Conception des systèmes hétérogènes multilanguages

P. Coste

UNIVERSITE JOSEPH FOURIER-GRENOBLE 1
SCIENCES & GEOGRAPHIE

|__/__/__/__/__/__/__/__/__/__|

THESE

pour obtenir le grade de
DOCTEUR DE l'UNIVERSITE JOSEPH FOURIER

Discipline : INFORMATIQUE

Présentée et soutenue publiquement
par

Pascal COSTE

le 12 janvier 2001

Titre :

CONCEPTION DES SYSTEMES HETEROGENES MULTILANGAGES

Directeur de thèse :
M. Ahmed-Amine JERRAYA

Composition du Jury :

M.	Bernard Courtois	, Président
M.	Eric Martin	, Rapporteur
M.	Jean-Paul Calvez	, Rapporteur
M.	Ahmed-Amine Jerraya	, Directeur de thèse
M.	Bernard Conq	, Examinateur

A Cécile,
A mes parents,

Remerciements

C’est avec beaucoup d’émotions que j’ai l’occasion de témoigner par écrit de ma profonde reconnaissance envers les personnes qui m’ont apporté aide, soutien, confiance et amitié durant ce travail au sein du laboratoire TIMA.

Je remercie ma tendre épouse Cécile pour son soutien et ses encouragements dans les instants les plus difficiles. Merci pour son aide et pour avoir supporté mes humeurs durant cette période dense en activité, infiniment merci.

Je remercie mon directeur de thèse Monsieur Ahmed-Amine Jerraya, chargé de recherche au CNRS et responsable du groupe de Synthèse au niveau système (SLS) du laboratoire TIMA de m’avoir accueilli dans son groupe malgré mes contraintes. Merci pour tout ce temps passé ensemble et pour cette formidable disponibilité. Merci de m’avoir transmis en partie la rigueur qui le caractérise. Je lui suis tout particulièrement reconnaissant de m’avoir permis de découvrir beaucoup d’aspects de la micro-électronique que j’ignorais.

Je remercie Monsieur Bernard Courtois, Directeur de Recherches au CNRS et Directeur du laboratoire TIMA, de m’avoir accueilli dans son laboratoire et de m’avoir fait l’honneur de présider le jury de ma thèse.

Je remercie Monsieur Bernard Conq, responsable de projet CNET(FT), pour m’avoir offert la chance de collaborer à un projet industriel innovant et à la pointe de la technologie.

Je tiens à remercier Fabiano, Philippe et Gabriela pour leur collaboration dans ce travail et leur amitié. Merci à Paul pour m’avoir fait découvrir l’enseignement en université, une formidable expérience.

Merci à tous mes amis ayant séjourné un peu ou beaucoup au laboratoire TIMA, dans le désordre, Fabiano, Amer, Benoît, Fabien, Damien, Vincent, Philippe, Paul, Alexandre, Hubert, Jean-Marc, Sonja, Zoltan, Thierry, Lovic, Polen, Sabi, Wander, Rodolphe, Salvador et j’en oublie, pour ces moments partagés à refaire le monde autour d’un repas ou d’un café…

Enfin, merci à toute l’équipe SLS et à tous les membres du laboratoire TIMA pour leur grande sympathie.

Résumé

La conception d'un système électronique devient de plus en plus difficile pour des raisons de complexité et d'hétérogénéité sans cesse croissantes, ajouté à cela, les méthodes de travail et les compétences des concepteurs évoluent moins vite que les possibilités techniques d'intégration. Ces constatations amènent à la conclusion que les méthodes de conception actuelles avouent leurs limites et ont besoin d'évoluer.

Le sujet de cette thèse porte sur une méthode de conception permettant d'appréhender de façon globale un système électronique complexe. Cette méthode repose sur une approche modulaire d'un système où sont séparés le comportement d'un module et les communications entre les modules. Le raffinement et la simulation de chaque module sont confiés aux outils habituels et adaptés au domaine d'application. La coordination globale est décrite au travers d'un langage de coordination et la simulation globale fait appel à la technique de cosimulation géographiquement répartie. L'environnement de raffinement des communications permet de préciser le comportement des communications afin de suivre le raffinement du comportement des modules jusqu'à la synthèse.

La méthode présentée doit permettre de réduire significativement le temps de mise sur le marché d'un produit par son approche globale permettant de simuler très tôt un système. Cette méthode, et plus particulièrement l'environnement de cosimulation, a été utilisée avec succès lors d'une expérience menée en collaboration avec le CNET(France Télécom), ST–microelectronics et AREXSYS.

Mots Clés : Conception multilangage, cosimulation, synthèse de la communication, langage de coordination

Abstract

The design of electronic systems becomes more and more difficult mainly because of 2 points:

- Increase in complexity and heterogeneity,
- Engineers have more and more difficulties to meet the requirements of new integration possibilities.

This explains the need of new design methodologies.

The main goal of this work is to develop a new design methodology based on a global approach and modularity, whereby the behaviors and the communications are divided into different specifications. Each module is refined and simulated by specific and adapted tools. A coordination language specifies the global coordination and the simulation uses the cosimulation approach. The communication synthesis environment is used to refine communications to match the needed abstraction.

This methodology can reduce drastically the “time to market” of a new product by the global approach and the early simulations. This methodology has been validated by a successful experiment on a VDSL modem carried out in collaboration with CNET (France-Telecom), ST–microelectronics, AREXSYS and TIMA on a VDSL modem experience.

Keywords : Multilanguage Design, cosimulation, communication synthesis, coordination language.

Sommaire

INTRODUCTION 17

1.1 INTRODUCTION 18
1.2 CONCEPTION MULTILANGAGE 18
1.3 FLOT DE CONCEPTION 19
1.4 CONTRIBUTIONS 20
1.4.1 CONCEPTION DE SYSTÈMES ÉLECTRONIQUES 20
1.4.2 RAFFINEMENT DES COMMUNICATIONS 20
1.4.3 VALIDATION DES ÉTAPES DE CONCEPTION PAR SIMULATION 21
1.5 PLAN DU DOCUMENT 21

SPÉCIFICATION ET CONCEPTION DE SYSTÈMES ÉLECTRONIQUES 23

2.1 INTRODUCTION 24
2.2 LANGAGES DE DESCRIPTION DE SYSTÈMES ÉLECTRONIQUES 24
2.2.1 CONCEPTS EMPLOYÉS POUR L'ÉTUDE 24
2.2.1.1 Principaux concepts employés dans les langages de spécification système 25
2.2.1.2 Modèles d'exécution des langages de spécification système 26
2.2.2 SDL 26
2.2.2.1 Structure 27
2.2.2.2 Communications 27
2.2.2.3 Comportement 28
2.2.2.4 Environnement de conception SDL 32
2.2.3 MATLAB 33
2.2.3.1 Structure 33
2.2.3.2 Communications 35
2.2.3.3 Comportement 36
2.2.3.4 Environnement 36
2.2.4 COSSAP 37
2.2.4.1 Structure 37
2.2.4.2 Communications 39
2.2.4.3 Comportement 40
2.2.4.4 Environnement 41
2.3 CONCEPTION DE SYSTÈMES ÉLECTRONIQUES 42
2.3.1 CONCEPTS DE BASE ET NIVEAUX D'ABSTRACTION POUR LA SPÉCIFICATION DES SYSTÈMES ÉLECTRONIQUES 43

2.3.1.1 Concepts de base pour la spécification des systèmes électroniques 43
2.3.1.2 Niveaux d'abstraction pour la spécification des systèmes électroniques 45
2.3.1.3 Particularisation des concepts au travers des niveaux d'abstraction 46
2.3.2 FLOT DE CONCEPTION POUR LES SYSTÈMES ÉLECTRONIQUES 47
2.3.3 LES LANGAGES DE SPÉCIFICATION DES SYSTÈMES ÉLECTRONIQUES 49
2.3.3.1 Méthodes actuelles de spécification système 49
2.3.3.2 Capacités des langages pour la modélisation des concepts de base 50
2.3.4 SOLUTIONS POUR LA SPÉCIFICATION DES SYSTÈMES ÉLECTRONIQUES 51
2.3.4.1 Solutions pour la spécification homogène 51
2.3.4.2 Solution pour la spécification hétérogène des systèmes électroniques 52
2.3.4.3 Analyse des solutions proposées pour la spécification des systèmes électroniques 52
2.4 CONCLUSION 53

SYNTHÈSE DE LA COMMUNICATION DANS LE CADRE DES SYSTÈMES HÉTÉROGÈNES MULTILANGAGES 55

3.1 INTRODUCTION 56
3.1.1 SPÉCIFICATION DES SYSTÈMES HÉTÉROGÈNES : DÉFINITION DU PROBLÈME 56
3.1.1.1 Modélisation de la communication dans le cadre des systèmes hétérogènes 57
3.1.1.2 Synthèse de la communication dans un système hétérogène 58
3.1.2 DOMAINE D'APPLICATION 59
3.1.2.1 Réutilisation de composants existants 59
3.1.2.2 Interconnexion de modules décrits dans des langages différents 59
3.1.3 CONTRIBUTIONS 59
3.2 TRAVAUX ANTÉRIEURS 60
3.2.1 MODÉLISATION DE LA COMMUNICATION DANS LE CADRE DES SYSTÈMES HOMOGÈNES. 61
3.2.1.1 Modèle de communication homogène 61
3.2.1.2 Interconnexion des modules homogènes 62
3.2.1.3 Notion de canal abstrait homogène 62
3.2.1.4 Modélisation des unités de communication homogène 63
3.2.2 SYNTHÈSE DE LA COMMUNICATION DANS LE CADRE DES SYSTÈMES HOMOGÈNES 63
3.2.2.1 Fusion de canaux abstraits 64
3.2.2.2 Sélection de protocole et allocation des unités de communication 64
3.2.2.3 Synthèse d'interface 64
3.3 MODÉLISATION ET SYNTHÈSE DE LA COMMUNICATION DANS LE CADRE DES SYSTÈMES HÉTÉROGÈNES 65
3.3.1 MODÉLISATION DE LA COMMUNICATION DANS LE CADRE DES SYSTÈMES HÉTÉROGÈNES 65
3.3.1.1 Modèle de communication pour les systèmes hétérogènes 65
3.3.1.2 Modèle d'interconnexion des modules hétérogènes : introduction du concept de connecteur 66
3.3.1.3 Notion de canal abstrait hétérogène 66
3.3.1.4 Modélisation des unités de communication hétérogène 67
3.3.2 FLOT GLOBAL DE SYNTHÈSE DE LA COMMUNICATION POUR LES SYSTÈMES HÉTÉROGÈNES 68
3.3.2.1 Modèle d'entrée 68
3.3.2.2 Modèle produit par la synthèse de la communication 68
3.3.2.3 Méthode de synthèse de la communication 68
3.3.2.4 Outil de synthèse pour les systèmes hétérogènes 69
3.4 RÉSULTAT : L'EXEMPLE DU CONTRÔLEUR DE BRAS DE ROBOT 71
3.4.1 MODÉLISATION DU SYSTÈME DE CONTRÔLE DE MOTEUR 72
3.4.1.1 Flot de conception du système de contrôle de moteur 72

3.4.1.2 Modélisation du contrôleur PID 73
3.4.1.3 Modélisation du moteur électrique 75
3.4.1.4 Spécification du système global 77
3.4.2 SYNTHÈSE DES COMMUNICATIONS DU SYSTÈME DE CONTRÔLE DE MOTEURS 79
3.4.2.1 Protocoles utilisés pour la synthèse des communications 79
3.4.2.2 Transformation du modèle hétérogène SDL-Matlab en une architecture C-VHDL-Matlab 79
3.4.3 COVALIDATION DU SYSTÈME 82
3.4.3.1 Covalidation du système au niveau système 82
3.4.3.2 Covalidation du système au niveau architecture 83
3.4.3.3 Covalidation du système au niveau cycle 83
3.4.4 RÉSULTATS 84
3.5 CONCLUSION 85

COSIMULATION GÉOGRAPHIQUEMENT DISTRIBUÉE 87

4.1 INTRODUCTION 88
4.2 ETAT DE L'ART DE LA COSIMULATION 88
4.2.1 MODÈLES DE SYNCHRONISATION 88
4.2.1.1 Modèle maître / esclaves 89
4.2.1.2 Modèle distribué 89
4.2.2 MODÈLES TEMPORELS 89
4.2.2.1 Validation fonctionnelle 90
4.2.2.2 Validation temporelle 90
4.2.3 OUTILS EXISTANTS 91
4.2.3.1 Cosyma 91
4.2.3.2 Ptolemy II 91
4.2.3.3 MCSE 92
4.2.3.4 VCI 92
4.2.3.5 Seamless 92
4.2.3.6 Cossap 93
4.2.3.7 Coware N2C 93
4.3 COSIMULATION GÉOGRAPHIQUEMENT DISTRIBUÉE VS LOCALE 93
4.3.1 EXEMPLES DE SUPPORTS 93
4.3.2 TECHNIQUES DE COMMUNICATION 94
4.3.2.1 Communication inter-processus (I.P.C.) 94
4.3.2.2 Sockets 95
4.3.3 TECHNIQUES DE SYNCHRONISATION 95
4.3.4 BILAN DE LA COSIMULATION GÉOGRAPHIQUEMENT DISTRIBUÉE 95
4.4 PROTOTYPE MCI 95
4.4.1 CARACTÉRISTIQUES DU PROTOTYPE MCI 96
4.4.1.1 Validation fonctionnelle distribuée 96
4.4.1.2 Conception concurrente du système 96
4.4.1.3 Validation géographiquement répartie 96
4.4.1.4 Description de la coordination inter modules du système 96
4.4.2 ARCHITECTURE GLOBALE DU PROTOTYPE MCI 97
4.4.3 SIMULATEURS COMMERCIAUX ACCEPTÉS PAR LE PROTOTYPE MCI 98
4.4.4 LIEN AVEC L'OUTIL DE CONCEPTION CONJOINTE MUSIC 98
4.5 EXTENSION DU PROTOTYPE MCI AUX LANGAGES COSSAP ET VHDL (RTL) 98
4.5.1 INTERFACE COSSAP 99

