chapitre 4 la configuration applicative. En outre, une application avionique contient deux types d'instructions : les appels aux services de l'API ARINC653, et les instructions dites de calcul. Nous nous sommes appuyés sur cela pour développer une méthode de description de la partie applicative par blocs algorithmiques et instructions d'appels aux services de l'API ARINC653. Suivant le niveau d'information disponible, les blocs algorithmiques servant à modéliser la partie applicative peuvent être de niveaux de précision différents. Nous avons des blocs de niveau fonctionnel, c'est-à-dire représentant des fonctions de type FFT par exemple, et des blocs de niveau instruction, de type calcul d'une valeur ou stockage d'une variable. SI la partie applicative est définie avec des blocs fonctionnels, la précision sera moindre, mais cela permettra toutefois d'effectuer une modélisation de l'applicatif et donc de poursuivre le processus de modélisation avec cette base.

© 2012 TELECOM ParisTech

Reprenons l'exemple de la FFT : dans le cas où l'application à modéliser fait appel à une FFT, deux solutions s'offrent à l'utilisateur.

- Utiliser le bloc fonctionnel FFT, avec une implémentation générique de la FFT qui ne sera pas forcément l'algorithme développé dans l'application ;
- Décomposer la FFT en sous-blocs. Dans ce cas, la FFT sera décrite comme un ensemble d'opérations (affectation, calcul, etc) correspondant à l'algorithme utilisé.

Dans notre processus, nous avons développé des blocs algorithmiques de type "instruction", car nous avons essentiellement adressé des applications existantes dont la mise en oeuvre n'est pas encore accessible pour des raisons de confidentialité. Ainsi, en fournissant une bibliothèque de blocs algorithmiques au développeur d'applications, ce dernier peut fournir une description de son application sans fournir le code source ou le binaire, et sans donner de nom de fonctions afin d'éviter toute opération de rétro-engineering.

6.2.2 Description des tâches applicatives

Chaque application est exécutée au sein d'une ou plusieurs partitions par des tâches logicielles. Nous avons pour cela un fichier de configuration reprenant les propriétés de chaque partition (position dans la MAF, durée), et les propriétés des tâches (période, deadline, priorité) vues dans le chapitre 1. Ce fichier référence aussi pour chaque tâche un fichier de description algorithmique associé : nous avons donc, pour une application, plusieurs fichiers correspondant aux tâches applicatives. Chaque fichier de description avec les blocs algorithmiques et les appels services ressemble à la description suivante :

```
affectation
affectation

appel service RECEIVE_BUFFER

calcul(deux entiers)
calcul(deux entiers)
calcul(trois entiers)
affectation
```

© 2012 TELECOM ParisTech

```
calcul(deux entiers)

appel service SEND_BUFFER

appel service PERIODIC_WAIT
```

Cette description de la partie applicative va être utilisée pour deux opérations. Premièrement, la configuration est utilisée pour modéliser la partie applicative du modèle AADL que nous avons vu. Deuxièmement, les fichiers de description vu ci-dessus sont utilisés pour la génération du scénario de simulation de la plate-forme d'exécution SystemC. Nous allons décrire cette opération dans la sous-section suivante.

6.3 Génération des stimuli applicatifs

6.3.1 Principe

En l'absence du code de l'application, l'étape précédente était nécessaire afin de servir de base pour la génération des stimuli applicatifs. La méthode que nous avons développée consiste en un ensemble de patrons de générations de stimuli. Un patron consiste en une ou plusieurs règles de génération de stimuli qui récupèrent les paramètres des blocs algorithmiques précédents. Par exemple, un bloc de calcul aura pour paramètres le type des données (entiers ou flottants) et éventuellement un nombre plus ou moins approximatif des variables concernées par le calcul. A partir de cela, le patron de génération de stimuli créera autant de stimuli d'accès mémoire en lecture (chargement de données) que de nombre de variables, ainsi que des stimuli d'accès au processeur (calcul en lui-même) et d'accès en écriture à la mémoire (stockage du résultat).
Pour chaque fichier de description décrit ci-dessus, nous parcourons la description algorithmique. A chaque ligne du fichier, nous appliquons un patron de génération, qui prend le cas échéant le/les paramètres présents. Dans le cas des services API, le patron récupère simplement le nom du service et créé un stimuli qui cible ce service. La figure 6.1 donne un aperçu de la méthode de génération des stimuli via des patrons de génération : un parcours de la description algorithmique de la tâche est effectué, et les patrons correspondant aux différents blocs sont appliqués pour générer les stimuli applicatifs correspondants.

© 2012 TELECOM ParisTech

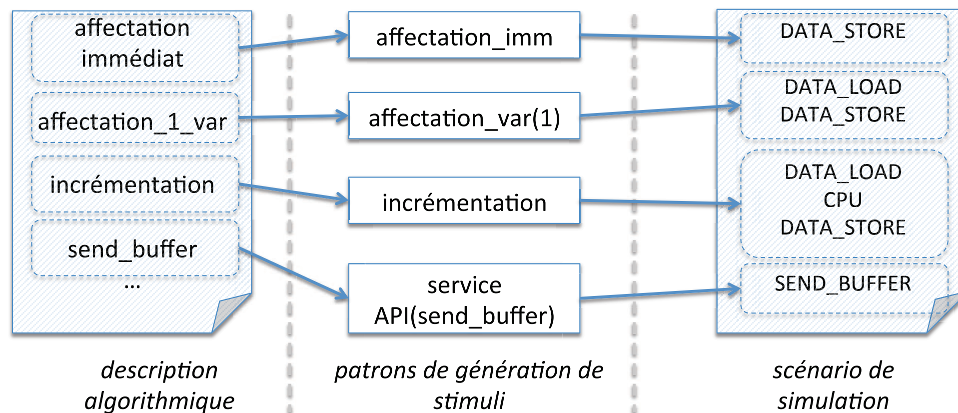

FIGURE 6.1 – Méthode de génération de stimuli par patrons de génération

6.3.2 Adaptation aux variantes architecturales de la partie matérielle de la plate-forme

Dans le cas où l'on souhaite tester plusieurs architectures pour une même partie applicative, les stimuli à générer peuvent varier. En effet, suivant les variantes d'architecture, des composants peuvent être ajoutés ou enlevés de la plate-forme d'exécution. La méthode de patrons de génération de stimuli associée au mécanisme d'abonnement aux stimuli vu dans la sous-section précédente permet de s'adapter rapidement à ces changements : la méthode de modélisation de la partie applicative par blocs algorithmiques, qui constitue l'entrée des patrons de génération de stimuli, ne change pas ; seuls les stimuli générés peuvent être amenés à changer. Ainsi, lorsque l'utilisateur souhaite modifier son architecture, deux possibilités s'offrent à lui :

- Soit les composants ne changent pas, seules certaines de leurs propriétés ont varié : dans ce cas les stimuli ne changent pas. C'est également le cas lorsque seuls des composants de communications de type bus changent au sein de l'architecture ;
- Soit les composants (hors bus) varient, auquel cas les patrons doivent être adaptés avec l'intégration / la suppression des stimuli du/des composants.

Le schéma 6.2 représente le premier cas : l'architecture change par ajout d'un composant intermédiaire, un gestionnaire des entrées-sorties pour optimiser leur traitement ("I/O Processor"). Ce composant n'est pas ciblé directement par un stimulus, et

© 2012 TELECOM ParisTech

aucun autre composant n'a été enlevé ou ajouté. Ainsi, même si l'architecture change, l'ensemble des stimuli utilisés pour stimuler la plate-forme ne change pas.

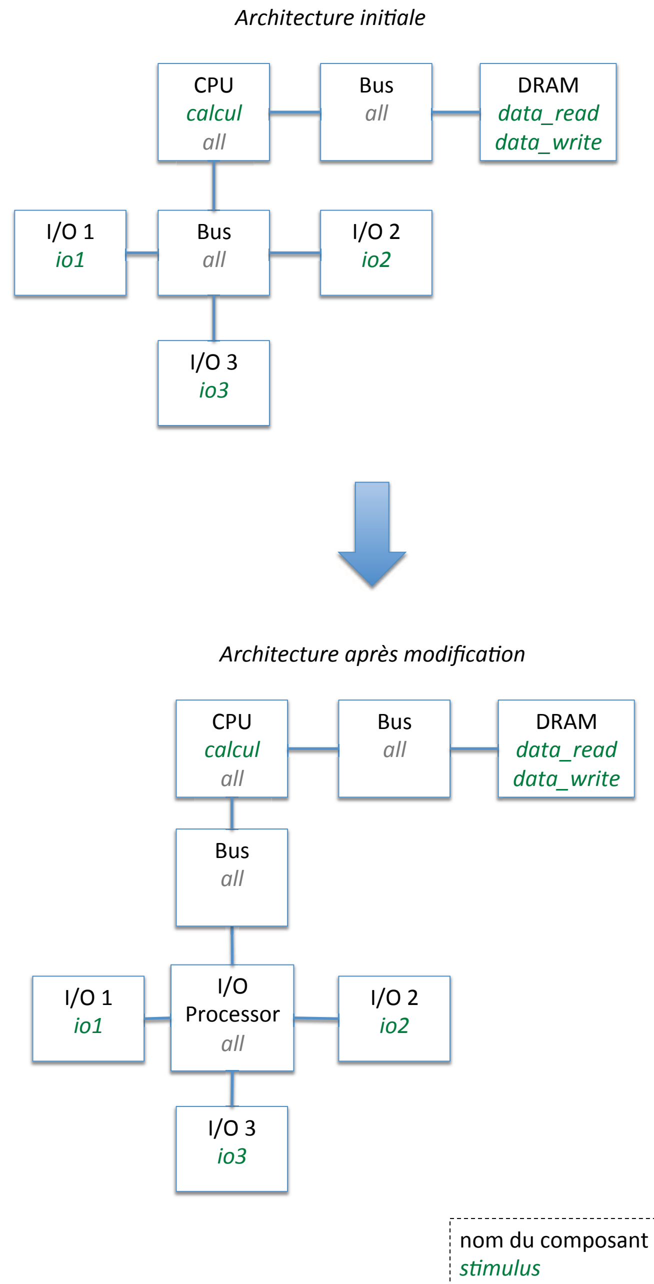

FIGURE 6.2 – Modification de l'architecture sans modification du jeu de stimuli

© 2012 TELECOM ParisTech

Le schéma 6.3 présente le second cas : un composant mémoire, "Flash", a été ajouté à l'architecture. Ce composant ajoute un stimulus au jeu de stimuli ("data_flash"), qui doit être intégré dans un ou plusieurs patrons de génération de stimuli.

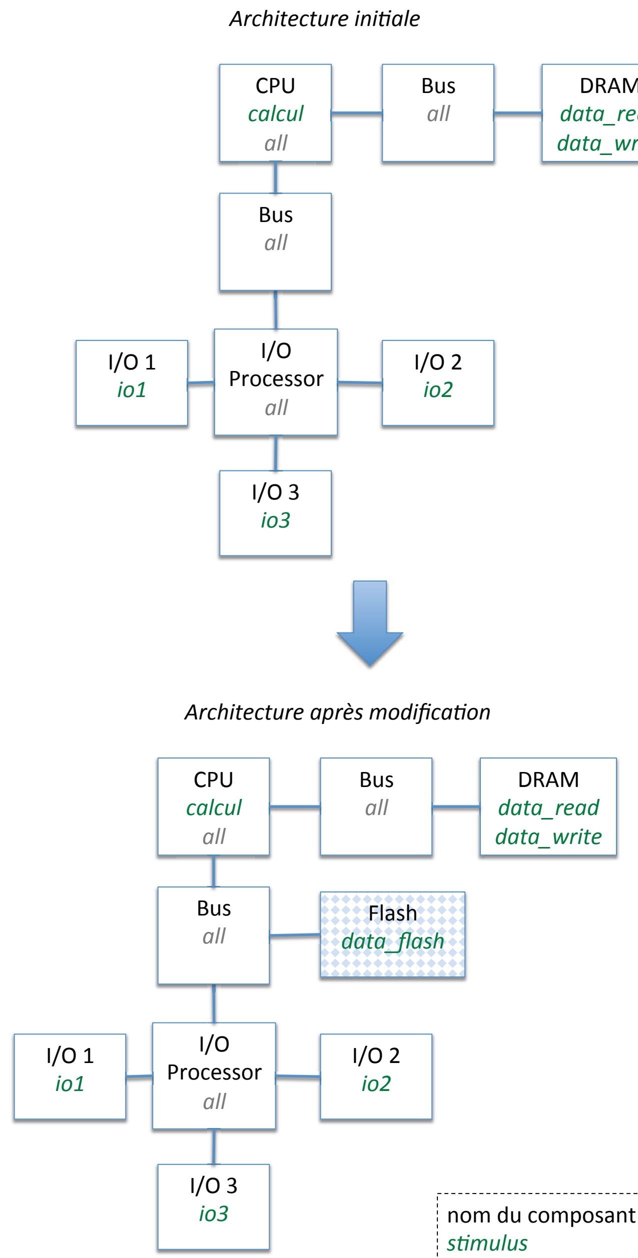

FIGURE 6.3 – Modification de l'architecture avec modification du jeu de stimuli

© 2012 TELECOM ParisTech

Comme pour la base de données de parties comportementales, la base de données de patrons de génération de stimuli a pour but d'être enrichie, et ainsi de permettre la réutilisabilité des patrons, en se basant sur le fait qu'une architecture est souvent une évolution d'une architecture précédente.

6.4 Synthèse

Nous avons vu dans cette sous-section comment nous modélisons la partie applicative, et comment nous générons le scénario de simulation de la plate-forme d'exécution SystemC. L'utilisation de blocs algorithmiques pour la description de la partie applicative permet au développeur d'applications de fournir une modélisation de ces dernières sans toutefois livrer un code permettant de remonter au code source. Ceci nous donne la possibilité de travailler sur une description relativement précise, malgré l'absence du code source.

Cette description algorithmique sert de base à la seconde étape décrite dans cette partie, à savoir la génération des stimuli applicatifs. L'utilisation de patrons de génération de stimuli correspondant aux blocs algorithmiques précédents permet de tester plusieurs architectures pour une même partie applicative, sans avoir à modifier la modélisation de celle-ci. Seuls les patrons, dans le cas de changements importants comme l'ajout ou le retrait de composants, ont éventuellement besoin d'être adaptés. Dans le chapitre suivant, nous allons mettre en oeuvre le processus de modélisation sur deux cas d'étude.

© 2012 TELECOM ParisTech

Quatrième partie

Validation

Chapitre 7

Expérimentations et Résultats

Sommaire

7.1 Introduction 112
7.2 Cas d'Etude 112
7.2.1 Calculateur avionique 112
7.2.2 Plate-forme avionique 113
7.3 Mise en oeuvre du processus sur un calculateur 114
7.3.1 Description des applications 115
7.3.2 Modélisation architecturale du calculateur 117
7.3.3 Génération de la plate-forme d'exécution en SystemC . . 118
7.3.4 Simulation et analyse des résultats 120
7.3.5 Résumé 122
7.4 Mise en oeuvre du processus sur une plate-forme 123
7.4.1 Présentation de la plate-forme de test 123
7.4.2 Modélisation du réseau ARINC664 124
7.4.3 Modélisation des services de l'API ARINC653 125
7.4.4 Modélisation d'un calculateur au sein de la plate-forme . 126
7.4.4.1 Modélisation architecturale 126
7.4.4.2 Modélisation comportementale 127
7.4.5 Simulation et analyses des résultats 128
7.4.5.1 Analyse des temps d'exécution 128
7.4.5.2 Temps de simulation 129
7.4.5.3 Extraction de l'utilisation des ressources matérielles 130
7.5 Synthèse 132

7.1 Introduction

Dans ce chapitre nous allons mettre en oeuvre notre processus de modélisation - simulation et voir comment il permet de répondre à la problématique globale exprimée dans le chapitre 1. Lors cette mise en oeuvre, nous décrirons pour chaque étape les parties qui s'appuient sur des travaux existants, ainsi que les ajouts, modifications ou parties nouvelles qui constituent nos contributions. Nous verrons pour chacune d'entre elles comment elles répondent aux points techniques évoqués dans la problématique du chapitre 2. Dans un premier temps, nous allons décrire les cas d'étude utilisés pour la validation expérimentale de nos travaux. Ensuite, nous déroulerons notre processus sur ces deux cas d'étude. L'architecture logicielle et matérielle utilisée sera décrite dans chacune des sections concernées. L'objectif du premier cas d'étude sera de montrer l'aspect fonctionnelle et valide de l'approche, tandis que le second montrera en quoi les choix effectués au sein de nos contributions répondent aux points techniques.

7.2 Cas d'Etude

7.2.1 Calculateur avionique

Architecture matérielle : lors de nos expérimentations, nous avons utilisé un calculateur avionique. Dans un premier temps, nous n'avons utilisé que le coeur du calculateur. Celui-ci contient un processeur mono-coeur MPC8610, dédié à l'exécution d'applications au sein d'un système embarqué, une mémoire cache de niveau L1 ainsi qu'une mémoire cache de niveau L2, les deux intégrées dans le coeur. Le processeur est relié à une mémoire DDR2 via un bus mémoire. Le calculateur contient également un accès au réseau avionique via un bus PCI et une mémoire Flash, que nous n'utilisons pas ici.

Architecture logicielle : les trois applications utilisées pour tester notre méthode dans ce premier cas consistent en des applications de calcul. Plus précisément, nous avons une application de type FFT, un calcul de Pi et une application de calcul simple assez similaire à la FFT. Ces applications ont l'avantage d'utiliser la plate-forme matérielle sans passer par les services de l'APEX. Elles ne font également pas appel aux entrées-sorties, et n'utilisent donc pas le réseau avionique. Le schéma 7.1 présente le calculateur utilisé pour cette phase d'expérimentations.

© 2012 TELECOM ParisTech

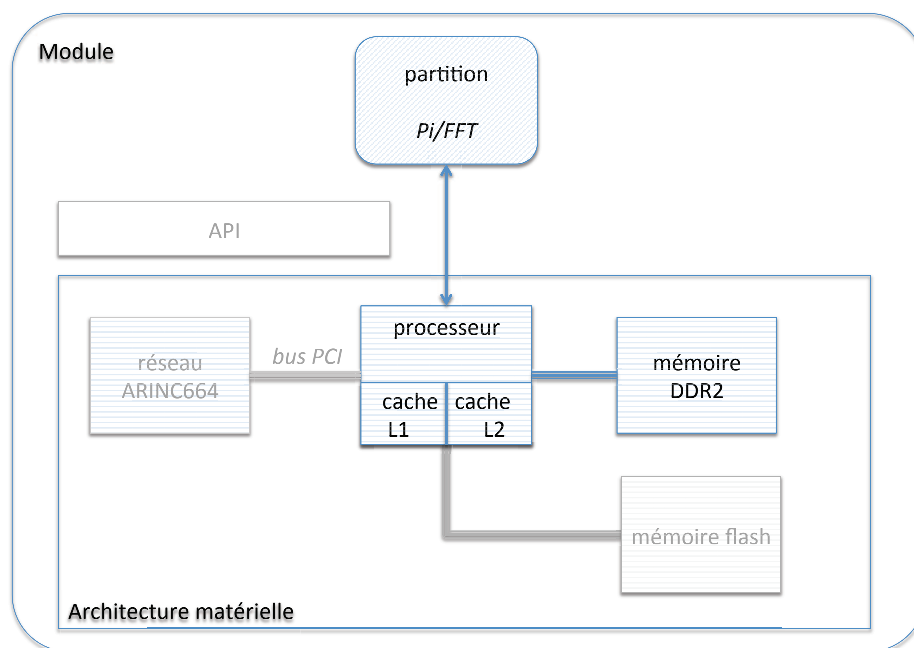

FIGURE 7.1 – Architecture du coeur du calculateur avec partition applicative

7.2.2 Plate-forme avionique

Architecture matérielle : dans un second temps, nos expérimentations se sont portées sur une plate-forme avionique plus complète. Celle-ci est constituée de deux instances du calculateur décrit par la figure 7.1, reliés par un réseau avionique ARINC664.

Architecture logicielle : lors de cette phase d'expérimentations, nous avons utilisé deux applications avioniques. La première consiste en une application avionique utilisée pour les tests réseaux. Cette application est répliquée sur deux partitions, une sur chaque calculateur. Elle est constituée d'un code effectuant du calcul et des appels à trois services ARINC653 : *receive_queuing_port, send_queuing_port* et *periodic_wait*. Les deux premiers permettent de lire les données présentes sur le port en question, et d'écrire une/des données sur ce port. Le dernier service est utilisé pour mettre la tâche en pause en attendant sa prochaine activation. La seconde application utilisée pour tester notre méthode est similaire à la première. Elle contient les mêmes appels aux services de l'APEX et des phases de calcul, mais est répartie dans trois partitions.

© 2012 TELECOM ParisTech

Nous avons une plate-forme composée de deux calculateurs, sur lesquels s'exécutent deux partitions pour le premier cas d'utilisation présenté par la figure 7.2, trois partitions pour le second cas de figure.

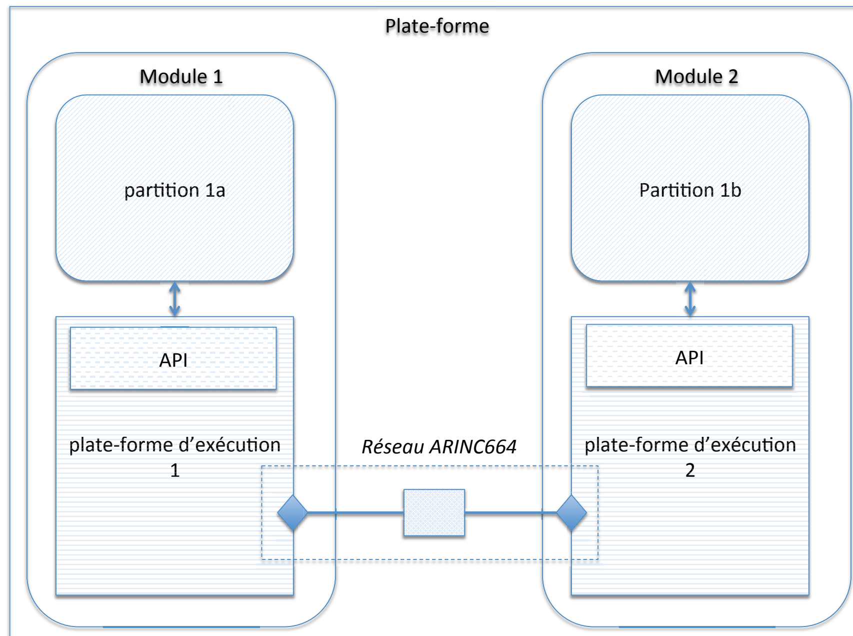

FIGURE 7.2 – Architecture de la plate-forme de test

Nous allons désormais détailler la mise en oeuvre de notre processus sur ces configurations.

7.3 Mise en oeuvre du processus sur un calculateur

La première étape de validation expérimentale de notre processus consiste en une mise en oeuvre sur un calculateur avionique exécutant une application. L'objectif est ici de montrer que la méthode est fonctionnelle et utilisable en pratique. Ce cas d'étude nous a en outre servi à valider notre approche sur un cas d'étude relativement simple, avant de le déployer sur une plate-forme plus complexe.