4.5.1.1 Présentation de l'interface COSSAP 99
4.5.1.2 Modèle de comportement de l'interface COSSAP 99
4.5.1.3 Limitations de l'interface COSSAP 100
4.5.2 INTERFACE VHDL-RTL 100
4.5.2.1 Présentation de l'interface VHDL-RTL 100
4.5.2.2 Modèle de comportement de l'interface VHDL-RTL 100
4.5.2.3 Limitations de l'interface VHDL-RTL 101
4.6 EXEMPLES D'UTILISATION DE MCI ÉTENDU 101
4.6.1 COSIMULATION DE SYSTÈMES HÉTÉROGÈNES SDL – COSSAP 101
4.6.1.1 Le système hétérogène 101
4.6.1.2 Cosimulation SDL-COSSAP 102
4.6.2 COSIMULATION DE SYSTÈMES HÉTÉROGÈNES COSSAP – VHDL-RTL 103
4.6.2.1 Le système hétérogène 103
4.6.2.2 Cosimulation VHDL-COSSAP 103
4.7 EXPÉRIMENTATION INDUSTRIELLE, UN MODEM VDSL 104
4.7.1 INTRODUCTION AUX TECHNIQUES XDSL (X DIGITAL SUBSCRIBER LINE) 104
4.7.2 PRÉSENTATION DU MODEM VDSL EXPÉRIMENTAL 105
4.7.3 EXPÉRIMENTATION MENÉE 106
4.7.3.1 Répartition géographique de la simulation 107
4.7.3.2 Intégration d'un module VHDL-RTL à l'expérience 109
4.7.4 CONCLUSION 110
4.7.4.1 Interface avec un accélérateur Voyager - IKOS 110
4.7.4.2 Amélioration des performances des simulateurs VHDL actuels 111
4.8 CONCLUSION 111

CONCLUSION 113

RÉFÉRENCES 117

INDEX BIBLIOGRAPHIQUE 125

PUBLICATIONS 129

Table des illustrations

Figure 1 : Flot de conception Multilangage 20
Figure 3 : Bloc englobant deux processus et bloc hiérarchique 27
Figure 3 : Système de processus SDL 29
Figure 5 : Légende SDL 30
Figure 5 : Validation de la transition s2(x,y) 30
Figure 6 : Sauvegarde d'un signal 30
Figure 7 : Exécution d'une transition implicite 31
Figure 8 : Condition d'autorisation, signal continu 31
Figure 9 : Utilisation d'un bloc prédéfini générateur de signaux 34
Figure 10 : Inclusion d'un modèle dans un bloc 34
Figure 11 : Bloc utilisateur Matlab 35
Figure 12 : Schéma flot de données 38
Figure 13 : L'outil de spécification DBE 41
Figure 14 : Concepts de base utilisés pour la modélisation des concepts de base 44
Figure 15 : Exemple de modèle de système électronique 45
Figure 16 : Flot de conception sur exemple de système 48
Figure 17. Flot de conception multilangage 56
Figure 18. Niveaux d'abstraction des communications inter modules 58
Figure 19. Spécification au niveau système 63
Figure 20. Flot global de synthèse de la communication homogène 64
Figure 21. Exemple d'un connecteur 66
Figure 22. Les canaux abstraits hétérogènes 67
Figure 23. Unités de communication hétérogènes 68
Figure 24. Bibliothèque de définitions des connecteurs 69
Figure 25. Module muni d'un connecteur 70
Figure 26. Résultat de l'application de la primitive map hétérogène 71
Figure 27. Schéma global du contrôleur de bras de robot 72
Figure 28. Flot de conception du système de contrôle de moteurs 73
Figure 29. Principe de la correction 73
Figure 30. Algorithme du contrôleur 75
Figure 31. Représentation SDL du contrôleur 75
Figure 32. Algorithme du moteur 76
Figure 33. Spécification Matlab d'un moteur 77
Figure 34. Bibliothèque de connecteurs pour l'exemple de moteurs 77
Figure 35. Spécification hétérogène SDL-Matlab 78
Figure 36. Spécification du système mis à plat 80
Figure 37. Bibliothèque de protocoles de communication 80
Figure 38. Spécification au niveau architecture 81
Figure 39. Cosimulation SDL-Matlab 82

Figure 40. Cosimulation C-VHDL-Matlab ____ *83*
Figure 41. Cosimulation au niveau cycle ____ *84*
Figure 42. Résultats de la Cosimulation ____ *84*
Figure 43. Résultats de la synthèse de la communication ____ *85*
Figure 44 : Architecture du prototype MCI ____ *97*
Figure 45 : Bibliothèque interface COSSAP - bus de cosimulation ____ *99*
Figure 46 : Système hétérogène SDL – COSSAP ____ *102*
Figure 47 : Cosimulation SDL – COSSAP en cours d'exécution ____ *102*
Figure 48 : Système hétérogène COSSAP – VHDL RTL ____ *103*
Figure 49 : Système hétérogène COSSAP / VHDL – RTL en cours d'exécution ____ *103*
Figure 50 : Schéma global du modem VDSL ____ *105*
Figure 51 : Expérimentation menée sur le modem VDSL ____ *106*
Figure 52 : Simulation de référence ____ *107*
Figure 53 : Expérience avec deux simulateurs COSSAP ____ *107*
Figure 54 : Simulation COSSAP – COSSAP en mode local et en cours d'exécution ____ *108*
Figure 55 : Expérience avec deux simulateurs COSSAP et un simulateur VHDL (VSS – SYNOPSYS) ____ *109*
Figure 56 : Simulation avec deux simulateurs COSSAP et un VHDL en mode local ____ *109*

Chapitre 1

INTRODUCTION

Ce chapitre présente les motivations, les objectifs et les contributions de ce travail de recherche dans le domaine de la conception des systèmes hétérogènes multilangages.

1.1 Introduction

Ce travail s'inscrit dans le domaine de la conception multilangage pour les télécommunications. Les systèmes actuels sont d'une complexité sans cesse croissante et composés de modules de natures totalement différentes. Les différentes équipes de développement disposent de flots de conception propres et de langages de spécification différents ce qui rend difficiles les tâches de validation et de synthèse du système global. Les méthodes de conception actuelles proposent aux différentes équipes de développer leurs modules suivant un cahier des charges défini lors de la spécification mais la validation du système complet ne s'effectue que lors de la construction du prototype. Ce travail a pour but de proposer une méthode de conception multilangage s'appuyant sur un environnement de synthèse d'interface et de validation multilangage et ainsi fournir à l'ensemble des équipes de conception le support à un travail coopératif au travers de leurs outils commerciaux. En particulier, cette méthode permet de réduire le temps de conception par une validation rapide et très tôt de la spécification du système [49]. Ce document présente une étude sur les environnements de conception employés dans le domaine des télécommunications, un environnement de synthèse de la communication et un environnement de cosimulation. Ces éléments de base permettent de construire une méthode de conception multilangage adaptée aux nécessités actuelles.

1.2 Conception Multilangage

La conception multilangage est devenue très populaire grâce aux deux principales approches que sont la composition et la cosimulation.

L'approche par composition consiste à transformer les spécifications de chaque module en un modèle unifié. Cette approche autorise la vérification formelle de la consistance et de la cohérence globale du système. Les systèmes POLIS [4], [16], [85], Garden [88], JavaTime [115], OpenJ [118], SPI [119], FunState [102], SpecC [23], [117] et les approches de Wiles [113] et Zave [116] introduisent l'approche par composition dans la conception multilangage. POLIS et SPI utilisent un modèle unifié interne tandis que JavaTime et SpecC introduisent un nouveau langage de composition (respectivement Java et SpecC) utilisable directement par le concepteur.

L'approche basée sur la cosimulation consiste à interconnecter les environnements de conception associés à chaque module du système. Comparée à la composition, cette approche propose une intégration beaucoup moins profonde des modules mais tout à fait suffisante pour appréhender une spécification multilangage et permettre une conception modulaire. Cette approche implique l'utilisation des environnements associés à chaque module du système. La cosimulation permet de valider le système global durant toutes les étapes de la conception [90], [91].

Les principales phases de conception sont la synthèse des communications inter modules et le raffinement des modules du système. La plupart des outils existants dans ce domaine proposent seulement quelques raffinements et utilisent une spécification peu abstraite en entrée, typiquement, RTL pour le matériel ou C pour le logiciel. Coware [92], Seamless [52] et [103], sont des environnements de cette nature. Une description mixte VHDL et C leur sert d'entrée. Ces environnements proposent une cosimulation du système en cours de développement, cependant, seul Coware propose une méthode de synthèse d'interface entre les modules. La littérature recense très peu d'environnements de conception de plus haut niveau d'abstraction basés sur la cosimulation. On peut citer : RAPID [87], [89], Ptolemy [59], [86] et SEA [53]. Malheureusement la plupart ne proposent qu'une cosimulation du système [89] et [86].

Ce travail propose de dépasser ces limites en partant d'une spécification multilangage de haut niveau et en proposant des méthodes de raffinement et de synthèse d'interface multilangage de haut niveau. La méthode de synthèse de communication multilangage est basée sur des méthodes similaires aux

techniques employées dans les outils : LYCOS [55], CHINOOK [105] et [81]. La méthode de validation emploie une approche par cosimulation.

La méthode de conception présentée dans ce document adresse principalement les domaines suivants :

- Conception de systèmes embarqués complexes. Actuellement, un système complexe est défini par un ensemble de modules basés sur des concepts différents et interconnectés via des canaux de communication. Chaque module utilise un moyen de communication différent, selon la spécification du module.
- Réutilisation de composants existants. La complexité croissante des systèmes embarqués nécessite la réutilisation de modules existants logiciel/matériel afin de réduire le temps de mise sur le marché. Ces modules ont une interface spécifique bien souvent non modifiable, et des protocoles de communication spécifiques dont l'objectif est de satisfaire les contraintes de coûts et de performances.

La méthode de synthèse proposée dans ce travail permet l'intégration et la réutilisation des composants existants logiciels ou matériels. Cette méthode prend en compte l'interface définie pour le composant existant ainsi que le protocole de communication utilisé, et génère les interconnexions nécessaires à la communication entre les modules sans entrer dans les détails de la spécification du module existant. Ainsi, le concepteur peut modifier la fonctionnalité d'un composant existant ou remplacer un composant existant par un autre sans être obligé de refaire le processus de synthèse. La condition est qu'il doit conserver l'interface et le protocole de communication du composant existant.

1.3 Flot de conception

L'objectif de ce travail est d'aboutir au flot de conception formé par les cinq niveaux d'abstraction : service, message, driver, RTL et prototype (Figure 1).

Le **niveau service** représente la communication comme une combinaison de requêtes de services. Les différents modules communiquent par des requêtes de services, via des *réseaux abstraits* qui garantissent le routage et la synchronisation des connexions établies dynamiquement. La primitive de communication typique est une requête de service, par exemple 'imprimer (fichier)'. CORBA (Common Object Request Broker Architecture) [79] est un exemple de ce niveau d'abstraction.

Au **niveau message**, les différents modules du système communiquent via un réseau explicite de *canaux de communication*, qui sont dits *actifs*. En plus de la synchronisation, ces canaux peuvent disposer d'un comportement complexe, par exemple la conversion des protocoles spécifiques aux différents modules communicants. Les détails de la communication sont englobés par des primitives de communication de haut niveau (par exemple send/receive) et aucune hypothèse sur la réalisation de la communication n'est déposée. Des exemples de langages modélisant les concepts spécifiques de ce niveau sont : SDL (System Description Language) [104], UML (Unified Modeling Language) [95] et ObjecTime [78]. Certaines implémentations de StateCharts [41] peuvent être placées à ce niveau.

Le **niveau driver** est un niveau spécifique à la communication par des *fils abstraits* englobants des protocoles de niveau driver (registre, file). Un modèle de ce niveau implique le choix d'un protocole de communication et la topologie du réseau de connexions. Un exemple de primitive de communication spécifique est l'écriture d'une valeur ou l'attente d'un événement sur un port. Les langages employés à ce niveau d'abstraction sont : CSP [45], LOTOS [61], SystemC 1.1 [101], Cossap [99] et StateCharts [41].

Au **niveau RTL,** la communication est réalisée par des *fils ou bus physiques*. L'unité de temps devient le cycle d'horloge, les primitives de communication sont du type set / reset sur des ports activés lors d'un nouveau cycle d'horloge. Les langages les plus utilisés pour la modélisation de systèmes à ce niveau sont SystemC 0.9-1.0, Verilog [110] et VHDL [112].

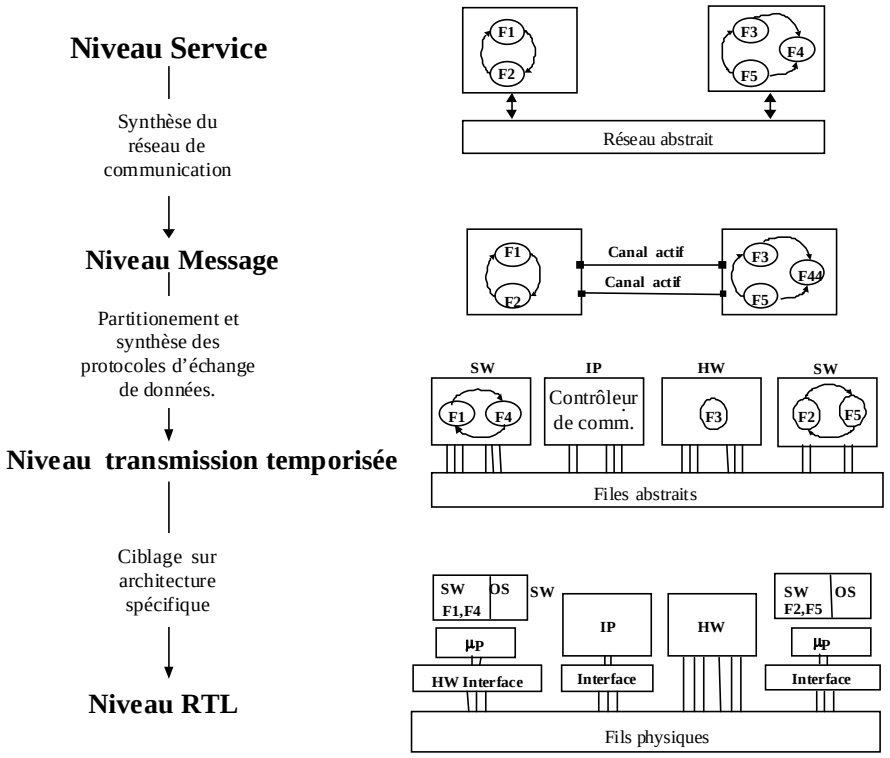

Figure 1 : Flot de conception Multilangage

La méthode de conception consiste à raffiner la spécification d'un système en une version moins abstraite. Pour ce faire le raffinement est décomposé en deux parties : les modules et les communications. Un environnement de cosimulation permet de valider le système entier à chaque étape de raffinement.

1.4 Contributions

Ce document présente trois principales contributions : une étude des méthodes de spécification et de conception des systèmes électroniques, une méthode de spécification et de raffinement des communications pour les systèmes à spécification hétérogène et un environnement de cosimulation développé au sein de l'équipe System Level Synthesis [100].

1.4.1 Conception de systèmes électroniques

Cette étude se compose de deux parties. La première partie présente une étude des langages existants de spécification de système électronique. Cette étude apporte une vue commune des méthodes disponibles de spécification. La seconde partie propose une approche pragmatique de la conception de système électronique en décomposant les communications du comportement des modules du système. Pour cela, des concepts syntaxiques et des niveaux d'abstraction sont définis. Les méthodes actuelles sont analysées au travers de ces définitions et une méthode de spécification est présentée.

1.4.2 Raffinement des communications

L'étude des différentes formes de spécification a permis de montrer qu'il est nécessaire d'employer des interfaces spécifiques entre des modules hétérogènes. Une méthode de synthèse d'interface multilangage est nécessaire afin d'appréhender la cosimulation et la synthèse de systèmes hétérogènes. Cette méthode de synthèse de communication inter langage tient compte des différents formalismes de description,

modèles de calcul et schémas de communication. Les objectifs principaux de cette méthode sont : permettre la synthèse de communication inter langage, permettre la modélisation du comportement du système indépendamment de la communication, permettre la réutilisation des modèles de communication existants dans une bibliothèque et permettre à l'utilisateur de choisir entre différents protocoles de transmission de données. L'outil de synthèse de communication prend en entrée le fichier de configuration utilisé par l'outil de cosimulation et une interface générique prédéfinie, puis produit une interface adaptée. Le résultat est un ensemble de modules communiquant par des bus et éventuellement un composant de contrôle.

1.4.3 Validation des étapes de conception par simulation

Le besoin de validation globale d'un système hétérogène amène à proposer un environnement de cosimulation. La solution de cosimulation permet de connecter les simulateurs propres à chaque langage et ainsi profiter des possibilités de chacun. L'environnement proposé (prototype XXI) permet la connexion de modules C, VHDL, Matlab, SDL et est facilement extensible à d'autres langages. Cette solution permet de valider la fonctionnalité d'un système aux niveaux *message* et *driver* du flot de conception présenté. Cette approche est basée sur un protocole de communication appelé bus de cosimulation. Le bus de cosimulation est responsable du transfert des données entre les différents simulateurs. L'implémentation du bus de cosimulation est basée sur les fonctions standard du système de communication par socket (BSD-UNIX). L'environnement utilise en entrée un fichier de configuration défini par l'utilisateur qui décrit les interconnections entre les sous-systèmes.

A partir de ce fichier, les interfaces entre les sous-systèmes et le bus de cosimulation sont générées automatiquement, permettant la cosimulation de l'ensemble des sous-systèmes. L'environnement actuel de cosimulation nommé MCI [43] est commercialisé par la société AREXSYS et à servi de support à l'expérience VDSL menée en collaboration avec le centre de recherche France Télécom (CNET).

1.5 Plan du document

Ce document se compose de trois chapitres. Le chapitre 2 présente une étude des méthodes de conception des systèmes électroniques et présente une nouvelle approche. Le chapitre 3 introduit une méthode de spécification et de raffinement des communications dans le cadre des systèmes hétérogènes. Le chapitre 4 présente les contraintes de la cosimulation géographiquement distribuée et du domaine du traitement du signal. L'adaptation de l'environnement commercial de cosimulation MCI à ces contraintes et l'expérience menée sur un modem VDSL sont présentées. Le dernier chapitre présente les conclusions sur ce travail mené au sein de l'équipe System Level Synthesis ainsi que les perspectives à moyen et long termes.

Chapitre 2

SPECIFICATION ET CONCEPTION DE SYSTEMES ELECTRONIQUES

Ce chapitre présente une étude des langages de spécification décomposée en deux parties. La première partie propose une étude approfondie de trois langages usuels et selon les concepts employés dans la littérature. Cette partie permet de dégager une vision globale et commune sur les langages et outils de spécification de systèmes électroniques. La seconde partie propose une approche globale, synthétique et pragmatique de la conception des systèmes électroniques. Cette étude est issue de la collaboration avec Gabriela Nicolescu et Olivier Meunier.

2.1 Introduction

Aujourd'hui, les systèmes complexes font appel à des domaines scientifiques aussi variés que la micro-électronique, l'électronique analogique, l'informatique, la mécanique, l'optique, etc. Chaque partie d'un système est confiée à une équipe spécialisée. Chaque équipe emploie une méthode, des outils et des langages spécifiques et dédiés à son domaine d'application. Lors de la conception d'un système, plusieurs langages de description ou spécification sont employés.

Les langages de spécification actuels ne permettent de décrire qu'une partie d'un système et seulement pour quelques phases de conception. Aucune solution ne permet de décrire un module tout au long de sa conception, et encore moins de solutions ne permettent d'aborder globalement la conception d'un système complet.

Le premier objectif de ce travail est une étude des solutions existantes d'un point de vue technique et à partir des concepts employés dans leur domaine d'application. Le second objectif est une analyse des concepts de description employés dans chaque domaine et une méthode de conception articulée autour de niveaux d'abstraction et permettant une approche globale d'un système et de son raffinement jusqu'au prototype.

Ce chapitre s'articule en deux volets. La première partie présente une étude de langages employés dans le domaine des systèmes de télécommunication. La seconde partie propose une approche de la conception des systèmes électroniques basée sur des concepts communs à tous les domaines d'application et sur des niveaux de description autorisant une plus ou moins grande abstraction.

2.2 Langages de description de systèmes électroniques

Cette section présente une analyse de langages de description de systèmes électroniques. Les langages choisis sont issus du domaine des télécommunications et représentent une grande partie des classes de langages employés.

Dans le domaine du traitement du signal l'environnement COSSAP [99] utilise un modèle flot de données dynamique pour la modélisation. L'outil SIMULINK associé à Matlab [68] permet de décrire le comportement physique de l'environnement (mécanique, mais pas uniquement) d'un système. Il utilise un modèle flot de données synchrone. La description de protocoles dans le domaine des télécommunications fait appel au langage SDL [7], [26], [28], [36]. La méthode de spécification est basée sur l'utilisation de machines d'états finis concurrentes et communicantes. L'environnement ObjectGEODE supporte le langage SDL et permet de simuler le système décrit.

Cette section propose une vision commune des langages de spécification et des environnements de cosimulation motivée par le fait que tous modélisent le comportement de modules concurrents. L'étude porte sur la structure d'une spécification dans chaque langage ou environnement, sur les formes de communications proposées et le comportement des modèles construits. Pour chacune des méthodes, l'environnement employé pour l'étude est précisé.

Les concepts employés sont tout d'abord détaillés. Les langages SDL, Matlab et COSSAP sont ensuite analysés.

2.2.1 Concepts employés pour l'étude

Cette partie présente les aspects retenus pour l'étude des méthodes de spécification.

2.2.1.1 Principaux concepts employés dans les langages de spécification système

Les principaux concepts de spécification dégagés de la littérature : concurrence, hiérarchie, communication et synchronisation sont ici détaillés.

2.2.1.1.1 Concurrence

Le concept de concurrence représente l'exécution parallèle de plusieurs tâches. Il est caractérisé par la dimension du grain d'exécution parallèle et par l'expression de l'ordre des opérations. Le grain exprime la taille des tâches parallèles, elle peut être : du niveau « bits » (un additionneur n-bits), du niveau de l'opération (plusieurs unités fonctionnelles dans un flot de données), du niveau du processus (spécification multiprocessus), ou du niveau du processeur (modèle de processeurs distribués). La concurrence peut être exprimée par l'ordre d'exécution des opérations ou par un schéma flot de données. Les modèles orientés contrôle emploient une spécification explicite de séquence des opérations. Les machines d'états finis concurrentes et les modèles basés sur CSP sont des exemples typiques. Les modèles orientés données expriment la concurrence par les dépendances de données. Les schémas flot de données et les architectures sont les exemples typiques.

2.2.1.1.2 Hiérarchie

La hiérarchie permet d'aborder la complexité des systèmes de taille trop importante. Un système est décomposé en modules de taille plus faible et plus abordable. Deux formes de hiérarchie se dissocient : comportementale et structurelle. La hiérarchie comportementale permet de cacher des « sous comportements locaux» dans un module. Les procédures et les sous-états permettent d'exprimer cette forme de hiérarchie. La hiérarchie structurelle permet de décomposer un système en sous systèmes communicants. Chaque composant est défini à l'intérieur de limites bien définies. Les interactions sont spécifiées à un niveau signal (fils) ou par le biais de canaux abstraits englobant un protocole.

2.2.1.1.3 Communication

Les communications expriment les échanges de données ou de signaux de contrôle entre les différents modules d'un système. Les deux principaux modèles de communication sont le passage de messages et la mémoire partagée. Le passage de message consiste en l'utilisation d'opérations spécifiques d'envoi et de réception de messages. Afin d'échanger des informations via une mémoire partagée, les différents modules doivent connaître l'emplacement de cette dernière. La mémoire partagée est très similaire d'emploi à une boite à lettre.

Un modèle hybride est souvent utilisé, l'appel de procédure à distance. Le concept reprend exactement l'appel local de procédure. Les paramètres en entrée et en sortie et l'appel sont transmis selon l'implémentation, par l'un des modèles précédents.

2.2.1.1.4 Synchronisation

La synchronisation permet de coordonner l'exécution des différents modules d'un système. Deux termes sont employés pour spécifier le type de synchronisation, synchrone et asynchrone. Ces termes peuvent porter sur le modèle d'exécution des modules du système ou sur la forme de communication utilisée. Un modèle d'exécution est qualifié de synchrone lorsque l'avancement de chaque module est lié aux autres. Par opposition, le modèle asynchrone précise que les modules ne sont pas explicitement contraints dans leur exécution. Le mode synchrone de communication indique une communication sûre puisque la transmission n'a lieu que lorsque les deux (ou plus) parties en jeu sont prêtes à échanger des informations. A l'opposé, le mode de communication asynchrone indique une forme de communication où l'émetteur comme le récepteur ne se soucie pas de l'état de l'autre, seule l'information à transmettre est importante. Il est intéressant de remarquer que chaque modèle peut être modélisé à partir de l'autre.

2.2.1.2 Modèles d'exécution des langages de spécification système

Les modèles d'exécution peuvent être regroupés sous deux orientations principales, flot de données et flot de contrôle.

2.2.1.2.1 Orientation flot de données : Schéma de blocs

Un flot de données est modélisé par un ensemble de blocs fonctionnels connectés, parfois appelés acteurs. Le réseau de l'ensemble des blocs forme un ordre partiel. Un bloc est invoqué lorsque suffisamment de données sont présentes en entrée. A chaque exécution, un bloc consomme des entrées et produit un résultat en sortie. Deux modèles de flots de données existent :

- **Flot de données synchrone,** le nombre de données consommées et produites est constant d'une invocation à l'autre.
- **Flot de données dynamique,** les blocs sont complètement indépendants et le nombre de données consommées et produites peut varier d'une invocation à l'autre.

Le flot de données est une méthode de spécification bien adaptée à la modélisation de traitement de données, elle permet de représenter la chaîne de traitements opérés. Bien que le caractère synchrone ou dynamique d'un flot de données ne préjuge pas de l'implémentation, un flot de données synchrone est le point de départ d'un module matériel et inversement, un flot dynamique est plus orienté logiciel car le caractère dynamique implique un certain contrôle.

2.2.1.2.2 Orientation flot de contrôle : Machine d'états

Une machine d'états finis est un ensemble d'états liés par des transitions. Un état correspond à une action et les transitions explicitent l'ordre d'exécution. Selon le type de machine d'états finis, le traitement des données, l'émission ou la réception d'un signal est supporté par un état ou une transition.

Afin d'appréhender la complexité des systèmes actuels, les machines d'états finis sont hiérarchiques. Un état peut être une entité simple ou une nouvelle machine d'états finis.

Bien souvent une seule machine d'états finis ne suffit pas à modéliser le comportement d'un module du système, plusieurs parallèles et communicantes sont nécessaires.

Les environnements de spécification sous la forme de machine d'états finis fournissent de nombreux outils de vérification formelle. Cependant, ils sont habituellement pauvres en traitement de données. Les descriptions de ce type sont réalisables en logiciel ou matériel.

2.2.2 SDL

Le langage SDL (Specification and Description Language) [7], [21], [28] est dédié à la modélisation de systèmes temps - réel distribués et plus particulièrement aux télécommunications (protocoles). Il est né de la nécessité de spécifications précises entre plusieurs équipes et est maintenant standardisé par le CCITT.

Le langage SDL permet de spécifier un système sous la forme abstraite de processus concurrents, communicant au travers de signaux.

La section prochaine présente les concepts de la structure d'une modélisation SDL. La section 2.3.1.2 aborde les communications entre processus. La troisième section décrit le comportement global et local d'un processus. La dernière section présente l'environnement de développement SDL : ObjectGEODE.

2.2.2.1 Structure

Cette section détaille les concepts de structure d'une spécification SDL. La structure générale, puis les concepts avancés SDL sont présentés.

2.2.2.1.1 Structure hiérarchique

Un système SDL est défini par un ensemble d'automates d'états finis communiquants [35], [59]. Il peut être ouvert, connecté à son environnement ou fermé, sans interaction avec l'extérieur. Le langage SDL permet une description hiérarchique d'un système. L'entité de plus haut rang est appelée système (system).

Un système est décrit par une structure hiérarchique de blocs. Comme le montre la Figure 2, un bloc peut se présenter sous deux formes :

- Composé d'autres blocs interconnectés formant une hiérarchie,
- Composé de processus (process) interconnectés formant un bloc terminal.

Les blocs sont interconnectés par l'intermédiaire de canaux transportant les signaux émis par les processus. Les interconnections cachent des files d'attentes infinies.