© 2012 TELECOM ParisTech

7.3.1 Description des applications

Nous avons donc modélisé le calculateur seul, sans système d'exploitation, sur lequel sont exécutées les applications une à une, sans dépendance.

Nous avons appliqué sur ces applications notre méthode de modélisation par description algorithmique. Nous allons prendre une partie de l'application FFT utilisée. Ci-dessous, une partie du code initial développé par l'utilisateur :

```
temp=(out13−out9)*(SIN_2PI_16);
out9=out9*(C_P_S_2PI_16)+temp;

out14*=(SIN_4PI_16);
out10*=(SIN_4PI_16);
```

Et maintenant un exemple de description spécifiée par l'utilisateur à l'aide de blocs algorithmiques mis à sa disposition :

```
data_load(3)
calculus(2)
data_store()
data_load(2)
calculus(2)
data_store()

data_load()
calculus()
data_store()
data_load()
calculus()
data_store()
```

Une FFT étant un code très répétitif, nous nous sommes ici volontairement limité à quelques lignes de code, qui suffisent à comprendre le mécanisme de modélisation de l'application. La première ligne de code de FFT consiste en un calcul sur des variables dont le résultat est stocké dans une autre variable. Derrière cela se cachent

© 2012 TELECOM ParisTech

trois étapes : un chargement des variables, un calcul sur ces variables et le stockage du résultat, ce qui a été décrit par l'utilisateur. Il en va de même pour tout le code et pour les autres applications utilisées (calcul de Pi et calcul intensif) que nous avons également modélisées.

Nous avons ensuite appliqué sur ces descriptions algorithmiques des patrons de génération de stimuli SystemC. Lorsque que la description fournie en entrée est suffisamment précise, comme dans notre cas, le procédé de génération des stimuli consiste principalement en une traduction des blocs algorithmiques (ici "data_load", "calculus" et "data_store") en stimuli compatibles avec le modèle SystemC. Par exemple, dans le cas du stimulus "calculus", cela revient à interagir avec le processeur principal, une ou plusieurs fois suivant le paramètre associé au stimulus. L'opération est la même pour les deux autres stimuli, qui ciblent quant eux la hiérarchie mémoire (mémoires cache et vive) via le processeur principal. Pour les deux premières lignes de la description algorithmique précédente, nous obtenons le scénario de simulation suivant :

```
data_load
//cpu -> cache L1 -> cache L2 (si L1 miss) -> DRAM (si L2 miss)

data_load
data_load
cpu
cpu
```

Ces stimuli correspondent aux stimuli acceptés par les éléments SystemC, c'est-à-dire qu'ils ont été définis dans le mécanisme d'abonnement de chaque container associé aux composants. Lorsque le container du processeur verra le stimulus "data_load", il verra qu'il n'est pas abonné directement à ce stimulus, mais abonné indirectement (i.e. il est noeud sur le chemin qui mène au composant cible). Il relaiera alors le stimulus vers le bon chemin, en l'occurrence vers la hiérarchie mémoire : il ciblera le cache L1, fera un cache hit ou miss, et dans ce dernier cas cherchera dans le L2 et éventuellement dans la mémoire principale. Dans le cas d'une requête "cpu", le container voit qu'il est la cible de ce stimulus et l'enverra au processeur qui traitera la requête (i.e. consommera du temps de calcul), avant de traiter la requête suivante.

© 2012 TELECOM ParisTech

7.3.2 Modélisation architecturale du calculateur

Comme nous l'avons vu au début de ce chapitre, la première plate-forme de test se compose d'un calculateur non connecté au réseau. L'utilisation des ressources matérielles est limitée aux composants de calcul et stockage : le processeur avec son cache L1, le cache L2, la mémoire principale DRAM et les bus de communication. Nous avons étendu les propriétés du processeur afin de prendre en compte les caches intégrés au bloc processeur. Il s'agit de booléen pour la présence ou non du cache et des latences associées.

L'APEX n'est pas utilisée et nous ne modélisons pas les services du système d'exploitation. Les applications sont directement chargées séparément en mémoire et chacune d'entre elles est exécutée séparément. Il n'y a donc pas encore de notion de partitionnement spatial ou temporel. La figure 7.3 représente le système modélisé, avec les données principales extraites des fichiers de caractérisation des éléments matériels. Les propriétés des éléments du modèle architectural AADL ne se limitent toutefois pas à celles-ci.

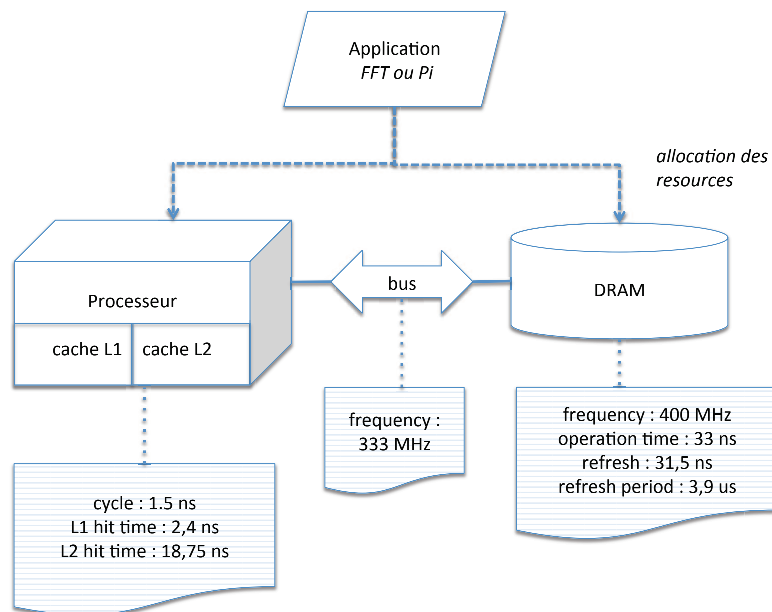

FIGURE 7.3 – Représentation architecturale du calculateur de test

Pour cette première étape, le modèle est assez simple, et ne comporte pas tous les mécanismes que nous avons décrits dans nos contributions. Par exemple, nous simulons ici uniquement les éléments génériques matériels (processor, memory, bus), sans

© 2012 TELECOM ParisTech

ajouter les propriétés spécifiant le type exact de ces composants (mémoire DDR2, processeur powerpc, etc.). De même, aucune analyse n'est menée sur ce modèle, le but étant ici de tester notre méthode de modélisation-simulation sur un premier cas concret.

7.3.3 Génération de la plate-forme d'exécution en SystemC

A partir du modèle architectural AADL précédent, nous avons généré la partie matérielle du calculateur en utilisant les automates comportementaux dits "génériques". Nous avons utilisé l'automate d'un processeur, dont la description comportementale a été affinée avec les fiches techniques, l'automate de bus et l'automate d'une mémoire vive. Chaque automate a été caractérisé par les propriétés extraites du modèle AADL, avant d'être encapsulé dans un container matériel. Ces containers ont ensuite été connectés entre eux suivant les informations de connexion du modèle AADL. Dans la figure 7.4 nous donnons un exemple de mémoire vive utilisé comme automate générique : depuis le modèle simplifié AADL, l'automate SystemC correspondant est caractérisé avec les valeurs extraites du modèle AADL, elles-mêmes extraites de la fiche technique correspondante. Les fonctions de communication qui apparaissent dans l'automate SystemC, comme "send_ok_response" correspondent aux fonctions fournies par le container pour la réception et l'envoi de trames au sein du système.

© 2012 TELECOM ParisTech

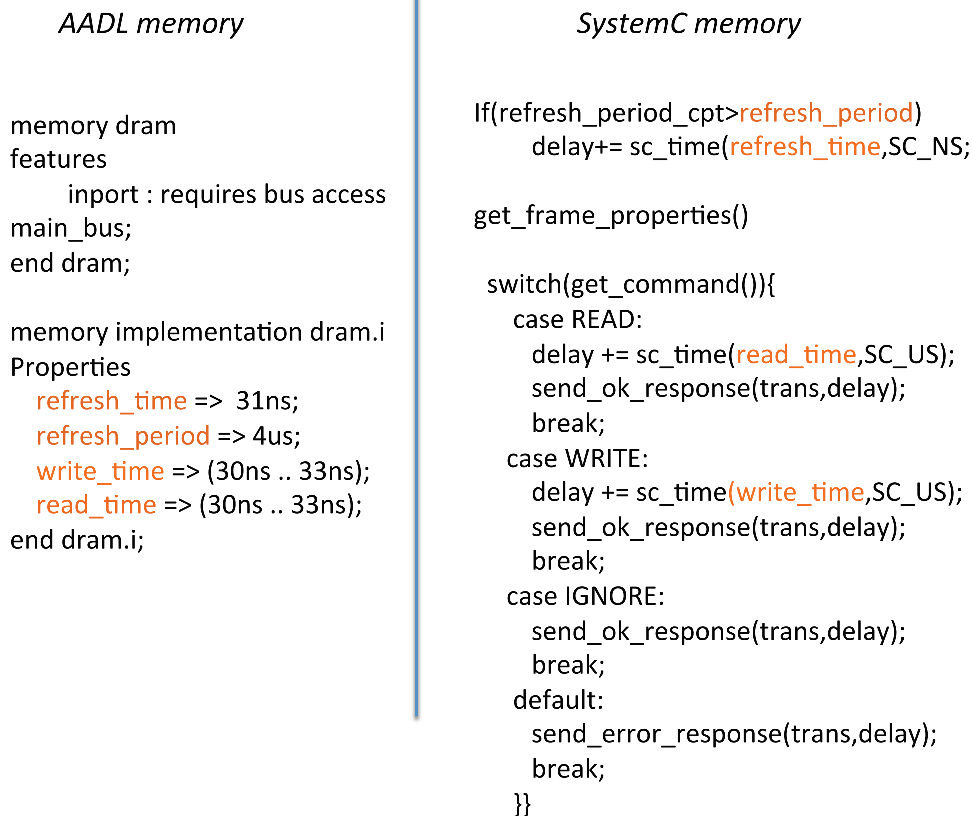

FIGURE 7.4 – Exemple de caractérisation de l'automate comportemental SystemC de mémoire à partir du modèle AADL

Nous avons exécuté le scénario de simulation, extrait dans la sous-section précédente, sur notre modèle SystemC. Chaque stimulus du scénario est exécuté suivant le principe décrit dans la sous-section précédente : grâce au mécanisme d'abonnement, chaque container sait quels sont les stimuli que peut exécuter le composant auquel il est rattaché. Après une phase de routage effectué suite à la connexion des containers, chaque stimulus est envoyé naturellement au composant cible, comme le montre la figure 7.5. Le stimulus "all" signifie que le composant accepte tous les stimuli. C'est le cas par exemple du processeur principal, par lequel tous les stimuli passent.

© 2012 TELECOM ParisTech

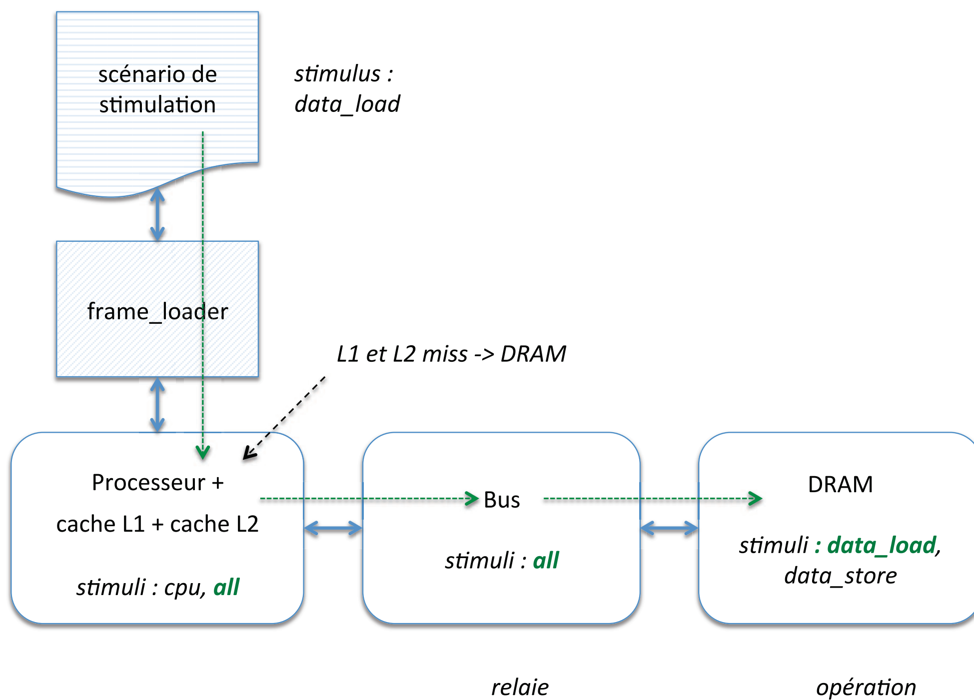

FIGURE 7.5 – Acheminement d'un stimulus vers le composant cible

7.3.4 Simulation et analyse des résultats

Dans cette dernière étape de la première partie de validation expérimentale de notre méthode, nous avons exécuté les deux scénarios extraits des applications FFT, calcul de Pi et calcul intensif. Afin de déterminer la validité des performances temporelles extraites, nous les avons comparé avec l'exécution de ces mêmes applications sur le calculateur réel. Nous présentons dans le graphe 7.6 les résultats obtenus et l'écart entre les temps d'exécution extraits sur notre modèle et ceux obtenus sur le calculateur cible. A noter que les scénarios ont été exécutés 10000 fois sur le modèle.

 © 2012 TELECOM ParisTech

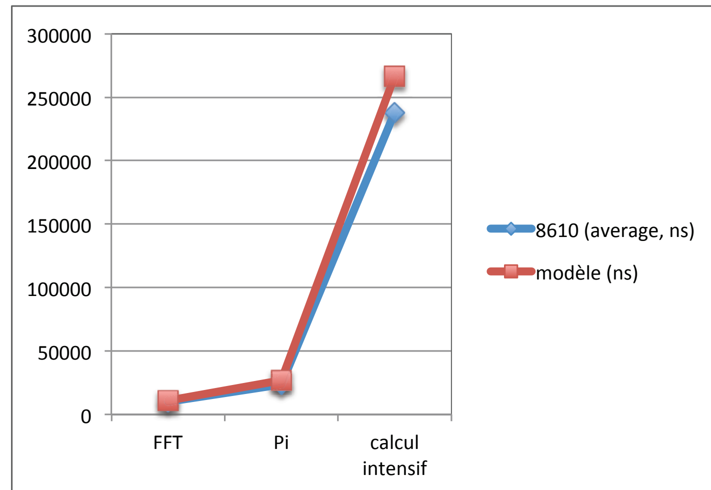

FIGURE 7.6 – Comparaison des temps d'exécution des trois applications

Les résultats 7.7 montrent que l'écart entre le temps d'exécution mesuré sur notre modèle et le temps d'exécution réel varie pour les trois applications entre 7% et 11%. L'objectif fixé dans cette première étape était un écart de moins de 15% sur ces petites applications, objectif qui est donc atteint.

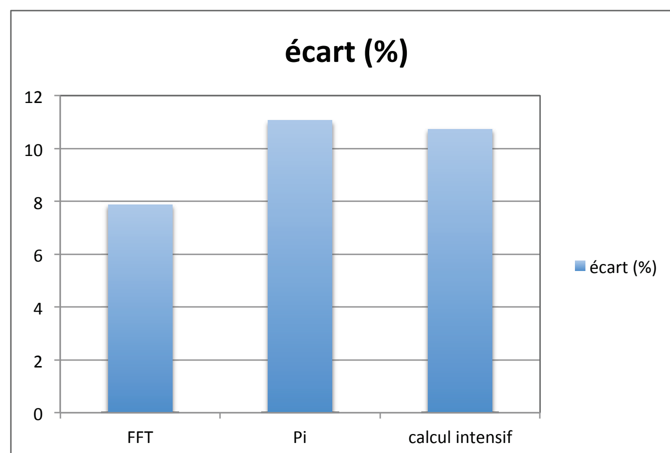

FIGURE 7.7 – Représentation de l'écart moyen entre l'exécution réelle et la simulation sur modèle des trois applications tests

© 2012 TELECOM ParisTech

Sur le graphe 7.8, nous nous sommes intéressés plus spécifiquement à la fonction calcul de Pi. Nous avons extrait de la simulation du scénario correspondant les temps d'exécution minimum, maximum et la moyenne sur les 10000 exécutions. Nous avons procédé de même pour l'exécution de l'application réelle sur le calculateur. Nous voyons dans le tableau que les valeurs extraites de la simulation sur le modèle sont non seulement cohérentes, mais permettent également de déterminer un intervalle représentatif de temps d'exécution, entre les valeurs min et max.

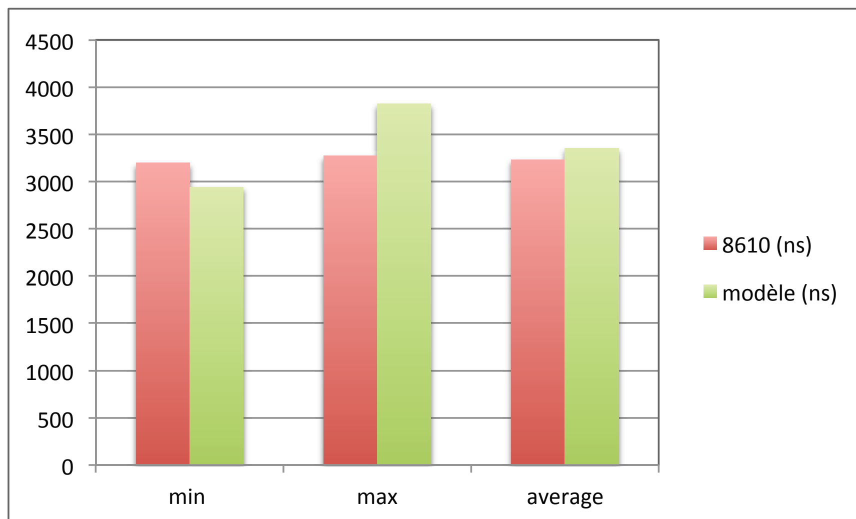

FIGURE 7.8 – Comparaison des temps d'exécution minimum, moyen et maximum pour l'application calcul de Pi

7.3.5 Résumé

Nous venons de voir que notre méthode de modélisation est fonctionnelle et permet d'obtenir des résultats cohérents par rapport à l'exécution réelle au niveau des temps d'exécution pour un système simple composé d'une application exécutée sur un calculateur. Nous allons maintenant appliquer cette méthode sur une plate-forme avionique plus complète, multi-partitions et avec les services de l'API ARINC653.

© 2012 TELECOM ParisTech

7.4 Mise en oeuvre du processus sur une plate-forme

7.4.1 Présentation de la plate-forme de test

Comme il a été présenté au début de ce chapitre, notre plate-forme est ici composée de deux calculateurs (deux fois le même que le calculateur précédent) connectés au travers d'un réseau avionique, comme présenté sur la figure 7.9. Nous avons donc le calculateur déjà présenté, avec la partie réseau ARINC664 et les services APEX en supplément du cas d'étude de la première étape de la partie validation expérimentale. La partie applicative est en revanche différente, nous l'avons également décrite au début de ce chapitre. Nous avons deux applications semblable de test réseau. Elles consistent en des appels aux fonctions de communication (réception et envoi de messages sur les files d'entrée et sortie) et des phases de calcul sur les valeurs reçues. Nous avons donc des appels aux services de l'APEX ainsi que du calcul. La première application est répliquée sur deux partitions, qui communiquent l'une avec l'autre. La seconde est répartie dans trois partitions.

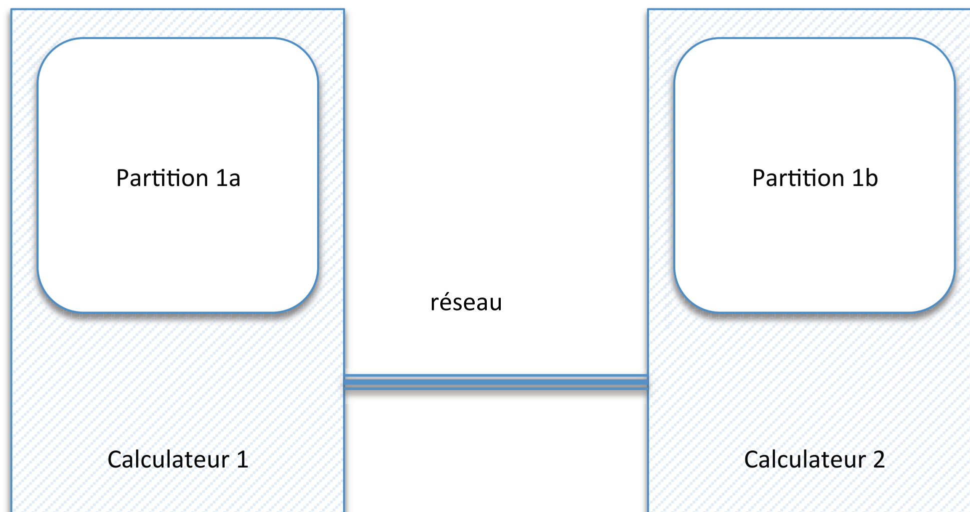

FIGURE 7.9 – Architecture simplifiée de la plate-forme de test

© 2012 TELECOM ParisTech

Ci-dessous, nous donnons une description algorithmique succincte de l'application utilisée :

```
receive_queuing_port()
if(no error){
    calculs
    send_queuing_port()
}
receive_queuing_port()
if(no_error){
    calculs
    send_queuing_port()
}
calculs
periodic_wait()
```

7.4.2 Modélisation du réseau ARINC664

Le réseau est un élément dimensionnant de la plate-forme, car les latences de traversée des messages d'un calculateur à un autre peuvent varier suivant plusieurs paramètres (congestion au niveau des routeurs, nombre de routeurs traversés, etc.). Nous avons vu dans le chapitre 3 que la modélisation du réseau avionique en vue de sa simulation pour extraire des latences de traversée est une opération complexe. Si l'on souhaite, à chaque envoi d'un message sur le réseau, calculer le temps de traversée sur ce dernier en fonction des autres messages, il est nécessaire de modéliser toutes les parties du réseau, à savoir les routeurs et les composants terminaux.

Par manque de temps, nous sommes passés par un outil qui a été développé en parallèle de cette thèse, au cours d'un stage que j'ai encadré. A l'image des travaux définis dans [Grieu, 2004], notre outil est basé sur la méthode "network calculus" qui permet de calculer les bornes de latence de transmission d'un message sur chaque chemin logique. Ainsi, nous pouvons dans notre cas configurer les deux chemins logiques reliant le calculateur 1 et le calculateur 2 dans chaque sens. Les latences exprimées sont statiques, mais elles permettent d'exprimer les temps de communications et ainsi de ne pas interrompre le processus de modélisation dans le cadre d'une modélisation globale d'une plate-forme.

© 2012 TELECOM ParisTech

7.4.3 Modélisation des services de l'API ARINC653

La modélisation des services de l'APEX que nous utilisons au sein de l'application ("receive_queuing_port", "send_queuing_port" et "periodic_wait" se fait en deux étapes. La première consiste à donner la liste des services utilisés au sein du modèle AADL. Dans la partie suivante, nous verrons que nous avons ajouté une propriété au processeur permettant de donner cette liste. Une seconde liste définit les temps de latence des services, temps qui sont donnés dans les différents documents techniques liés à la caractérisation des services du système d'exploitation.