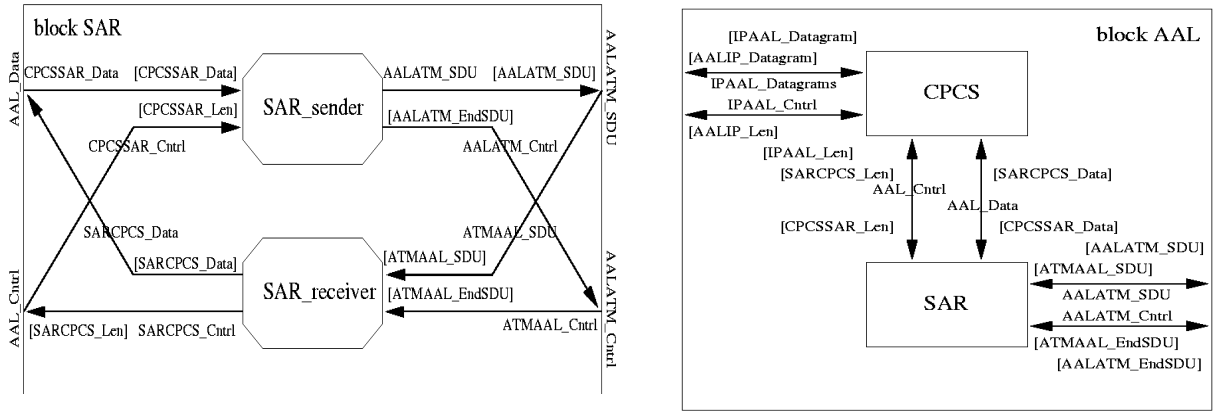

Figure 2 : Bloc englobant deux processus et bloc hiérarchique

Un processus est formé d'un automate d'états finis, d'un ensemble de variables locales et d'un ensemble de signaux d'entrées/sorties. L'automate d'états finis modélise le contrôle de l'exécution du processus tandis que les tâches effectuées lors des transitions modélisent le traitement des données.

Un automate d'états finis est défini par un ensemble d'états et de transitions entre ces derniers. Dans SDL, les entrées et sorties ont lieu lors des transitions. Une transition est une séquence de tâches de traitements et d'entrées/sorties.

Deux modes de description sont disponibles, l'un textuel, l'autre graphique. Dans ce dernier, les états sont modélisés par des rectangles arrondis, tandis que les transitions sont modélisées sous la forme de rectangles et de flèches (voir Figure 4).

2.2.2.2 Communications

Les processus SDL communiquent par le biais de signaux transmis au travers de routes et canaux liant les processus et les blocs. Les concepts avancés de variables et de procédures exportées sont aussi des procédés de communication acceptés. Ces trois méthodes de communications sont développées par la suite.

2.2.2.2.1 Signaux

Dans le langage SDL, la communication est basée sur le modèle de passage asynchrone de messages entre les processus. Les messages sont appelés signaux car ils peuvent porter une donnée ou non. Les signaux transportant des données sont typés. Les signaux sont acheminés par le biais des routes et des canaux assurant des connexions point à point.

2.2.2.2.1.1 Routes

Les routes assurent les connexions entre deux processus d'un même bloc ou entre un processus et la frontière du bloc contenant ce dernier. Dans ce cas la route est connectée à un et un seul canal.

Plusieurs signaux peuvent emprunter la même route qui peut être mono ou bidirectionnelle.

L'ordre d'envoi des signaux au travers d'une route est préservé et le temps d'acheminement est nul, mais l'ordre d'arrivée de signaux acheminés par des routes différentes ne peut être prédit.

2.2.2.2.1.2 Canaux

Les canaux assurent les connexions entre blocs et les connexions entre un bloc et la frontière du système, l'environnement. A la frontière d'un bloc, plusieurs routes peuvent fusionner en un seul canal, et inversement, un canal peut se décomposer en plusieurs routes.

Comme une route, un canal peut être mono ou bidirectionnel et plusieurs signaux peuvent être transportés par un même canal. Un canal assure la fonction de routage d'un signal à destination d'un processus particulier.

Le transport d'un signal à travers un canal peut subir un retard indéterminé. L'ordre des signaux au sein d'un canal ne peut être prédit. En effet, un canal peut employer des routes différentes pour deux messages de même destination, l'ordre d'arrivée est alors inconnu.

2.2.2.2.2 Variables exportées

Le langage SDL supporte l'importation de la valeur d'une variable d'un processus par un autre. Le processus possédant la variable doit autoriser la transaction par la commande « export ». La demande de valeur s'effectue lors d'une tâche spécifique « import », la valeur courante est alors retournée au processus requérant.

2.2.2.2.3 Procédures exportées

Dans le langage SDL, la notion de variables exportées est étendue aux procédures. Un processus peut appeler une procédure appartenant à un autre processus si ce dernier en autorise l'appel. La requête est traitée comme un appel de procédure à distance couramment appelé Remote Procedure Call.

2.2.2.3 Comportement

Cette section présente le comportement d'un système SDL. En premier lieu, la globalité du système est abordée puis le modèle d'exécution interne d'un processus est détaillé.

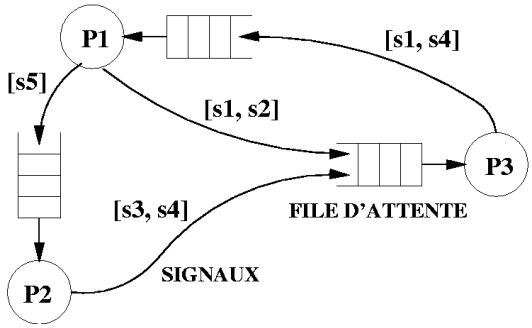

Figure 3 : Système de processus SDL

2.2.2.3.1 Comportement global

Un système décrit en langage SDL est un ensemble de processus concurrents communicants. Un processus est un automate d'états finis communicant de manière asynchrone avec les autres au travers de signaux. A chaque processus est associé une file d'attente de signaux infinie (Figure 3). Le processus consomme les signaux dans leur ordre d'arrivée car la file d'attente suit le comportement First-In-First-Out.

Ce modèle asynchrone faiblement couplé semble être bien adapté à la modélisation de systèmes distribués.

2.2.2.3.2 Comportement d'un processus

Un processus est constitué d'un automate d'états finis, composé d'un ensemble d'états et de transitions. Lors de l'initialisation du processus, l'automate est placé dans l'état déclaré comme initial (start). A chaque état, une transition valide est exécutée et place l'automate dans un nouvel état.

L'exécution d'une transition consiste à traiter les tâches portées par celle-ci. Plusieurs types de tâches (task) sont disponibles :

- La manipulation de variables,
- L'appel de procédure,
- L'émission de signaux.

Les transitions sont généralement gardées par la réception d'un signal, la transition n'est validée qu'une fois le signal correspondant reçu. Ainsi, un état source de plusieurs transitions gardées par des signaux est un état d'attente. L'arrivée de l'un des signaux valide une transition alors immédiatement exécutée.

Tout signal reçu par un processus est pris en compte immédiatement par une transition implicite si aucune n'est spécifiée pour ce signal. Une transition implicite consomme un signal et replace l'automate dans le même état. Cette particularité rend les systèmes modélisés en langage SDL réactifs.

Une tâche particulière appelée sauvegarde (save) permet de conserver le message d'une transition implicite afin de le traiter ultérieurement. Ce procédé permet de construire un ordre de priorité sur les signaux.

La garde d'une transition par un signal peut être complétée par une condition d'autorisation. Une condition booléenne est ajoutée dans la validation d'une transition. Ainsi une transition invalidée par la condition booléenne voit son signal sauvegardé et l'automate retourne dans le même état.

Une transition peut être gardée par une condition booléenne seule. Une telle garde est appelée signal continu. Une transition gardée par un signal continu n'est prise en compte que si aucune autre transition, gardée ou non, n'est validée. Le cas de plusieurs transitions gardées par des signaux continus est résolu par l'affectation de priorités.

La Figure 4 montre la représentation adoptée pour la description graphique de processus.

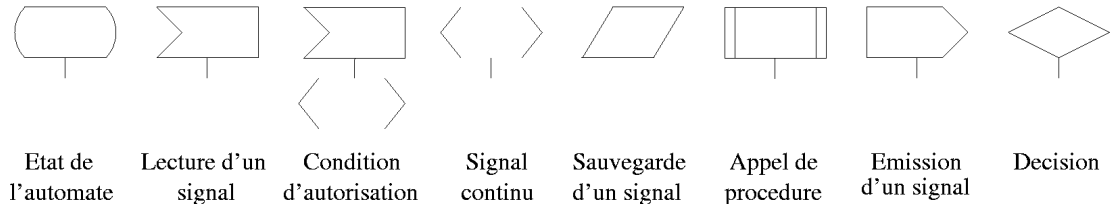

Figure 4 : Légende SDL

Un exemple d'exécution de processus SDL :

- Le processus se trouve dans l'état *st1* d'attente d'un signal (*s2(x,y)* ou *s3(x)* ou *s1*). Le signal *s2* se trouvant dans la file d'attente valide la transition gardée par *s2(x,y)*, permet son exécution et replace le processus dans l'état *st1* (Figure 5).
- Le signal *s4* se présente sur la file d'attente et est consommé par une transition implicite de l'état *st1* (Figure 7).
- Le signal *s1* valide la transition de sauvegarde du signal. Le signal *s1* est sauvé et sera restauré lorsque la file d'attente des signaux sera vide. Le signal *s3(3)* valide ensuite la transition menant à l'état *st2* et l'exécute (Figure 6).
- L'état *st2* est un état d'attente, source de trois transitions gardées. La première transition est gardée par le signal *s1*, la seconde par le signal *s4* complété par la condition d'autorisation *z=7* et la dernière est gardée par un signal continu *not b1*, une condition booléenne toujours valuée (Figure 8).

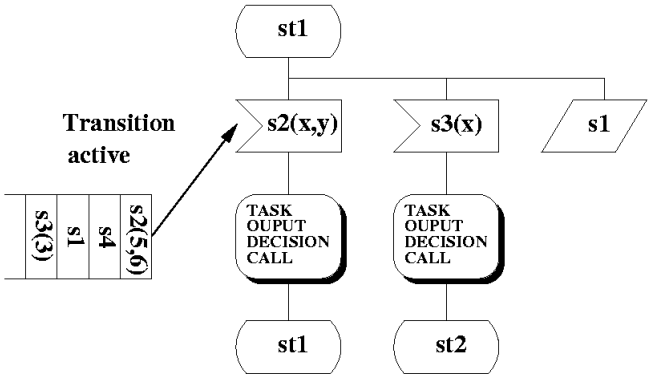

Figure 5 : Validation de la transition s2(x,y)

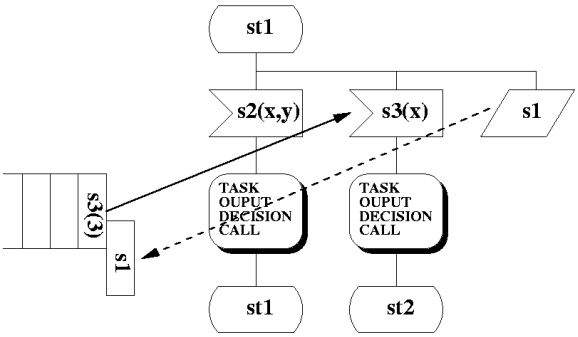

Figure 6 : Sauvegarde d'un signal

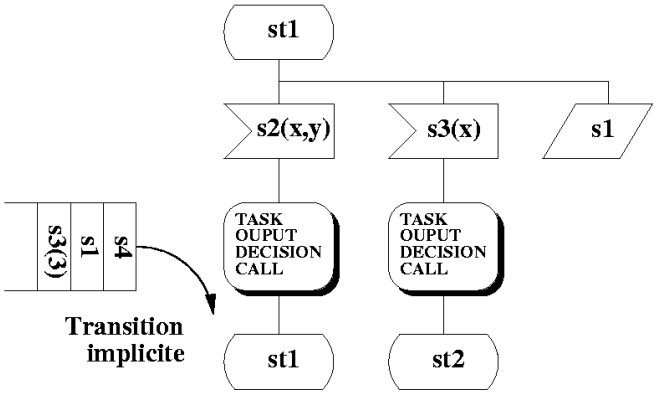

Figure 7 : Exécution d'une transition implicite

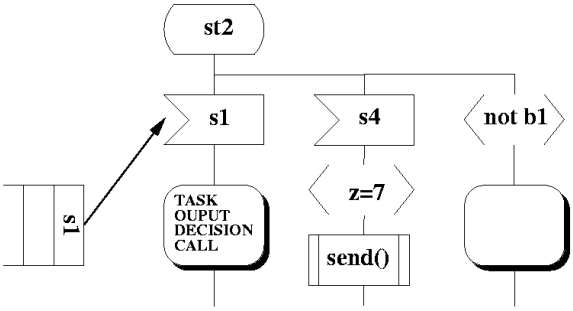

Figure 8 : Condition d'autorisation, signal continu

2.2.2.3.3 Concepts avancés SDL

Les différents concepts de structure d'une description SDL sont :

2.2.2.3.3.1 Transitions implicites

Une transition implicite est créée pour chaque signal pour lequel aucune transition n'est précisée. La transition implicite ne change pas d'état et son unique effet est de consommer un signal de la file d'attente. Les transitions implicites permettent d'utiliser SDL comme un langage de description de systèmes réactifs, tout signal peut être pris en compte immédiatement.

2.2.2.3.3.2 Procédures

Une procédure est un automate d'états finis, elle définit des variables locales et partage la file d'attente du processus auquel elle appartient

L'appel d'une procédure est effectué lors d'une transition d'état d'un automate par une tâche spécifique. Ce concept introduit la notion de machine d'états finis hiérarchique. La définition proposée par SDL laisse le choix au concepteur de la sémantique d'exécution entre l'appelé et l'appelant. Cependant, cette hiérarchie n'autorise pas de parallélisme.

2.2.2.3.3.3 Services

Un processus peut être formé par un ensemble de services. Un service est un automate d'états finis comme nous avons vu précédemment. Ce concept introduit la concurrence entre automates au sein d'un processus.

Les services sont exécutés pas à pas avec priorité au processus qui peut consommer le signal en entrée du processus.

2.2.2.3.3.4 Exceptions

Le langage SDL fournit une facilité syntaxique qui permet de définir le traitement d'exceptions dans un automate. L'état (state *) représente tous les états de l'automate, sauf ceux explicitement exclus. Cette facilité permet de définir simplement le traitement d'un signal dans tous les états d'un automate. L'état « nextstate » (-) fournit la possibilité de revenir dans l'état où a eu lieu le traitement de l'exception.