La seconde étape consiste en la modélisation comportementale du service, afin d'extraire des appels aux ressources matérielles de la plate-forme d'exécution. Comme nous l'avons vu dans le chapitre 6, il est possible, à partir d'une description pseudo-algorithmique de chaque service, d'en extraire une représentation comportementale décrivant l'accès aux ressources matérielles. Par exemple, le service "Periodic_wait" peut se traduire en séquence d'appels matériels comme suit :

1. calculer le prochain point de relâchement de la tâche : précédent point de relâchement + période de la tâche
=> accès mémoire pour rapatrier le précédent point de relâchement et la période de la tâche, puis effectuer le calcul et stockage mémoire de la nouvelle valeur

2. modifier l'état courant de la tâche
=> modifier la valeur de la variable en mémoire

3. modifier la valeur de la deadline de la tâche : prochain point de relâchement + time_capacity
=> accès mémoire (rapatrier le point de relâchement et "time_capacity"), effectuer le calcul et stocker la valeur en mémoire

4. envoyer un code de retour
=> écrire la valeur en mémoire

Cette traduction en appels aux ressources matérielles n'est pas toujours tout à fait exacte et précise. Cependant, elle permet d'assurer la continuité dans la simulation et autorise l'étude de l'utilisation des ressources matérielles de la plate-forme.

© 2012 TELECOM ParisTech

7.4.4 Modélisation d'un calculateur au sein de la plate-forme

7.4.4.1 Modélisation architecturale

Notre objectif est ici d'étudier l'utilisation des ressources d'un calculateur au sein de la plate-forme. Pour cela, nous modélisons le calculateur déjà vu, en y ajoutant l'entrée-sortie ARINC664 utilisée lors des envois et réceptions. Le premier modèle reprend le modèle AADL précédent, auquel nous avons ajouté un composant "device" pour modéliser le réseau.

Afin de préciser le type de l'entrée-sortie au sein du composant AADL "device", nous avons utilisé l'extension de propriétés que nous avons présenté dans le chapitre 5 afin de préciser le type de l'entrée-sortie que l'on utilise. Nous avons également ajouté au processeur une propriété permettant de lister les services utilisés par la partition, et leurs temps de latences. Pour ce modèle AADL nous avons donc le même modèle que dans la première partie de ce chapitre, enrichi de deux composants matériels supplémentaires (l'entrée-sortie et le bus permettant de la connecter au processeur) et de la liste des services APEX utilisés. Le schéma 7.10 représente ce calculateur modélisé.

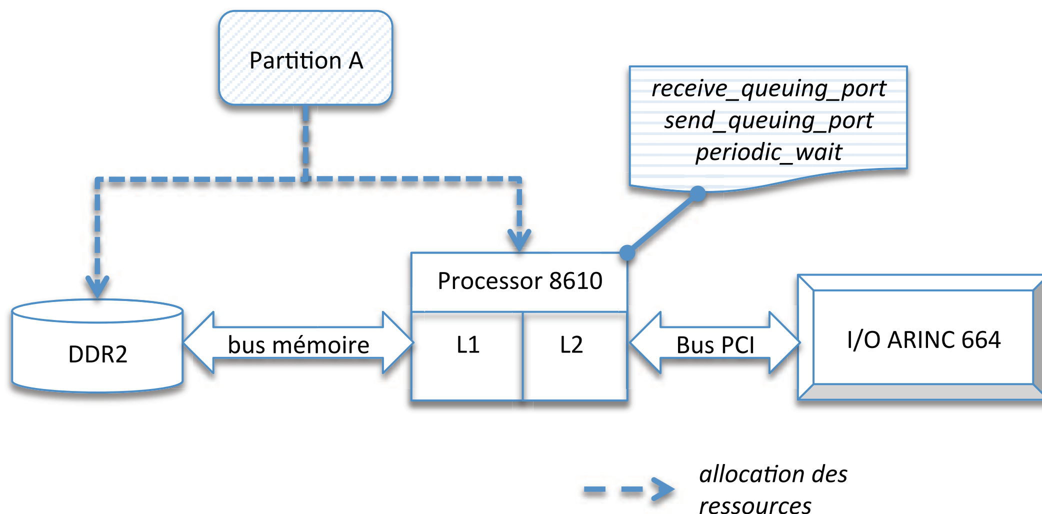

FIGURE 7.10 – Représentation du modèle architectural AADL du calculateur de test au sein de la plate-forme

© 2012 TELECOM ParisTech

7.4.4.2 Modélisation comportementale

A partir de ce modèle AADL du calculateur, nous appliquons la même méthode que précédemment pour générer la plate-forme d'exécution en SystemC. Pour chaque composant matériel, nous configurons l'automate comportemental correspondant et le rattachons à un container. Pour le moment, les stimuli de chaque composant matériel sont renseignés dans des fichiers de configuration qui sont parcourus par un module du container. Cette configuration permet la phase de création des tables de routage pour les stimuli. De même que pour les automates comportementaux SystemC des éléments matériels de la plate-forme d'exécution, nous avons développé des automates SystemC correspondant aux services APEX que nous utilisons. Ces automates ont été développés en suivant la méthode décrite dans la section "Modélisation des services de l'API ARINC653". Ils transforment les appels service en stimuli matériels qui sont relayés vers les composants matériels.

Concernant l'ordonnancement des partitions et tâches, celui-ci se fait via un composant SystemC indépendant du modèle, que nous avons appelé le "scheduler". Dans notre plate-forme, nous avons deux calculateurs exécutant chacun une partition monotâche. Nous avons donc un ordonnancement statique ARINC653 pour les 2 partitions. Chacune d'elle dure 20ms et se lance toutes les 40ms à tour de rôle. Dans cet exemple nous n'avons donc pas d'ordonnancement pour les tâches. Comme nous nous intéressons particulièrement à un calculateur au sein du modèle, nous avons donc l'unique partition du calculateur qui se lance toutes les 40ms. Toutefois, il serait possible de préciser l'ordonnancement dans le cas de partitions multi-tâches, sous réserve d'implémenter cette politique dans le composant "scheduler".
La plate-forme d'exécution SystemC peut par conséquent être schématisée comme sur la figure 7.11 : le module scheduler récupère les caractéristiques de la partition (durée, période), et lance son exécution suivant son "start_time" qui représente sa position dans le schéma d'ordonnancement fixe des partitions, la MAF. Ensuite, il récupère les stimuli du scénario de simulation correspondant et les envoie vers un module chargé de relayer ces stimuli soit vers les services de l'APEX, soit directement vers le processeur principal dans le cas de calculs.

© 2012 TELECOM ParisTech

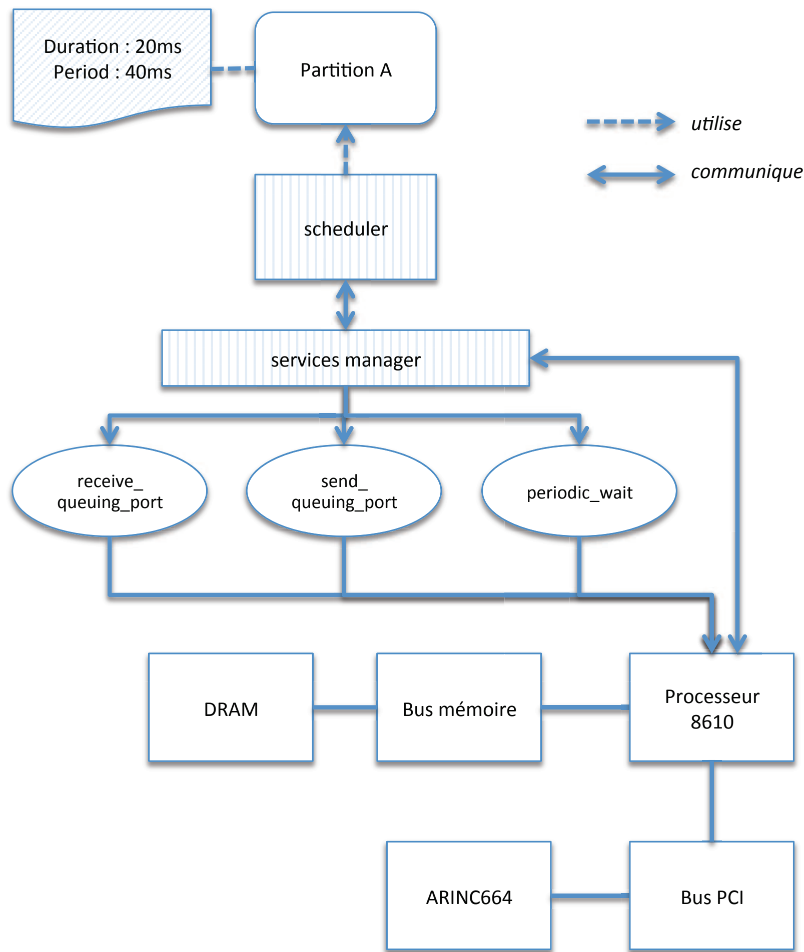

FIGURE 7.11 – Schéma du modèle comportemental du calculateur de test généré en SystemC

7.4.5 Simulation et analyses des résultats

7.4.5.1 Analyse des temps d'exécution

L'objectif est ici de valider notre processus de modélisation - simulation sur notre plate-forme composée des deux calculateurs, d'un système d'exploitation et des deux partitions. Pour cela, nous avons exécuté le scénario extrait de l'application initiale sur le modèle SystemC de la plate-forme d'exécution que nous avons généré. Nous avons exécuté l'application en trois étapes : pour la première étape, nous avons exécuté la première partie de l'application (premier appel service et le contenu de la première boucle). Ensuite nous avons ajouté le second appel service avec la seconde boucle, avant de lancer l'application complète. Le tableau 7.12 montre que les temps d'exécution mesurés par notre méthode sur le modèle de plate-forme sont très proches des

© 2012 TELECOM ParisTech

temps d'exécution mesurés sur la plate-forme réelle, l'écart étant inférieur à 5%.

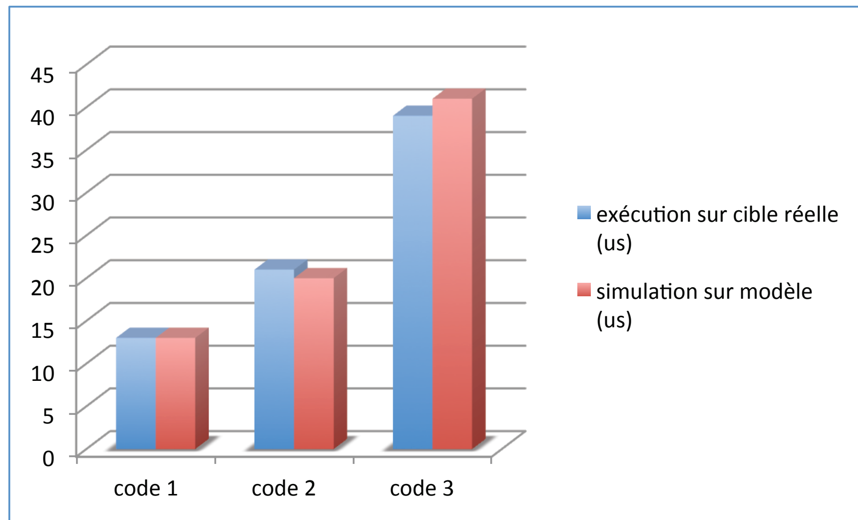

FIGURE 7.12 – Comparaison entre les temps d'exécution réels et simulés pour les différents codes exécutés

7.4.5.2 Temps de simulation

Les temps de simulation des 3 codes que l'on a exécutés sont données dans la figure 7.13. Ils sont compris entre 7,2 et 12,9 secondes, augmentant avec la taille du code et donc le nombre de stimuli à traiter, pour 10000 itérations de chaque code. Associé à la relative bonne précision des résultats que l'on a extrait précédemment, cela montre que notre méthode représente de simulation en avance de phase suit l'un des objectifs fixés, à savoir un bon compromis entre précision des résultats et temps de simulation.