2.2.2.3.3.5 Types de données abstraits (Abstract Data Types)

Un processus peut définir des variables sur lesquelles il est possible d'effectuer des opérations arithmétiques simples.

Les types prédéfinis dans le langage SDL se limitent aux types communs. Le langage SDL permet la définition de nouveaux types et des opérateurs associés. Cette facilité permet au concepteur utilisant SDL de détailler le fonctionnement d'un système et non de rester à un niveau très abstrait qu'il serait difficile de traduire automatiquement vers un langage de spécification moins abstrait.

Plusieurs méthodes sont disponibles pour la définition de nouveaux types et opérateurs :

- Une méthode axiomatique par un ensemble d'équations,
- Une méthode algorithmique.
- L'utilisation d'un langage extérieur tel que C.

L'utilisation d'opérateur définis en C permet de bénéficier de toute la souplesse de description de ce langage. Cette option permet de travailler sur des données de types complexes, difficiles à traiter directement en SDL.

2.2.2.3.3.6 Caractère dynamique

Deux méthodes sont utilisées pour créer un processus : une méthode statique, le processus est créé lors de l'initialisation du système, une méthode dynamique, un processus est créé lors de l'exécution du système.

Afin de maîtriser la vie d'un processus, les commandes de création et de suppression sont disponibles. Lors de la création de processus, il est possible de passer des paramètres au processus. La suppression d'un processus se déroule par son suicide avec l'instruction « stop ». Un processus possède une adresse unique permettant de l'identifier. L'instruction « self » lui permet d'obtenir son adresse.

2.2.2.4 Environnement de conception SDL

Cette section introduit l'environnement de conception et de simulation de systèmes SDL employé pour l'étude. L'environnement ObjectGEODE comporte différents modules, ne sont présentés que les modules directement rattachés à SDL.

2.2.2.4.1 Environnement ObjectGEODE

L'environnement ObjectGEODE résulte de la combinaison de l'environnement GEODE pour SDL et de l'environnement LOV pour OMT, tous deux en provenance de la société Verilog. ObjectGEODE est composé des outils suivants :

- Un environnement SDL, incluant un éditeur graphique, un simulateur et un générateur de code.
- Un environnement OMT, incluant un générateur de squelette C++.
- Un éditeur MSC.
- Un outil de suivi de projet, permettant de diviser une description OMT en plusieurs fichiers.

2.2.2.4.2 Environnement SDL

L'éditeur fournit une interface graphique de construction de systèmes SDL. Une palette des objets disponibles à chaque instant permet de disposer les objets et de les connecter. A l'intérieur d'un processus, la palette propose les diverses options du langage, à savoir : la création d'états, l'affectation de tâches aux

transitions. La seule limitation dans la construction est l'appel de procédures à distance, non mis en œuvre actuellement.

Le simulateur s'appuie sur la sémantique du langage SDL. Cependant la concurrence est simulée par l'exécution entrelacée des différents processus. La priorité est donnée au processus disposant de signaux en entrée alors en situation d'exécuter une transition.

Le simulateur offre les options suivantes :

- La trace d'une simulation peut être conservée pour être exécutée à nouveau.
- La commande « undo » permet de revenir en arrière dans l'exécution.
- Les extensions SDL :
  - Le rendez-vous
  - La diffusion et la diffusion sélective
  - La possibilité d'intégration de code externe sous la forme de types et fonctions C (Types de données abstraits).

2.2.3 Matlab

Matlab est un environnement d'évaluation de modèles numériques. Des bibliothèques de fonctions pour plusieurs domaines en mathématiques numériques sont proposées et différents formats de visualisation sont fournis, en particulier, graphiques. Matlab inclut une interface avec le langage C permettant une ouverture sur d'autres types de langages. Dans le domaine des systèmes de traitement du signal Matlab est utilisé en conjonction avec l'environnement SIMULINK. Option venant compléter le noyau MATLAB, Simulink fourni une interface graphique pour la modélisation de systèmes dynamiques sous forme de schémas – blocs. Grâce aux nombreux blocs de base fournis, il est possible de créer des modèles rapidement et clairement, sans écrire une seule ligne de code. A partir des modèles Simulink, des fonctions sophistiquées de simulation et d'analyse permettent d'obtenir des résultats rapides et précis : méthodes d'intégration pour systèmes à pas fixe ou variable, simulation interactive avec visualisation des signaux, simulations de Monte-Carlo, définition des points d'équilibre stable, linéarisation, etc.

Cette étude porte sur l'environnement utilisé par les concepteurs de circuits de traitement du signal formé par l'association de Matlab et SIMULINK.

2.2.3.1 Structure

Simulink fournit une interface graphique pour la modélisation de systèmes dynamiques sous forme de schémas – blocs. Le contrôle et la modélisation sont aisés ; les fonctions de transfert sont écrites sous la forme de blocs et les liaisons sont réalisées par des arcs orientés. Différents types de signaux peuvent être générés et visualisés à l'aide d'instruments virtuels.

Différents types de blocs sont disponibles :

- **Bloc issu d'une bibliothèque fournie**. L'utilisateur emploie un bloc prédéfini et disponible dans une bibliothèque standard Simulink. Des bibliothèques pour toutes sortes de problèmes numériques sont proposées. A titre d'exemple, on peut trouver Figure 9 une utilisation de blocs prédéfinis. La fenêtre en haut à droite montre les différentes bibliothèques disponibles, la fenêtre en bas à droite détaille la bibliothèque *Sources* et la fenêtre à gauche présente un schéma utilisant un générateur de signaux issu de cette dernière.

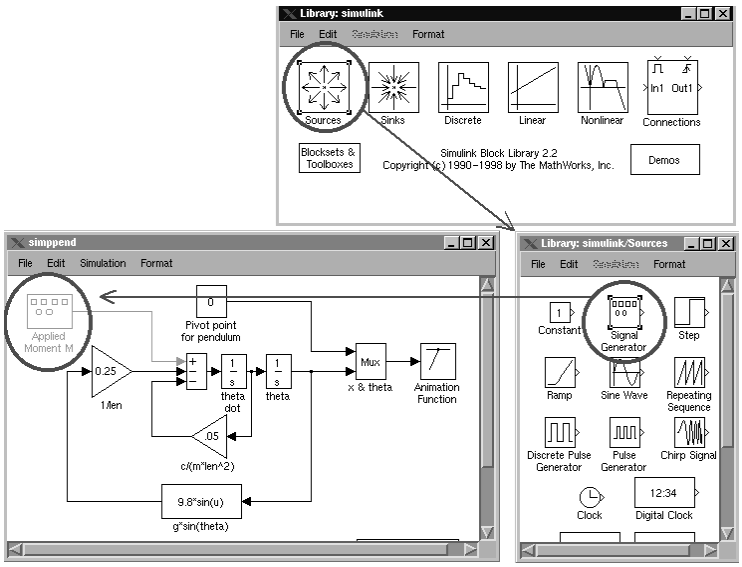

Figure 9 : Utilisation d'un bloc prédéfini générateur de signaux

- **Bloc « hiérarchique ».** Un modèle construit à l'aide d'un assemblage de blocs élémentaires peut être englobé ou masqué par un bloc. Le modèle masqué doit comporter les ports d'entrée et de sortie correspondant aux ports du bloc « hiérarchique ». La hiérarchie établie permet d'exprimer des comportements complexes en un seul bloc. Un tel bloc peut enrichir les bibliothèques disponibles sous Simulink afin d'être réutilisé dans un nouveau schéma. La Figure 10 montre un bloc défini par masquage d'un modèle complexe.

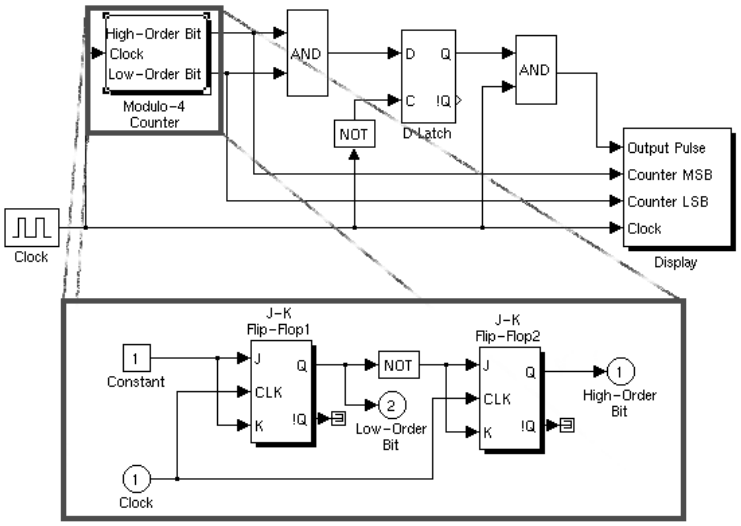

Figure 10 : Inclusion d'un modèle dans un bloc

- **Bloc utilisateur Matlab.** Un bloc peut être décrit dans le langage Matlab. Pour ce faire, l'utilisateur doit se conformer aux règles d'écriture d'une fonction Simulink, une « fonction S ». Ce type de blocs est très utilisé pour construire les interfaces utilisateur d'un modèle Simulink. La Figure 11 montre un bloc utilisateur défini à l'aide du langage Matlab. Ce concept est très important pour l'extensibilité du langage. Il sera utilisé au chapitre 4 dans le développement de l'interface de cosimulation.

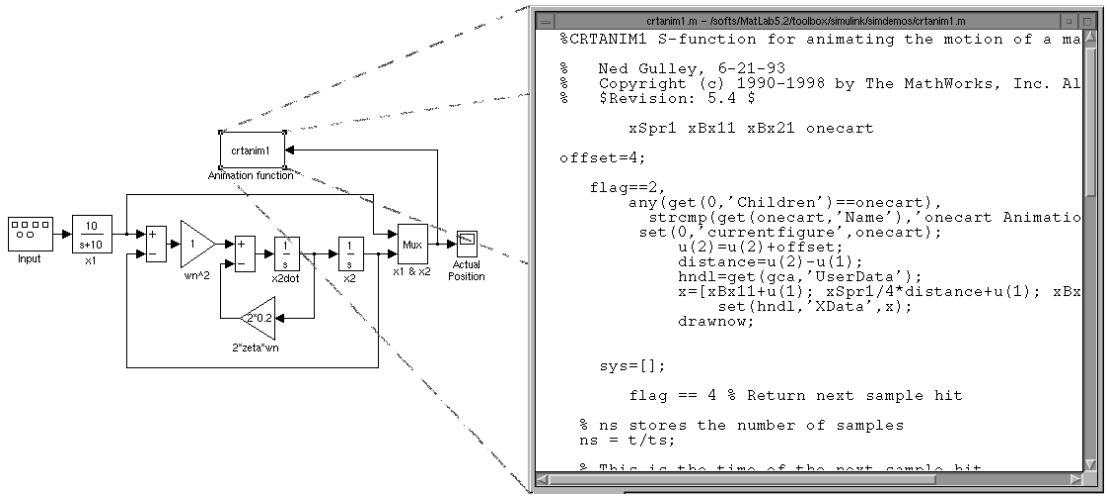

Figure 11 : Bloc utilisateur Matlab

Fonction S :

A chaque création d'un système, Simulink génère un fichier décrivant le comportement du système, dit fonction S. Cette fonction est une fonction Matlab dont la syntaxe est :

Sys = Nom_systeme(T, X, U, Flag)

Nom_systeme: nom du système crée sous Simulink,

T : temps,

X : vecteur d'état du système,

U : vecteur d'entrée du système,

Flag : paramètre de contrôle des résultats de la fonction.

Cette syntaxe générique permet de représenter la simulation de systèmes continus, discrets ou mixtes.

L'architecture ouverte permet d'étendre l'environnement de simulation par :

- la création de blocs personnalisés et de bibliothèques de blocs avec les icônes et interfaces de l'utilisateur, à partir du code MATLAB, Fortran ou C, ou bien de façon graphique.
- l'intégration de code Fortran ou C existant pour récupérer les modèles validés.
- la génération de code C à partir des modèles avec le Real-Time Workshop (optionnel).

2.2.3.2 Communications

Les communications entre les blocs d'un système Simulink sont modélisées par des arcs orientés. Un arc assure la connexion entre un émetteur et tous les récepteurs et réalise une fonction de transmission instantanée (fils physique). Il ne transmet qu'un seul signal et le seul type de données pouvant être échangé est le type 'flottant '.

Un système Simulink peut utiliser l'environnement Matlab dans sa modélisation. Le système Simulink peut alors communiquer des données à l'espace de travail de Matlab. L'échange de données entre

Simulink et l'espace de travail Matlab peut se faire à l'aide de variables communes ou par l'intermédiaire de fichiers Matlab.

2.2.3.3 Comportement

2.2.3.3.1 Comportement global

Le comportement global d'un système Simulink est typiquement le comportement d'un système flot de données. La spécification du système définit les dépendances de données entre les blocs et ainsi un ordre d'exécution de ces derniers de façon à ce que chacun dispose des données lors de son exécution. L'exécution d'un bloc produit des données et permet à un nouveau bloc d'être exécuté.

2.2.3.3.2 Comportement local d'un bloc

Un bloc Simulink modélise le comportement dans le temps d'une partie du système. Toute évaluation d'un bloc nécessite un point d'observation dans le temps. Un pas de calcul est donc un intervalle temporel qui défini la précision de la simulation. Le pas de calcul peut être ajusté pour obtenir la précision souhaitée. Le modèle d'exécution est appelé continu car il est possible de réduire le pas de calcul autant que désiré.

2.2.3.3.3 Concepts avancés Simulink 2

- **Sous-systèmes exécutés sur condition**. Deux blocs - les blocs 'Enable' (autorisation) et 'Trigger' (déclencheur) - peuvent être utilisés pour créer un sous-système exécuté sur condition dans Simulink 2. Pour cela, il suffit de placer un de ces blocs dans le sous-système et de connecter un signal déclenchant dans le système parent. Les deux blocs déclencheur et autorisation peuvent être présents dans le même sous-système. Cela donne un sous-système qui n'est activé que lorsque le signal autorisation est 'haut' et que le signal déclenché est reçu (front montant ou descendant).
- **Systèmes avec des événements discontinus**. Dans Simulink 2, des blocs non linéaires spéciaux, qui peuvent produire une discontinuité, ont la capacité intrinsèque d'enregistrer des événements auprès du moteur de simulation. Durant la simulation une taille de pas relativement grande est utilisée. Une fois l'événement produit, le solveur revient au moment précis où l'événement s'est produit et la solution est trouvée rapidement. Les solutions sont extrêmement précises pour des simulations de systèmes présentant des discontinuités.
- **Boucles et contraintes algébriques**. Simulink 2 peut résoudre des systèmes comportant à la fois des boucles algébriques multiples et des boucles algébriques couplées hybrides. Cette fonctionnalité permet de résoudre des équations différentielles. Les types de systèmes que Simulink manipule ont été étendus par l'ajout d'un nouveau bloc de contrainte algébrique qui permet aux utilisateurs d'ajouter des contraintes algébriques à leurs équations différentielles.

2.2.3.4 Environnement

Cette partie présente l'environnement utilisé pour l'étude. Il est composé de deux parties, Matlab le simulateur et Simulink, l'environnement graphique de description.

2.2.3.4.1 Matlab

Le nom Matlab désigne un langage et un interprète de ce dernier. Le langage Matlab permet de décrire tout problème mathématique en particulier toute sorte de systèmes d'équations. L'interprète permet d'exécuter un programme écrit dans le langage Matlab. L'environnement Matlab est constitué de l'interprète du langage et de modules optionnels permettant d'utiliser Matlab dans des domaines très

pointus tels que le traitement du signal, l'analyse numérique, la modélisation de phénomènes physiques, etc. Cet environnement est un support pour tous les modules dédiés à des travaux particuliers.

2.2.3.4.2 Simulink

Simulink est un frontal Matlab permettant la description d'un système sous la forme d'un flot de données ou schéma – blocs. Cette forme d'expression permet de s'abstraire de la description impérative habituelle pour ne s'attacher qu'au comportement par l'association de blocs représentant des sous systèmes. Simulink est un environnement très adapté au traitement du signal et à la modélisation de comportements physiques. De nombreuses bibliothèques additionnelles permettent d'orienter l'outil vers un domaine plus spécifique.

De même que MATLAB et ses boites à outils 'Toolboxes', Simulink peut être complété par des bibliothèques de blocs spécialisés - les 'Blocksets', qui viennent s'ajouter à la bibliothèque de base. Un produit majeur, Stateflow vient compléter l'environnement de modélisation et de simulation Simulink. Il permet d'incorporer des contrôles complexes et des logiques de supervision dans des modèles Simulink. Il est basé sur la théorie des machines d'états finis, le formalisme Statecharts et la notation en diagrammes d'états.

2.2.4 COSSAP

COSSAP n'est pas un langage mais un environnement de spécification complet permettant de spécifier, de vérifier puis de synthétiser un système de traitement du signal. COSSAP est un outil de la suite d'outils CAO de la société SYNOPSYS.

Cet environnement permet de décrire un système sous la forme d'un flot de données [35]. Pour ce faire, des modules de traitement, appelés ensuite blocs, sont associés pour former une chaîne de traitement des données.

En premier lieu la structure utilisée dans ces descriptions ainsi que la forme de communication interblocs employée sont présentées. La seconde partie présente la forme d'exécution du schéma flot de données construit. En dernier lieu, les outils de spécification, de vérification et de synthèse de l'environnement COSSAP sont présentés.

2.2.4.1 Structure

Cette section présente la forme de description d'un système COSSAP. En premier lieu la forme de schéma flot de données est décrite, puis l'architecture interne d'un bloc est détaillée.

2.2.4.1.1 Schémas de Blocs

Un schéma flot de données COSSAP est construit par composition de blocs provenants de bibliothèques fournies ou construites par l'utilisateur. Les blocs sont connectés par des canaux et modélisent le chemin des données au travers du système. Un système est toujours fermé, aucune facilité n'est prévue pour permettre d'interagir avec l'environnement.

Un bloc COSSAP représente une fonction de traitement portant sur les données entrantes et sortantes. Les entrées et les sorties d'un bloc comportent des files d'attentes. Les blocs se présentent sous deux formes :

- Les blocs hiérarchiques sont composés d'autres blocs. Ils facilitent la tâche du concepteur en élevant le niveau d'abstraction de la vision globale du système.
- Les blocs de base englobent des entités logicielles ou matérielles décrites dans un langage extérieur.

La Figure 12 présente un schéma flot de données COSSAP. Ce détecteur de phase est constitué d'un module générateur de signaux, 'signal generator', du filtre composé des trois blocs : 'VCQ', 'phase detector' et 'loop filter', et d'un module d'arrêt de la simulation après une détection réussie.

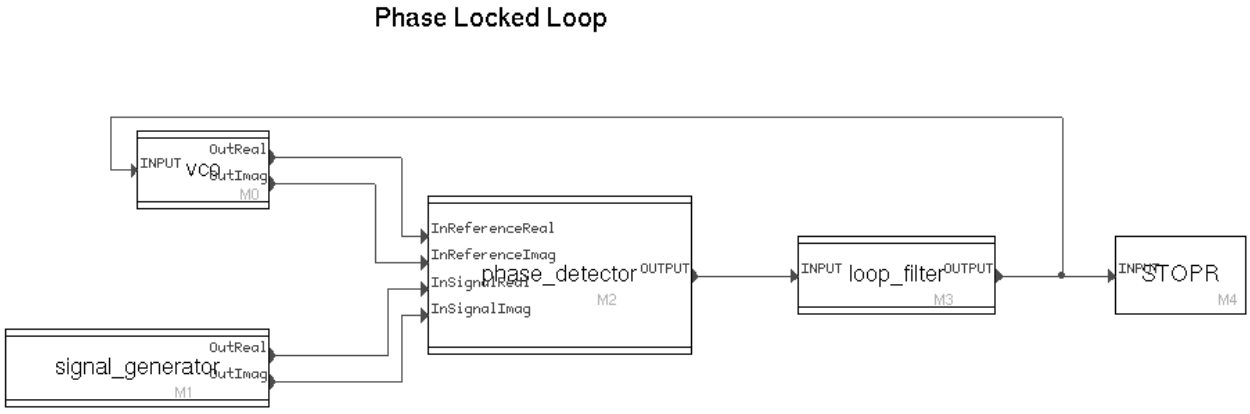

Figure 12 : Schéma flot de données

2.2.4.1.2 Blocs de base

Les blocs de base sont des blocs primitifs. Ils contiennent un module logiciel ou matériel de traitement. Deux langages permettent de spécifier un bloc primitif :

- **Le langage C** permet de décrire trois types de modules :
  - Un algorithme destiné à être implanté sur un processeur DSP.
  - Un module matériel non encore réalisé dont seul le comportement est connu.
  - Le comportement de l'environnement dans lequel se trouve le système.
- **Le langage VHDL** permet de décrire des modules matériels au niveau comportemental et au niveau RTL.

Certains blocs sont paramétrables. Le paramètre agit sur la fonction du bloc et peut être modifié durant une simulation du système.

Un même bloc peut contenir plusieurs vues de sa fonction. Il est possible d'associer dans un même bloc différentes réalisations matérielles et logicielles. Le choix du modèle utilisé lors de la simulation se fait par un paramètre accessible via l'interface de spécification, l'éditeur Bloc Diagram Editor (BDE).

2.2.4.1.2.1 Formes d'exécution

Deux types de blocs sont disponibles :

- **A flux de données statique**, le bloc a un fonctionnement synchrone. Une exécution du bloc provoque la consommation en entrée et la production en sortie d'un nombre fixe de données. Le bloc peut alors être vu comme une fonction.
- **A flux de données variable**, le bloc a un fonctionnement asynchrone. Le nombre de données consommées et produites est dépendant des données.

Un bloc spécifié dans le langage VHDL doit être un bloc à flux statique et posséder une horloge qui rythme le cheminement des données internes. Un bloc spécifié dans le langage C peut être des deux formes présentées.

2.2.4.1.2.2 Spécification en langage C

La spécification d'un bloc se compose de trois parties :

- La partie **initialisation** permet d'initialiser les états internes et d'écrire sur les ports d'entrée du bloc pour l'initialiser ou introduire un délai. Cette partie n'est exécutée qu'une seule fois.
- La partie **traitement du signal** est l'algorithme exécuté à chaque activation du bloc. A ce niveau l'algorithme peut lire et écrire des signaux et les états internes sauvegardés.
- La partie de **post-traitement** décrit le traitement lorsque la simulation est terminée. Il n'est alors plus possible de lire ou écrire sur les ports.

Un bloc possède quatre espaces mémoires :

- **La zone des paramètres** est l'espace mémoire où sont conservés tous les paramètres d'un bloc.
- **La zone des files d'attentes** est l'espace partagé par tous les blocs où est placée la file d'attente de chaque signal COSSAP. Le partage par tous les blocs d'un seul espace permet d'optimiser l'espace requit et de proposer des files d'attentes dynamiques potentiellement infinies.
- **La zone de travail** est un espace commun alloué dynamiquement à un bloc pour son exécution. Le partage permet de disposer pour chaque bloc de la totalité de la mémoire du système.
- **La zone de sauvegarde** est un espace permettant de conserver des données entre deux exécutions d'un bloc.

L'environnement COSSAP dispose d'interfaces avec les langages C et VHDL. Ces interfaces permettent l'utilisation de fonctions C ou d'entités VHDL pour décrire un bloc COSSAP. Elles se présentent sous la forme d'un ensemble de fonctions ou entités pour VHDL permettant l'accès aux entrées et sorties du bloc COSSAP. L'utilisateur conserve toutefois la charge de respecter les protocoles d'échanges de données avec les entrées et les sorties afin de ne pas perturber la simulation.

Deux méthodes sont disponibles pour décrire un bloc C : La méthode manuelle qui consiste à décrire tout l'algorithme d'accès et de traitement des données ou la méthode "automatique" qui consiste à ne décrire que le traitement des données.

Le langage « Generic C », destiné à la description d'algorithmes exécutables sur différents types de processeurs, permet de s'affranchir des algorithmes d'accès et de sauvegarde des données. Malheureusement ce langage ne permet pas tout. En particulier, seuls des blocs ayant un comportement à flux de données statique peuvent être spécifiés. Il est nécessaire de modifier le programme C généré automatiquement à partir du programme « Generic C » pour décrire des blocs plus complexes.

2.2.4.2 Communications

Les communications inter blocs sont réalisées par de simples files d'attentes First-In-First-Out et sont des connexions point-à-point. A chaque entrée d'un bloc de base est associée une file d'attente non bornée.

Le temps n'étant pas modélisé dans l'environnement COSSAP, les communications prennent un temps nul. Cependant, il est impossible de prévoir l'ordre d'arrivée de signaux transmis par des connexions différentes.

Les types de données pouvant être échangées par ces connexions sont limités aux entiers et aux flottants, augmentés de leurs versions étendues (longs).

Se situant à un haut niveau d'abstraction, les communications sont simplifiées pour fournir à l'utilisateur une vision plus claire du système. Les protocoles et types de données sont des paramètres sujets à optimisations durant la phase de synthèse du système.

2.2.4.3 Comportement

Le simulateur COSSAP est un simulateur flot de données dynamique. Les blocs, parfois appelés acteurs, sont invoqués uniquement lorsqu'ils sont prêts à être exécutés.

Cette section développe le schéma d'exécution global puis détaille le fonctionnement interne d'un bloc.

2.2.4.3.1 Comportement global

L'exécution d'un système COSSAP est guidée par le flot de données. La simulation se déroule en trois étapes répétées :

- Sélection des candidats prêts à être exécutés. Un candidat prêt est un bloc dont les contraintes de disponibilités fixées par lui-même sur chacune de ses entrées sont vérifiées.
- Exécution d'un candidat.
- Transmission des résultats produits aux blocs connectés.

L'avantage d'un tel modèle de simulation est que le flot de données du système n'est pas nécessairement statique. Le rythme des données est libre de varier durant la simulation du système.

COSSAP est essentiellement destiné au traitement du signal. Un schéma ne doit comporter que des blocs à flux statique pour que le calcul soit périodique sur des chaînes de données régulières. Cependant des blocs à flux dynamique peuvent être utilisés et briser la régularité globale. L'utilisation de tels blocs est à proscrire au maximum pour profiter des outils d'analyse et de synthèse associés à COSSAP.

2.2.4.3.2 Comportement d'un bloc

Durant la simulation, un bloc lit les données présentes à ses entrées, effectue le traitement puis écrit les résultats sur ses sorties. L'organisation habituelle d'un bloc suit les étapes :

- Retrouver les valeurs de ses paramètres.
- Déterminer le nombre d'éléments disponibles sur chaque entrée.
- Déterminer le nombre d'éléments à traiter.
- Allouer un espace de travail.
- Lire les éléments en entrée.
- Traiter les données.
- Ecrire les résultats en sortie.

Lorsqu'une multitude de données sont disponibles sur un bloc, il peut être plus efficace d'exécuter plusieurs fois le bloc avant de passer à un autre bloc. Le flot de données n'est pas toujours statique (paragraphe 2.2.1.2.1), la quantité de données disponibles en entrée d'un bloc entre deux exécutions peut varier. Un bloc évalue le nombre d'éléments disponibles avant de lire ces données et détermine ainsi le nombre d'exécutions possibles.

A la suite d'une exécution, des données en entrée peuvent ne pas avoir été consommées. Une donnée non lue en entrée n'est pas perdue, elle reste dans la file d'attente associée à l'entrée et sera disponible lors de la prochaine activation du bloc.

2.2.4.4 Environnement

COSSAP est un environnement complet de conception permettant de spécifier, de simuler, d'analyser et de mettre en œuvre des systèmes complexes de traitement du signal et de communication.

La première étape de conception est la spécification. L'outil « Block Diagram Editor » (BDE) permet de construire une spécification du système sous la forme de blocs interconnectés par des fils.

La seconde étape est la simulation et l'analyse du système. L'outil « Stream Driven Simulator » (SDS) permet de simuler le système spécifié puis d'analyser les résultats produits. La simulation, guidée par le flot de données du système, offre de bonnes performances de simulation.

La dernière étape est la réalisation des parties matérielles et logicielles du système. L'outil « HDL Code Generator » (VCG) aide l'utilisateur dans la réalisation des parties matérielles par synthèse comportementale. Des outils d'ordonnancement et de compilation permettent un développement rapide des parties logicielles.