	Temps de simulation pour 10000 itérations (secondes)
Code 1	7,2
Code 2	11,5
Code 3	12,9

FIGURE 7.13 – temps de simulation des différents codes exécutés

© 2012 TELECOM ParisTech

7.4.5.3 Extraction de l'utilisation des ressources matérielles

Afin de vérifier que notre processus offre la possibilité d'extraire l'utilisation des ressources matérielles de la plate-forme, nous avons utilisé la seconde application s'exécutant sur le même calculateur. L'application est, comme la précédente, une application de test réseau, et utilise les mêmes services. Le but est ici de montrer que notre méthode permet de tester la bonne configuration d'une architecture vis-à-vis de la partie applicative.

La figure 7.14 décrit l'ordonnancement des trois partitions, sous lequel nous visualisons les résultats extraits d'un cas d'exécution normal : les deux schémas montrent le taux d'utilisation de la mémoire principale d'une part, de l'entrée-sortie réseau d'autre part. L'entrée-sortie n'est utilisée que par les partitions 1 et 3, la partition 2 ne faisant que du calcul simple. Sur la première figure, les premières analyses montrent que l'architecture matérielle proposée, telle qu'elle est configurée, semble compatible avec l'application. Le processus de conception de la plate-forme peut donc se poursuivre, avec le développement de l'architecture matérielle en question.

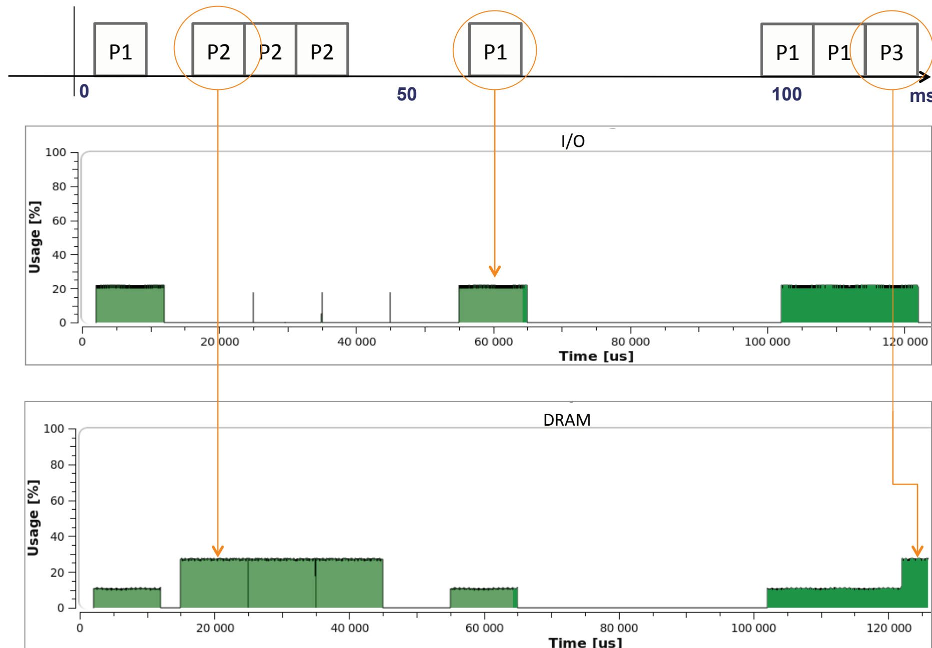

FIGURE 7.14 – Comparaison entre les temps d'exécution réels et simulés pour les différents codes exécutés

 © 2012 TELECOM ParisTech

Sur la figure 7.15, nous avons modifié la configuration de la plate-forme. L'entrée-sortie réseau a été configurée avec un temps de réponse très grand. Nous voyons sur les schémas que les résultats extraits de la simulation permettent de visualiser ce phénomène. Lors de l'exécution des partitions 1 et 3, qui utilisent cette ressource réseau, cette dernière monopolise la plate-forme. Nous voyons par exemple que l'utilisation de la ressource mémoire est anormalement basse. Notre méthode a donc permis de détecter que l'architecture matérielle proposée, telle qu'elle est configurée ici, n'est pas compatible avec l'application, et est ici à écarter.

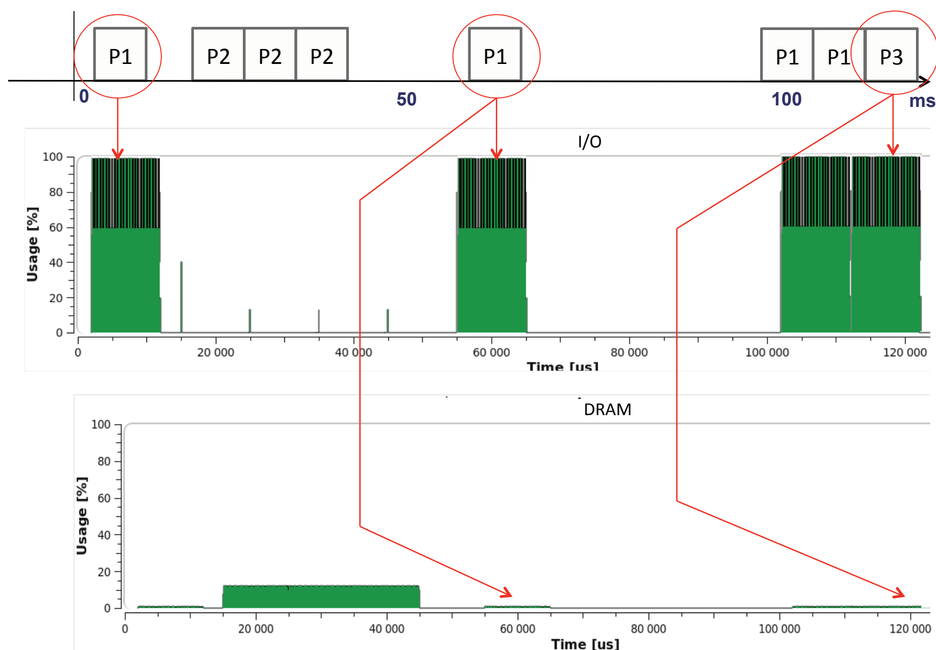

FIGURE 7.15 – Comparaison entre les temps d'exécution réels et simulés pour les différents codes exécutés

© 2012 TELECOM ParisTech

7.5 Synthèse

Adapter la modélisation en avance de phase à l'avionique : dans notre processus nous avons conservé les caractéristiques d'une modélisation en avance de phase, à savoir rapidité de modélisation et résultats de simulation suffisamment représentatifs par rapport à la réalité pour mener de premières analyses. Concernant la modélisation architecturale, les travaux déjà menés en AADL sur la modélisation de systèmes ARINC653 ont servi de base et permettent une modélisation rapide d'une plate-forme en prenant en compte les spécificités de l'avionique. De son côté, l'utilisation du standard TLM de SystemC permet la simulation du modèle de plate-forme d'exécution et offre le meilleur compromis entre temps de simulation et précision des résultats.

Les règles de transformation que nous avons développées pour générer un modèle SystemC à partir d'un modèle AADL, offrent la possibilité de s'appuyer sur une modélisation à deux niveaux d'abstraction. Ceci permet de mener des analyses spécifiques à chacun de ces niveaux, et permet également d'utiliser un large panel d'outils d'analyses existants.

Offrir une modélisation comportementale de la partie applicative : une contrainte du domaine avionique que nous avons identifiée réside dans l'aspect confidentiel des applications développées. Celles-ci ne sont pas toujours développées par le concepteur de la plate-forme d'exécution, et ne lui sont pas accessibles pour des raisons de confidentialité. Nous avons vu que notre méthode de description par blocs algorithmiques permet une modélisation de l'application, sans enfreindre cette contrainte de confidentialité. En effet, les blocs sont anonymes, aucun nom de fonction n'est présent, ce qui empêche de remonter au code source. Cette modélisation nous permet en outre de générer un scénario de simulation pour la plate-forme SystemC générée.

Tester la compatibilité entre la plate forme d'exécution et la partie applicative : la méthode permet d'explorer l'utilisation des ressources matérielles de la plate-forme par la partie applicative. Plus particulièrement, il est possible de détecter en avance de phase si l'architecture matérielle proposée permettrait d'exécuter les applications ou non. Cela permet de gagner du temps dans le processus de conception de la plate-forme, en évitant de détecter les mauvaises configurations plus en aval du cycle de développement. Ainsi, plusieurs configurations peuvent être testées ra-

© 2012 TELECOM ParisTech

pidement et comparées en fonction des résultats donnés par l'exécution de la partie applicative.

Automatiser le processus : les règles de transformation AADL vers SystemC, alliées au composant générique ”container”, offrent une génération quasi-transparente à l'utilisateur d'un modèle comportemental de la plate-forme d'exécution (système d'exploitation et partie matérielle). L'utilisateur fournit le modèle architectural AADL qui sert de point d'entrée au processus, et se concentre ensuite sur l'analyse des résultats.

Notre processus répond ainsi à l'objectif initial : fournir une méthode de modélisation d'une plate-forme avionique permettant d'explorer en avance de phase la compatibilité entre une architecture de plate-forme d'exécution (système d'exploitation et partie matérielle) et la partie applicative.

© 2012 TELECOM ParisTech

© 2012 TELECOM ParisTech

Cinquième partie

Conclusion Générale

Chapitre 8

Conclusions et Perspectives

Sommaire

8.1 Rappel des contributions 138

8.1.1 Génération d'un modèle comportemental de plate-forme d'exécution à partir d'un modèle architectural 138

8.1.2 Modélisation d'une application en avance de phase . . . 139

8.2 Conclusion 139

8.2.1 Méthodologie 139

8.2.2 Résultats expérimentaux 140

8.2.3 Automatisation du processus 140

8.3 Limites et perspectives 141

8.3.1 Limites 141

8.3.2 Perspectives 142

8.1 Rappel des contributions

Nous avons présenté dans ce mémoire nos contributions. Celles-ci ont pour but de répondre à notre principal objectif, à savoir fournir une méthode de modélisation d'une plate-forme avionique en avance de phase et d'exploration de ses performances, afin de déterminer la compatibilité entre une architecture matérielle et un ensemble d'applications, dans le respect d'exigences données.

8.1.1 Génération d'un modèle comportemental de plate-forme d'exécution à partir d'un modèle architectural

Notre première contribution concerne la génération d'un modèle comportemental de plate-forme d'exécution (système d'exploitation et partie matérielle) à partir du modèle architectural de la plate-forme complète. Le principal problème était ici l'absence de méthode de modélisation rapide d'une plate-forme avionique en avance de phase, qui permette de faire de la simulation pour en extraire les performances. Afin de remédier à cela, nous avons développé une méthode de modélisation s'appuyant sur deux niveaux d'abstraction. La première partie de la modélisation consiste en une description architecturale à l'aide du langage AADL de la plate-forme. Elle permet à l'utilisateur de décrire rapidement la plate-forme.
La seconde partie consiste en une génération d'un modèle comportemental de la plate-forme d'exécution, depuis le modèle architectural précédent. Grâce au composant générique que nous avons développé, le "container", cette génération est quasi transparente pour l'utilisateur. Pour chaque composant, le "container" découple la partie comportementale de la partie interface de communication. Il fournit une sur-couche aux fonctions de communications SystemC-TLM qui facilite la génération. Cette modélisation comportementale s'appuie sur le langage SystemC-TLM, et permet une simulation du modèle grâce au noyau de simulation intégré au langage.
Ces travaux ont été soumis à la communauté notamment au travers de publications [Lafaye et al., 2010a], [Lafaye et al., 2010b], [Lafaye et al., 2011a], [Lafaye et al., 2011c], [Lafaye et al., 2011b] et [Lafaye et al., 2012]. D'autre part, le composant générique "container" a donné lieu à un brevet étendu à l'international.