2.2.4.4.1 Environnement de spécification

L'environnement de spécification est constitué par l'outil « Block Diagram Editor » (BDE). L'interface graphique offerte permet de construire facilement un schéma représentant le système désiré.

La spécification d'un système est composée de blocs assurant une fonction, choisis parmi ceux proposés dans les bibliothèques fournies. Les blocs sont ensuite connectés par des fils représentant les chemins de données. Les seules données pouvant être transmises sont de type numérique. Les types disponibles sont : les entiers, les entiers longs, les réels et les réels longs.

Le schéma ainsi construit représente le flot de données du système. La Figure 13 montre un exemple de spécification d'un émetteur et récepteur radio de données numériques.

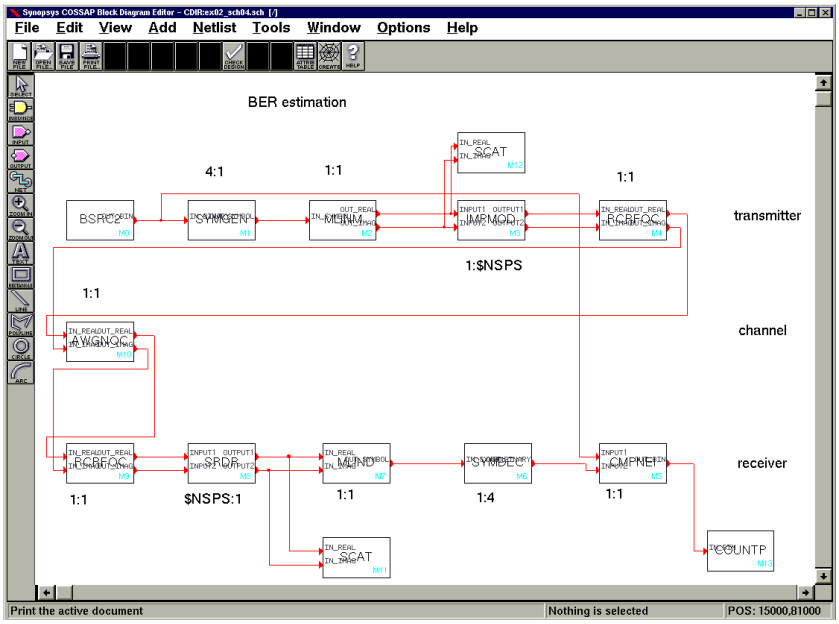

Figure 13 : L'outil de spécification DBE

2.2.4.4.2 Environnement de simulation

L'outil « Stream Driven Simulator » (SDS) est le noyau de l'environnement de simulation. Cet outil permet de simuler le système spécifié auparavant sous la forme d'un schéma de blocs.

La simulation se déroule en deux phases :

- Mise à plat du système, le simulateur décompose les blocs hiérarchiques afin de former un schéma composé uniquement de blocs primitifs. Les blocs sont alors combinés, suivant l'ordre dicté par le flot de données dans le système sans hiérarchie, afin de former un programme exécutable.
- Exécution du système, le simulateur VHDL est initialisé avec l'ensemble des blocs VHDL et la connexion avec le simulateur COSSAP est établie. Les blocs C sont exécutés sur la machine hôte du simulateur COSSAP sous la forme de programmes indépendants. La transmission des données entre les différents blocs est assurée par COSSAP qui se positionne en simulateur maître lors de l'exécution.

L'outil XSDSINT permet de piloter la simulation, l'avancement ou l'arrêt, et d'interagir par la modification de paramètres et l'observation des signaux sous différentes formes graphiques.

Enfin, les résultats sont analysés grâce à l'outil « Xchart » permettant de synthétiser les traces obtenues sous une forme graphique.

2.2.4.4.3 Environnement de synthèse

L'environnement de synthèse du système se compose principalement de deux parties : la synthèse matérielle et la synthèse logicielle. La synthèse matérielle s'articule autour des outils de synthèse comportementale Synopsys et produit une spécification VHDL synthétisable au niveau RTL. La synthèse logicielle consiste à choisir le processeur DSP puis à compiler et à ordonnancer les différents processus devant s'exécuter.

2.3 Conception de systèmes électroniques

La compréhension des concepts de base des langages de description de systèmes électroniques représente la clef de voûte de toute procédure automatique de conception. Toutes les études menées jusqu'à ce jour pour définir le langage idéal de description de systèmes, n'ont pas abouti à un résultat satisfaisant, et la cause est principalement la conjonction de plusieurs facteurs :

- Le processus de conception couvre généralement plusieurs étapes qui correspondent à plusieurs niveaux d'abstraction. Par conséquent, il est nécessaire de modéliser le système à différents niveaux d'abstraction. Il est même souvent nécessaire de composer des modules décrits à des niveaux d'abstraction différents, pour le même système. Ces niveaux d'abstraction manipulent des modèles de temps ou des concepts de communication souvent différents et difficiles à composer.
- Les systèmes peuvent être de natures très différentes (ex. discret/continu, numérique/analogique, logiciel/matériel) et comporter des composants non électroniques (ex. mécaniques, hydrauliques, optiques, etc.). Là aussi, il est souvent nécessaire de composer des modules de natures différentes dans le même système.
- Pendant le processus de conception, les langages peuvent être utilisés à plusieurs fins :
  - La spécification et la documentation des systèmes – aucun modèle exécutable n'est nécessaire.
  - La validation par simulation – un modèle exécutable est nécessaire.
  - La vérification des systèmes – un modèle formel du langage est nécessaire
  - Le raffinement et la synthèse – une sémantique de réalisation est nécessaire.

Les quatre utilisations citées ci-dessus ne sont toujours pas compatibles. Ce qui explique la définition de sous-ensembles particuliers de langages pour des utilisations spécifiques.

Tous les facteurs cités ci-dessus rendent l'analyse des langages de spécification difficile.

L'objectif de ce travail est d'étudier les concepts fondamentaux utilisés tout au long du flot de conception des systèmes électroniques afin d'analyser les points forts et les points faibles des langages existants mais aussi d'analyser les solutions proposées actuellement pour exploiter ces points forts ou pour améliorer ces points faibles.

Plusieurs études sur les langages existent dans la littérature. Gajski [34] utilise des critères plutôt syntaxiques pour comparer les langages. Les principaux critères qu'il utilise sont le style d'écriture et les constructions syntaxiques. Cette étude permet d'analyser la puissance d'expression des langages pour différents domaines d'application. Malheureusement elle ne permet pas de comprendre les niveaux d'abstraction supportés par ces langages. Ed. Lee [58] présente une étude sur les langages basée sur le concept de modèle de calcul. Il définit une notation formelle qui permet d'exprimer les concepts des différents langages, fournissant une base solide pour leur comparaison. Malheureusement, cette étude ne concerne que les niveaux d'abstraction proches de la réalisation et ne supporte pas certains concepts spécifiques à la communication de haut niveau. Hermani et Jantsh [47] présentent une étude des langages qui analyse les concepts de base au travers de différents niveaux d'abstraction. Les différents niveaux d'abstraction sont définis selon 4 axes (temps, communication, données et computation). L'application de cette étude reste très difficile à cause de la diversité des axes définissant les niveaux d'abstraction.

La principale contribution de ce travail est une étude des concepts fondamentaux utilisés lors de la conception d'un système, cela, au travers de différentes étapes du flot de conception. Les concepts considérés sont : le module, l'interface, le port, l'instance, le canal de communication et le processus (tâche réalisant du calcul et de la communication). Une étude des solutions possibles pour la spécification des systèmes, basée sur les concepts présentés et à travers les différents niveaux d'abstraction, sera présentée.

Le reste du chapitre est organisé comme suit. La section 2.3.1 introduit les concepts de base, les niveaux d'abstraction et décrit les différents concepts selon le niveaux d'abstraction. La section 2.3.2 décrit le flot de conception idéal permettant de raffiner une spécification de haut niveau en une architecture détaillée. La section 2.3.3 analyse les défaillances des langages existants pour la spécification des concepts de base à travers les niveaux d'abstraction et la section 2.3.4 analyse les solutions possibles pour définir des nouveaux langages permettant de couvrir tout le flot de conception.

2.3.1 Concepts de base et niveaux d'abstraction pour la spécification des systèmes électroniques

Ce paragraphe présente les concepts de base et les niveaux d'abstraction pour la spécification des systèmes électroniques. Cette étude est limitée aux modèles de spécification basés sur les concepts d'architecture. Sont exclus certains modèles purement fonctionnels et de très haut niveau tels que B et Z qui sont basés sur des formalismes algébriques et sont complètement indépendants de la réalisation du système. Bien que ces langages soient très performants pour effectuer les tâches d'analyse et de vérification formelle, ils ont des difficultés à représenter les concepts tels que la modularité et la communication dans les systèmes distribués. Cette étude est motivée par l'analyse des différents concepts couvrant toutes les étapes de conception d'un système.

2.3.1.1 Concepts de base pour la spécification des systèmes électroniques

Indépendamment du niveau d'abstraction, un système électronique est modélisé comme un ensemble de modules hiérarchiques représentant une architecture abstraite. Les concepts orthogonaux, la communication et le comportement sont modélisés en deux parties distinctes pour chaque module : l'interface et le contenu. Les différents modules sont interconnectés par des canaux de communication.

L'interface de chaque module contient un ensemble des ports (des points de communication d'un module avec l'extérieur) et les opérations effectuées sur ces ports (spécifiant l'interaction entre un module et le

reste du système). Ces opérations peuvent être internes (quand elles sont effectuées de l'intérieur du module contenant le port) ou externes (quand elles sont effectuées par un des modules externes).

Un module peut être hiérarchique, contenant une ou plusieurs instances de modules ou représenter une feuille dans la hiérarchie, contenant un comportement élémentaire appelé processus. Ce dernier type de module a une interface composée à son tour d'un ensemble de ports et d'un comportement. Le comportement peut être une tâche élémentaire ou un processus complexe pouvant cacher des flots d'exécution parallèle et hiérarchique. Pour simplifier l'étude nous avons omis certains concepts tels que la représentation des structures de données et de contrôle. Bien que très importants pour définir la puissance d'expression des langages, ces concepts peuvent généralement être considérés comme des facilités syntaxiques pouvant être introduites aux différents niveaux d'abstraction.

Les différents modules (processus ou modules hiérarchiques) sont interconnectés par des canaux de communication. Un canal de communication assure l'échange des données entre les modules. Transparent pour les modules qu'il interconnecte, il englobe tous les détails de communication et garantit l'échange des données. Les concepts qui définissent un canal de communication sont :

- le media qui véhicule l'information (exemple : des fils physiques ou abstraits),
- le comportement (la description des détails de communication),
- le type de données transmises (exemple : des données génériques ou données en représentation fixe).

Les concepts présentés et leurs relations structurelles sont illustrés par la Figure 14 :

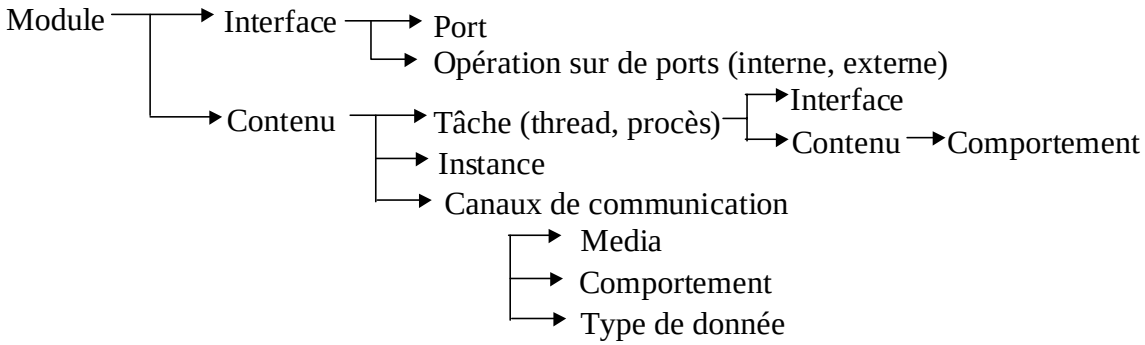

Figure 14 : Concepts de base utilisés pour la modélisation des concepts de base

Pour une meilleure illustration, les concepts de base sont présentés sur un exemple simple de système, dans la Figure 15. Le système entier est modélisé comme un module qui interagit avec l'extérieur par les ports P3 et P4. Il est composé de deux modules A et B. Le module A contient une tâche, et le module B contient deux instances de modules, B1 et B2. Les différents modules du système communiquent au travers des ports via leurs interfaces, par des canaux de communication (dans la figure, les modules A et B communiquent par le canal C1 et par l'intermédiaire des ports P1 et P2). Le contenu du module A est un processus qui suit un comportement composé des opérations de calcul et de communication.

Les concepts de la Figure 14 peuvent être décrits à plusieurs niveaux d'abstraction. Ainsi, la structure de la Figure 15 peut illustrer aussi bien une représentation physique qu'un modèle de très haut niveau.

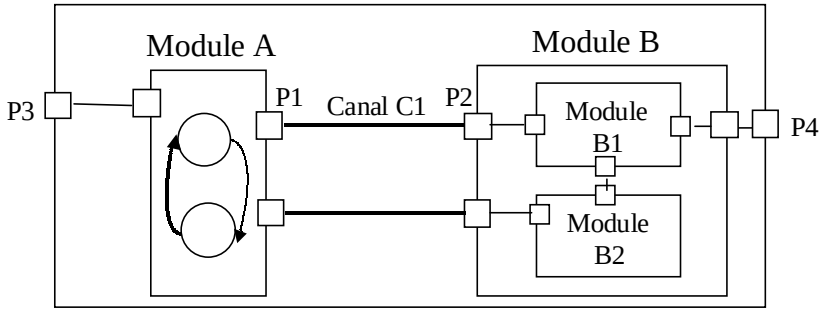

Figure 15 : Exemple de modèle de système électronique

2.3.1.2 Niveaux d'abstraction pour la spécification des systèmes électroniques

Les systèmes électroniques sont modélisés, comme présenté au paragraphe précédent, au travers de plusieurs niveaux d'abstraction. Au plus haut niveau d'abstraction, les détails non essentiels pour une analyse préliminaire du système sont ignorés. Le fossé entre les concepts utilisés pour cette spécification et l'implémentation du système, est réduit graduellement au travers des différents niveaux d'abstraction. En descendant dans l'abstraction, chaque niveau présente des aspects de plus en plus liés à la réalisation.