© 2012 TELECOM ParisTech

8.1.2 Modélisation d'une application en avance de phase

Notre seconde contribution consiste en une modélisation d'applications avioniques. Celles-ci sont souvent développées par un tiers, et non par le concepteur de plate-forme. Pour des raisons de confidentialité, ce dernier n'a généralement pas accès au code source des applications, interdisant les méthodes à base de parcours de code. Nous avons donc développé une méthode permettant de description d'une application avionique sous la forme d'un enchaînement de blocs algorithmiques (data_load, calculus, I/O_read, etc.). Nous avons une modélisation sous forme d'enchainement de macro-opérations, qui ne comporte aucun nom de fonction. Il n'est ainsi pas possible de remonter au code source, et nous respectons donc bien la contrainte de confidentialité. Cette contribution a été reprise au sein de Thales pour un projet en collaboration avec d'autres entreprises du secteur avionique dont Eurocopter.
La seconde partie de cette contribution consiste en la génération d'un scénario de simulation partant de la description algorithmique précédente. Le problème était ici d'extraire depuis la partie applicative un scénario de simulation pour le modèle comportemental de la plate-forme d'exécution, malgré l'impossibilité d'accéder au code source des applications. Notre solution est de partir de la description des applications par blocs algorithmiques, et d'y appliquer des patrons de génération de stimuli. Pour chaque bloc algorithmique correspond un patron qui transforme ce bloc en une suite de stimuli compréhensibles par le modèle comportemental.

8.2 Conclusion

8.2.1 Méthodologie

Découlant de ces deux contributions, nous avons définis une méthode de modélisation et de simulation d'une plate-forme avionique en avance de phase. Elle s'appuie sur deux langages de modélisation matures, chacun à un niveau d'abstraction : architectural pour AADL, comportemental pour SystemC. Grâce à cette modélisation multi-niveaux, il est possible de mener différentes analyses s'appuyant sur l'un ou l'autre de ces niveaux d'abstraction. Cette méthode répond aux différents problèmes identifiés dans le chapitre 2 : i) modélisation d'une application en avance de phase respectant le côté confidentiel de celle-ci, ii) une méthode de génération de scénario de simulation à partir de cette modélisation, iii) une modélisation de la plate-forme prenant en compte les contraintes avioniques, et iv) une modélisation comportemen-

© 2012 TELECOM ParisTech

tale de la plate-forme d'exécution pour la simulation et l'extraction de l'utilisation de ses ressources matérielles. L'analyse de l'utilisation des ressources de la plate-forme permet par la suite de déterminer la compatibilité entre l'architecture matérielle et les applications.

8.2.2 Résultats expérimentaux

Nous avons montré que les résultats obtenus par notre simulation d'exécution d'application sur le modèle comportemental de plate-forme d'exécution, sont proches et représentatifs de l'exécution des applications sur la plate-forme réelle. Même si la méthode doit être améliorer pour gagner en précision, cela montre que l'approche présentée ici et les concepts sur lesquels elle s'appuit sont valables et méritent d'être approfondis ou enrichis. D'autre part, la partie des travaux concernant le " container" a fait l'objet d'un brevet étendu aux Etats-Unis, démontrant l'intérêt des travaux proposés. Un autre brevet est également en cours pour la partie de modélisation d'une application par blocs algorithmiques.

La méthode répond aux exigences de l'exploration de performances d'une plate-forme en avance de phase, à savoir la possibilité de modéliser la plate-forme et de la simuler pour tester plusieurs variantes d'architectures. La méthode permet de déterminer si une architecture peut exécuter un ensemble d'applications dans le respect d'exigences données. Grâce à cela, la méthode permet de mieux contraindre la suite du processus de développement de la plate-forme d'exécution, en pré-validant ou rejetant certaines choix architecturaux. Enfin, l'automatisation partielle de la méthode que nous avons faite montre que celle-ci est simple et rapide d'utilisation, ce qui est également un critère de la modélisation en avance de phase.

8.2.3 Automatisation du processus

Les parties de description architecturale AADL de la plate-forme et de la description algorithmique des applications sont des opérations manuelles, qui peuvent toutefois être effectuées au travers d'outils (OSATE pour le modèle architectural AADL, outil en cours de développement pour la modélisation de l'application). Pour le moment, seules certaines parties du processus ont été partiellement automatisées : la

© 2012 TELECOM ParisTech

génération des stimuli à partir de la description algorithmique d'une application a été automatisée, mais cette automatisation doit être améliorée. La génération du modèle SystemC depuis le modèle AADL est automatisée, même si cette automatisation devra être adaptée aux futures évolutions. Enfin, la visualisation des résultats de simulation est automatisable mais seule une petite partie l'a été pour différentes démos effectuées lors de présentations.

8.3 Limites et perspectives

Les limites de nos travaux résultent d'une part du manque de temps dédié à l'implémentation de nos travaux, d'autre part du cadre restreint d'application de nos travaux. Liée à cela, une partie des perspectives consistera donc à compléter nos travaux, et à mener des analyses sur des systèmes plus larges que ceux étudiés.

8.3.1 Limites

Concernant les principes de blocs algorithmiques et de patrons de génération de stimuli, le développement s'est avéré limité aux cas d'utilisation lors des nos expérimentations. Toutefois, l'objectif était ici de fournir un concept et montrer sa validité, et non de développer nous-même tous les blocs algorithmiques ou patrons de génération de stimuli des différentes plates-formes cibles de Thales Avionics. Il en va de même pour la base de composants comportementaux SystemC, qui devrait être étoffée au sein de Thales afin de permettre la modélisation de plusieurs plates-formes en cours de développement / utilisation. Le développement de ces éléments devrait en outre permettre de gagner en précision au niveau de l'exploration de performances.

Une autre limitation réside dans la modélisation du réseau de communication entre les calculateurs. Là encore par manque de temps, une modélisation "pire cas / meilleur cas" a été effectuée afin de déterminer les latences de transfert de trames d'un calculateur à un autre. Une modélisation comportementale du réseau, prenant en compte les différents éléments de celui-ci (files d'envoi et réception, routeurs), permettrait de gagner en précision sur les performances obtenues lors de la modélisation d'une plate-forme. Des travaux doivent néanmoins démarrer dans le prolongement de ma thèse à Thales Avionics pour mener cette modélisation.

© 2012 TELECOM ParisTech

8.3.2 Perspectives

Notre processus constitue une base pour répondre au besoin de modélisation et d'exploration des performances d'une plate-forme en avance de phase. Par manque de temps, certaines parties restent à compléter. Les perspectives tiennent donc principalement en le prolongement des travaux, par l'enrichissement des parties incomplètes et l'amélioration des concepts développés.

Perspectives académiques

Les travaux de génération d'un modèle comportemental SystemC depuis un modèle architectural AADL ont été notamment soumis à la communauté AADL lors d'un meeting. Notre objectif est de contribuer à la transformation de modèle AADL, dans notre cas vers SystemC. De plus, la méthode, bien qu'initialement développée et testée pour les systèmes avioniques, n'est cependant pas exclusivement réservée à ce domaine. Elle pourrait être adaptée pour modéliser et simuler en avance de phase d'autres types de plates-formes.
Concernant cette même transformation, une prise en compte de l'annexe comportementale AADL dans le processus est à envisager. Développée pour décrire le comportement des composants d'un système AADL, l'annexe était encore en phase de développement au début de ma thèse. Elle est désormais mature et offre ainsi de nouvelles perspectives pour enrichir la phase de génération des modèles comportementaux.

Perspectives industrielles

La méthode présentée dans notre mémoire fait l'objet d'une industrialisation de la part de notre partenaire Thales Avionics. Un outil de modélisation - simulation va donc voir le jour, après une phase d'amélioration du processus. Cet outil sera intégré dans la chaine d'outils de modélisation utilisée par les équipes de développement de plates-formes de Thales Avionics.
Des travaux complémentaires sur la modélisation du réseau avionique viendront améliorer le processus global. Notamment, une modélisation AADL de l'architecture réseau a été proposée dans la thèse suivante [Mendez, 2009], comme ressources logicielles déployées sur des ressources matérielles (les calculateurs) connectés entre eux via le réseau ARINC664 principalement. D'autre part, il a été montré dans ces

© 2012 TELECOM ParisTech

travaux [Lauer et al., 2010] qu'une modélisation du réseau par automates comportementaux est une solution viable pour la partie description comportementale de la plate-forme. Ces travaux constituent une bonne base pour l'extension éventuelle de mes travaux de modélisation par ajout d'une modélisation plus poussée du réseau.
La présentation et démonstration de nos travaux dans plusieurs conférences a permis de nouer des contacts avec plusieurs entreprises du domaine avionique et domaines connexes, intéressées par notre méthode. Des collaborations sont à l'étude afin de compléter notre méthode et la tester sur des plates-formes encore plus complexes. Enfin, la collaboration avec Eurocopter sur la partie de modélisation des applications par blocs algorithmiques devrait permettre d'améliorer celle-ci.

© 2012 TELECOM ParisTech

© 2012 TELECOM ParisTech

Bibliographie

[Accelera, 2001] Accelera (2001). *System-Verilog Language Reference Manual*. Accelera.

[Adballah et al., 2011] Adballah, A., Gamatie, A., and Dekeyser, J. (2011). Modélisation uml/marte de soc et analyse temporelle basée sur l'approche synchrone. In *RSTI'11 proceedings*.

[Aeronautical-Radio-Inc, 1997] Aeronautical-Radio-Inc (1997). *ARINC report 651-1 : Design guidance for integrated modular avionics*. Aeronautical Radio Inc.

[Aeronautical-Radio-Inc, 2002a] Aeronautical-Radio-Inc (2002a). *ARINC 664 Aircraft data network part 1 : systems and concepts*. Aeronautical Radio Inc.

[Aeronautical-Radio-Inc, 2002b] Aeronautical-Radio-Inc (2002b). *ARINC 664 Aircraft data network part 2 : ethernet physical and data link layer specification*. Aeronautical Radio Inc.

[Aeronautical-Radio-Inc, 2002c] Aeronautical-Radio-Inc (2002c). *ARINC 664 Aircraft data network part 7 : deterministic networks*. Aeronautical Radio Inc.

[Aeronautical-Radio-Inc, 2006] Aeronautical-Radio-Inc (2006). *Avionics application software standard interface part 1 : required services*. Aeronautical Radio Inc.

[Aerospace, 2008] Aerospace, S. (2008). *Architecture Analysis and Design Language (AADL)*. SAE.

[Albert, 2009] Albert, V. (2009). *Evaluation de la Validité de la Simulation dans le Cadre du Développement des Systèmes Embarqués*. PhD thesis, Université de Toulouse III.

[Alena et al., 2006] Alena, R., Ossenfort, J., Laws, K. I., Goforth, A., and Figueroa, F. (2006). *Communications for Integrated Modular Avionics*. Nasa.

[Alena et al., 2007] Alena, R., Ossenfort, J.-P., Laws, K. I., Goforth, A., and Figueroa, F. (2007). Communications for integrated modular avionics. In *IEEE Aerospace Conference proceedings*, pages 1–18.

[André, 2007] André, C. (2007). *Le temps dans le profil UML MARTE*. Université Nice Sophia Antipolis - I3S.

[Berry and Gonthier, 1992] Berry, G. and Gonthier, G. (1992). *The Esterel Synchronous Programming Language : Design, Semantics, Implementation*. Science of computer programming.

[Borde, 2009] Borde, E. (2009). *Configuration et Reconfiguration des Systèmes Temps-Réel Répartis Embarqués Critiques*. PhD thesis, Telecom ParisTech.

[Bouhadiba et al., 2009] Bouhadiba, T., Maraninchi, F., and Funchal, G. (2009). *Formal and Executable Contracts for Transaction-Level Modeling in SystemC*. Verimag Research Report.

[Brunette et al., 2005] Brunette, C., Delamare, R., Gamatie, A., Gautier, T., and Talpin, J.-P. (2005). *A modeling paradigm for integrated modular avionics design*. INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES.

[Buchmann, 2006] Buchmann, R. (2006). *Modélisation et simulation rapide au niveau cycle pour l'exploration architecturale de systèmes intégrés sur puce*. PhD thesis, Université ParisVI.

[Cai and Gajski, 2008] Cai, L. and Gajski, D. (2008). *Transaction Level Models : an Overview*. Center for embedded computer systems.

[Casteres and Ramaherirariny", 2009] Casteres, J. and Ramaherirariny", T. (2009). Aircraft integration real-time simulator modeling with aadl for architecture tradeoffs. In *DATE'09*, pages 346–351.

[Charara, 2007] Charara, H. (2007). *Evaluation des performances temps réel de réseaux embarqués avioniques*. PhD thesis, Institut National Polytechnique de Toulouse.

[de Niz and Rajkumar, 2006] de Niz, D. and Rajkumar, R. (2006). Partitioning bin-packing algorithms for distributed real-time systems". *IJES*, pages 196–208.

[Delange et al.,] Delange, J., Pautet, L., , Plantec, A., Kerboeuf, M., Singhoff, F., and Kordon, F. Validate, simulate and implement arinc653 systems using the aadl. In *SIGAda'09 proceedings of the ACM SIGAda annual international conference on Ada and related technologies*, pages 31–44.

[Dissaux et al., 2006] Dissaux, P., Bodeveix, J., Filali, M., Gaufillet, P., and Vernada, F. (2006). Aadl behavioral annex. In *Proceedings of DASIA conference*.

[Dissaux and Singhoff, 2008] Dissaux, P. and Singhoff, F. (2008). Stood and cheddar : Aadl as a pivot language for analyzing performances of real time architectures. In *proceedings of 4th international congress on embedded real time systems (ERTS)*.

[Easwaran et al., 2009] Easwaran, A., Lee, I., Sokolsky, O., and Vestal, S. (2009). *A compositional framework for avionics ARINC653 systems*. Department of computer and information science, university of pennsylvania.

[Ermont et al., 2006] Ermont, J., Scharbarg, J., and Fraboul, C. (2006). Worst-case analysis of a mixed can/switched ethernet architecture. In *International conference on real-time and network systems*, pages 45–54.

[Feiler et al., 2004] Feiler, P., Lewis, B., Hudak, J., and Gluch, D. P. (2004). *Pattern-Based Analysis of an Embedded Real-time System Architecture*. Carnegie-Mellon University / SEI.

[Feiler et al., 2000] Feiler, P., Lewis, B., and Vestal, S. (2000). *Improving Predictability in Embedded Real Time Systems*. Carnegie-Mellon University / SEI.

[Gamatie and Gautier, 2002] Gamatie, A. and Gautier, T. (2002). *Synchronous Modeling of Modular Avionics Architectures using the SIGNAL Language*. INRIA Rennes.

[Gamatie et al., 2005] Gamatie, A., Gautier, T., Guernic, P. L., and Talpin, J. (2005). *Polychronous Design of Embedded Real Time Systems*. INRIA Rennes.

[Gamatié, 2004] Gamatié, A. (2004). *Modélisation polychrone et évaluation de systèmes temps réel*. PhD thesis, Université de Rennes 1.

[Ghenassia, 2005] Ghenassia, F. (2005). *Transaction Level Modeling with SystemC, TLM concepts and applications for embedded systems*. Springer-Verlag.

 © 2012 TELECOM ParisTech

[Grieu, 2004] Grieu, J. (2004). *Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques*. PhD thesis, Institu National Polytechnique de Toulouse.

[Halbwachs et al., 1996] Halbwachs, N., Caspi, P., Raymond, P., and Pilaud, D. (1996). The synchronous dataflow programming language lustre. In *IEE Colloquium on the Hardware-Software Cosynthesis for Reconfigurable*.

[Herbert et al., 2008] Herbert, P., Fellmuth, J., and Glesner, S. (2008). Model checking systemc designs using timed automata. In *CODES+ISSS '08 Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, pages 131–136.

[Hughes and Singhoff,] Hughes, J. and Singhoff, F. Développement de systèmes à l'aide d'aadl - ocarina / cheddar. In *Ecole d'été temps-réel 09 proceedings*, pages 25–34.

[IEEE-Computer-Society, 2000] IEEE-Computer-Society (2000). *IEEE Standard VHDL Language Reference Manual*. IEEE Computer Society.

[IEEE-Computer-Society, 2001] IEEE-Computer-Society (2001). *IEEE Standard Verilog Language Hardware Description Language*. IEEE Computer Society.

[IEEE-Computer-Society, 2006] IEEE-Computer-Society (2006). *IEEE Standard SystemC Language Reference Manual*. IEEEComputerSociety.

[Januzaj et al., 2009] Januzaj, V., Mauersberger, R., and Biechele, F. (2009). *A Modelling and Analysis Framework for Avionics Systems*. EADS.

[Kountouris and Guernic, 1991] Kountouris, A. and Guernic, P. L. (1991). Profiling of signal programs and its application in the timing evaluation of design implementations. In *IEEE 1991 proceedings*.

[la Camara et al., 2007] la Camara, P. D., del Mar Gallardo, M., and Merino, P. (2007). Model extraction for arinc 653 based avionics software. In *Proceedings of the 14th international SPIN conference on Model checking software*, pages 243–262.

[Lafaye et al., 2011a] Lafaye, M., Borde, E., Gatti, M., and Pautet, L. (2011a). Towards aadl to systemc mapping for partitioned systems. In *AADL meeting, Paris*.

[Lafaye et al., 2010a] Lafaye, M., Faura, D., Gatti, M., and Pautet, L. (2010a). A new modeling approach for ima platform early validation. In *Proceedings of the 7th ACM International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, pages 17–20.

[Lafaye et al., 2010b] Lafaye, M., Faura, D., Gatti, M., and Pautet, L. (2010b). A new modeling approach for ima platform early validation. In *Proceedings of the Junior Researcher Workshop on Real-Time Computing 2010*, pages 11–14.

[Lafaye et al., 2011b] Lafaye, M., Faura, D., Gatti, M., and Pautet, L. (2011b). Model driven early exploration of ima execution platform. In *2011 IEEE/AIAA 30th Digital Avionics Systems Conference proceedings*, pages 7A2–1 – 7A2–11.

[Lafaye et al., 2011c] Lafaye, M., Faura, D., Gatti, M., and Pautet, L. (2011c). New modeling approach for ima platform early exploration. In *Avionics Europe expo 2011 proceedings*.

[Lafaye et al., 2012] Lafaye, M., Faura, D., Gatti, M., Pautet, L., and Borde, E. (2012). Model driven resource usage simulation for critical embedded systems. In *Design, Automation and Test in Europe Conference and Exhibition 2012 proceedings*, pages 312 – 315.

© 2012 TELECOM ParisTech

[Lasnier et al., 2010] Lasnier, G., Robert, T., Pautet, L., and Kordon, F. (2010). Architectural and behavioral modeling with aadl for fault tolerant embedded systems. In *ISORC - IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*.

[Lauer et al., 2010] Lauer, M., Ermont, J., Pagetti, C., and Boniol, F. (2010). Analyzing end-to-end functional delays on an ima platform. In *ISoLA'10 Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation*, pages 243–257.

[Laurent, 2002] Laurent, J. (2002). *Estimation de la consommation dans la conception système des applications embarquées temps-réels*. PhD thesis, Université de Bretagne Sud.

[Li and Xiong, 2009] Li, X. and Xiong, H. (2009). Modelling and simulation of integrated modular avionics systems. In *28th Digital Avionics Systems Conference (DASC)*.

[Liu and Layland,] Liu, C. and Layland, J. *Scheduling algorithms for multiprogramming in a hard real-time environment*.

[Ma et al., 2007] Ma, Y., Talpin, J., and Gautier, T. (2007). *Virtual prototyping AADL architectures in a polychronous model of computation*. INRIA Rennes.

[Mallet et al., 2009] Mallet, F., Andre, C., and Deantoni, J. (2009). Executing aadl models with uml/-marte. In *ICECCS'09 proceedings*.

[Mallet and Simone, 2009] Mallet, F. and Simone, R. D. (2009). *MARTE vs AADL for discrete-event and discrete time domains*. Inria.

[Medina et al., 2001] Medina, J., Gonzalez, M., and Drake, J. (2001). Mast real-time view : a graphic uml tool for modeling object-oriented real-time systems. In *IEEE Real Time Systems Symposium 01*.

[Mendez, 2009] Mendez, Y. M. (2009). *Modeling and evaluation of modular spacecraft avionics network architectures*. PhD thesis, Université de Lulea et Astrium Toulouse.

[OMG, 2007] OMG (2007). *Unified Modeling Language technical report*. Object Management Group.

[OMG, 2008] OMG (2008). *System Modeling Language technical report*. Object Management Group.

[OMG, 2009] OMG (2009). *Modeling and Analysis of Real-time Embedded Systems specification version 1.0*. Object Management Group.

[OSCI, 2007] OSCI (2007). *TLM2 User Manual*. Open SystemC Initiative (OSCI).

[Ott, 2007] Ott, A. (2007). *System testing in the avionics domain*. PhD thesis, Université de Brême.

[Rolland, 2008] Rolland, J. (2008). *Développement et validation d'architectures dynamiques*. PhD thesis, Université Toulouse III.

[RTCA, 1992] RTCA (1992). *Software consideration in airborne systems and equipment certification*. RTCA.

[Rugina, 2007] Rugina, A.-E. (2007). *Modélisation et évaluation de la sureté de fonctionnement - de AADL vers les réseaux de Petri stochastiques*. PhD thesis, Institut National Polytechnique de Toulouse - LAAS.

[Sagaspe, 2008] Sagaspe, L. (2008). *Allocation sûre dans les systèmes aéronautiques : Modélisation, Vérification et Génération*. PhD thesis, Université de Bordeaux 1.

[Senn et al., 2008] Senn, E., Laurent, J., Juin, E., and Diguet, J. (2008). Refining power consumption estimations in the component based aadl design flow. In *Forum on Specification, Verification and Design Languages, 2008*, pages 173–178.

© 2012 TELECOM ParisTech

[Singhoff et al., 2005] Singhoff, F., Legrand, F., and Nana, J. (2005). Aadl resource requirements analysis with cheddar. In *SAE AADL Working Group meeting, Paris, October 18-21, 2005*.

[Singhoff et al., 2004] Singhoff, F., Legrand, J., Nana, L., and Marcé, L. (2004). Cheddar : a flexible real time scheduling framework. In *Proceedings of the 2004 annual ACM SIGAda international conference on Ada : The engineering of correct and reliable software for real-time and distributed systems using Ada and related technologies*, pages 1–8.

[Varona-Gomez and Villar, 2009] Varona-Gomez, R. and Villar, E. (2009). Aads : Aadl simulation and performance analysis in systemc. In *ICECCS '09 Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems*, pages 323–328.

[Varona-Gomez et al., 2005] Varona-Gomez, R., Villar, E., Quijano, D., and Martinez, M. (2005). *Scope : SoC Co-simulation and Performance Estimation in SystemC*. Universita of Cantabria, Teisca dept.

[Zalila et al., 2009] Zalila, B., Lasnier, G., Pautet, L., and Hughes, J. (2009). Ocarina : un environnement pour l'analyse de modèle aadl et la génération automatique d'applications temps réel réparties embarquées. In *Ecole d'été temps réels'09 Telecom ParisTech proceedings*.

© 2012 TELECOM ParisTech