Généralement l'abstraction concerne le comportement, la communication et le temps. En essayant de prendre en compte tous ces concepts, les approches classiques définissent trois niveaux d'abstraction : le niveau physique, le niveau RTL et le niveau système.

Plusieurs définitions des niveaux d'abstraction sont possibles selon le concept d'étude de l'abstraction. Dans ce travail, le principal concept est la communication. Plusieurs niveaux d'abstraction peuvent découler du niveau système : le niveau pilote des entrées/soties, nommé ensuite 'driver', le niveau message et le niveau service.

Les différents niveaux d'abstraction et leurs caractéristiques principales seront présentés dans la suite.

- Le **niveau service,** la communication est représentée comme une combinaison de requêtes de services. Les différents modules communiquent par des requêtes de services, via des *réseaux abstraits* qui garantissent le routage et la synchronisation des connexions établies dynamiquement. La primitive de communication typique est une requête de service, par exemple 'imprimer (fichier)'. CORBA (Common Object Request Broker Architecture) [79] est un exemple de ce niveau d'abstraction.
- Le **niveau message**, les différents modules du système communiquent via un réseau explicite de *canaux de communication*, qui sont dits *actifs*. En plus de la synchronisation, ces canaux peuvent disposer d'un comportement complexe, par exemple la conversion des protocoles spécifiques aux différents modules communicants. Les détails de la communication sont englobés par des primitives de communication de haut niveau (par exemple send/receive) et aucune hypothèse sur la réalisation de la communication n'est déposée. Des exemples de langages modélisant les concepts spécifiques de ce niveau sont : SDL (System Description Language) [95], UML (Unified Modeling Language) [104] et ObjecTime [78]. Certaines implémentations de StateCharts [41] peuvent être placées à ce niveau.
- Le **niveau driver** est un niveau spécifique à la communication par des *fils abstraits* englobant des protocoles de niveau driver (registre, file). Un modèle de ce niveau implique le choix d'un protocole de communication et la topologie du réseau de connexions. Un exemple de primitive de communication spécifique est l'écriture d'une valeur ou l'attente d'un événement sur un port. Les langages employés à ce niveau d'abstraction sont : CSP [45], SystemC 1.1 [101], Cossap [99] et StateCharts [41].

- Le **niveau RTL,** la communication est réalisée par des *fils ou bus physiques*. L'unité de temps devient le cycle d'horloge, les primitives de communication sont du type set / reset sur des ports activés lors d'un nouveau cycle d'horloge. Les langages les plus utilisés pour la modélisation de systèmes à ce niveau sont SystemC 0.9-1.0, Verilog [110] et VHDL [112].

Le Tableau 1 résume les principales caractéristiques des niveaux d'abstraction utilisés : la communication, les primitives de communication, et les modèles typiques.

	Communication	**Primitive de Communication typique**	**Models typiques**
Niveau service	Réseaux abstraits	Demande (Imprimer, dispositif, fichier)	CORBA
Niveau message	Canaux actifs	Send (fichier, disque)	SDL, UML
Niveau driver	Fils Abstraites (canaux abstraits)	Write(donnée, port) Wait until x=y	Cossap, StateCharts, CSP SystemC 1.1
RTL	Fils Physiques	Set (Valeur, port) Wait (clock)	VHDL, SystemC 0.9-1.0, Verilog

Tableau 1 : Niveaux d'abstraction

2.3.1.3 Particularisation des concepts au travers des niveaux d'abstraction

Ce paragraphe présente les concepts de base vus au travers des niveaux d'abstraction (section 2.3.1.2). Les systèmes peuvent être représentés comme un ensemble des modules hiérarchiques interconnectés, indépendamment du niveau d'abstraction. Cependant les concepts de base (module, interface, contenu et canal de communication) vont avoir des significations différentes selon le niveau d'abstraction.

Au niveau service, les interfaces des modules sont composées de *ports d'accès à des réseaux abstraits* présentés dans la section précédente. Ces ports fournissent des services d'un certain type et les opérations sur les ports sont des *requêtes de services*. Les tâches élémentaires sont des processus qui interagissent avec l'environnement via des requêtes des services. Le temps de communication est non nul et peut ne pas être prévisible. Le temps global est abstrait et sous la forme d'un ordre partiel entre les requêtes de services. A ce niveau les processus peuvent contenir des files d'exécution (threads) hiérarchiques similaires à ce qui est présenté dans StateCharts [41].

Au niveau message, les interfaces des modules sont composées des *ports d'accès aux canaux actifs*. Cette fois-ci, les ports d'accès fournissent des *appels de procédures* de haut niveau (exemple send, receive, put ou get). Par contre, chaque canal peut cacher un comportement complexe. Le comportement des tâches élémentaires sont des processus qui communiquent avec l'extérieur par des envois ou des réceptions de messages. Le temps de communication est aussi non nul et n'est pas toujours prévisible. Le temps est abstrait et se présente sous la forme d'un ordre partiel des envois de messages. A ce niveau les processus peuvent contenir des files d'exécution hiérarchiques.

Au niveau driver, les ports composant les interfaces sont des *ports logiques*, qui assurent l'interconnexion des différents modules par des fils abstraits. Les opérations effectuées sur ces ports sont des *assignations de données de type fixé* (ex. entier, réel). Ces opérations peuvent cacher le décodage des adresses et la gestion des interruptions. Le temps de communication est non nul et peut ne pas être prévisible. Le comportement des modules de base est décrit par des processus qui réalisent un pas de calcul et des opérations de communication. La notion d'ordre global entre les événements du système apparaît avec une horloge globale. A ce niveau les processus correspondent à des machines d'états finis étendues (EFSM) où

chaque transition peut cacher un calcul complexe pouvant durer plusieurs cycles d'horloge après la réalisation du système.

Au niveau RTL, les interfaces des modules sont composées de *ports physiques*, qui permettent de connecter les différents modules par des fils physiques. Au travers de ces ports s'effectuent des opérations du type « *set / reset* » sur des signaux. La communication étant réalisée par fils physiques, les données sont transmises instantanément en représentation binaire. A ce niveau la gestion des interruptions et le décodage des adresses sont explicites. L'unité temporelle devient le cycle d'horloge et les processus correspondent à des machines d'états finis où chaque transition correspond à un cycle d'horloge.

Le Tableau 2 résume la modélisation des concepts de base, au travers des différents niveaux d'abstraction : les types de modules, les interfaces (par leurs ports et les opérations afférentes) et le contenu d'un module à chaque niveau (le type de processus, et les détails de communication spécifiques).

Concept / Niveaux d'abstraction			**Niveau Service**	**Niveau message**	**Niveau driver**	**Niveau RTL**
Module	**Interface**	**Port**	Service typé	Accès au canal actif	Port logique	Port physique
		Opération	Demande d'un service	Send/Receive vers un processus identifié	Assignation de donnée sur un port	Set/reset bits
	Contenu	**Processus/tâche**	Threads hiérarchiques a la StateCharts	Threads hiérarchiques a la StateCharts	EFSM	Calcul au niveau cycle d'horloge
		Instance	Instance d'un module	Instance d'un module	Instance d'un module	Instance d'un module
		Canal de communication **- media** **- comportement** **- donnée**	- réseau abstrait - routage - demandes de services	- canal actif - conversion de protocole - transmission de données génériques	- fils abstraits - protocoles niveau driver - type fixé de données	- fils physiques - transmission - données en représentation fixe

Tableau 2 : Concepts de base à travers des niveaux d'abstraction

Ce tableau fait apparaître la difficulté de décrire tous les concepts au travers de tous les niveaux d'abstraction à l'aide d'un seul langage, la section 2.3.3 présente et apporte des réponses partielles.

2.3.2 Flot de conception pour les systèmes électroniques

Le flot de conception est composé de plusieurs niveaux d'abstraction. Chaque étape consiste à raffiner un modèle en un modèle enrichi en précisions. La Figure 16 montre un flot de conception générique permettant de passer du niveau service au niveau RTL. Chaque étape de ce flot comporte des tâches de validation/vérification et raffinement.

Le système est initialement spécifié au niveau service comme un ensemble de modules hiérarchiques et qui communiquent par des réseaux abstraits. Après la validation de cette spécification de haut niveau, la première étape du flot de conception, consiste à générer les canaux actifs explicitant le réseau de communication abstrait, à fixer les modules de routage des données et à adapter les interfaces des modules. La seconde étape consiste à résoudre les protocoles des canaux actifs. Une étape de synthèse de protocole peut être nécessaire [22]. Cette étape introduit des nouveaux modules qui réalisent les communications (contrôleurs de communication). Le modèle obtenu est une architecture abstraite au niveau driver, où chaque module représente un processeur de l'architecture finale. Un tel processeur peut

être réalisé par un processeur logiciel (ex. DSP ou micro-contrôleur), un processeur matériel (ex. coprocesseur matériel) ou un module déjà existant (IP, contrôleur de communication, environnement). Les modules de cette architecture sont interconnectés via des canaux qui englobent des protocoles fixes. L'étape suivante du flot de conception est la validation de l'architecture abstraite et son raffinement en une architecture détaillée au niveau RTL. Les modules logiciels sont transposés sur des processeurs spécifiques. Cela nécessite une synthèse des interfaces matérielles qui permettent d'adapter les communications du processeur avec le reste de l'architecture. Si la partie logicielle contient des tâches parallèles, la synthèse d'un OS (système d'exploitation) est aussi nécessaire. Les modules matériels sont raffinés au niveau du cycle d'horloge. Les blocs existants sont englobés dans l'architecture finale, par une interface d'adaptation de protocole.

A tous les niveaux d'abstraction, les modules sont modélisés au travers de concepts spécifiques, comme présenté à la section 2.3.1.3.

La Figure 16 illustre le flot de conception présenté sur un exemple de système électronique. Ce flot idéal considère que toutes les parties du système sont décrits au même niveau d 'abstraction. En réalité il est souvent nécessaire de traiter des spécifications où les différentes parties sont décrites à des niveaux d'abstraction différents. Dans ce cas, la conjonction de la communication et du comportement et la distinction des opérations sur les ports, en opération internes et externes, joue un rôle très important pour la spécification. Un module raffiné est représenté à un niveau d'abstraction inférieur sans modifications pour le reste du système. Dans ce cas seules les opérations internes sont raffinées. Ainsi le nouveau comportement du module accède à l'interface via les opérations internes raffinées tandis que le reste du système accède à cette interface via les opérations externes inchangées. De la même manière, la communication peut être raffinée sans impliquer de modifications aux modules connectés. Cette étape raffine aussi les interfaces de ces modules. Bien entendu dans ce cas seules les opérations externes sont raffinées. Ce schéma permet de supporter les systèmes composés de modules de nature très hétérogène (ex. continu/discret, données/contrôle) et/ou contenant des parties non électroniques (ex. mécanique, hydraulique).

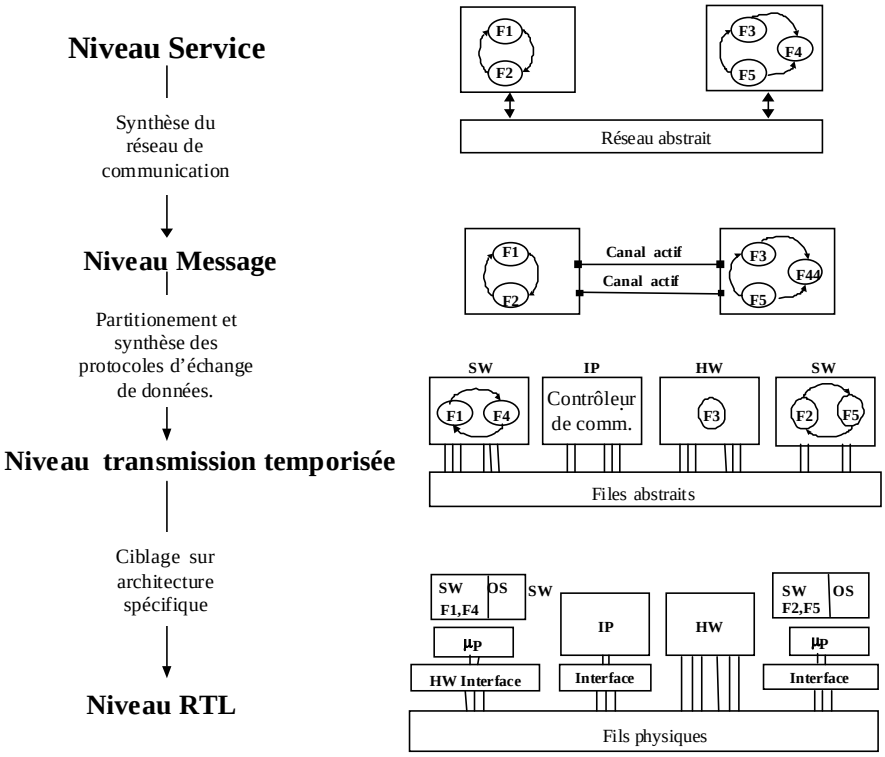

Figure 16 : Flot de conception sur exemple de système

2.3.3 Les langages de spécification des systèmes électroniques

Le langage de description idéal doit couvrir tout le flot de conception, tout en restant exécutable à tous les niveaux d'abstraction. Malheureusement, ce langage n'existe pas. Cette section analyse les limites des langages existant. Différentes solutions sont discutées dans la section 2.3.4.

2.3.3.1 Méthodes actuelles de spécification système

Pour la spécification des systèmes complexes, plusieurs méthodes de spécification ont été développées. Ces méthodes sont différentes par la granularité ou l'unité de parallélisme (processus synchrones, objets, etc.) ou par le mécanisme de communication (échange de messages, RPC, structures de données distribuées ou variables partagées). Les principales méthodes de spécification sont :

- **ADL (Architecture Description Languages).** Ces langages permettent une définition formelle de l'architecture de haut niveau d'un système complexe. Pour la spécification d'un système, les ADLs utilisent trois concepts de base [71] : le module, le connecteur et la configuration (une combinaison particulière des connecteurs et modules). Ces concepts sont interprétés différemment par les ADLs, le même système peut être décrit différemment en fonction du langage utilisé. Rapide [62], [63], Wright [1] et UniCon [97] sont des exemples d'ADL.

- **Méthodes orientées objet pour l'analyse et le design des systèmes (OOA/OOD).** Ces méthodes permettent une description des systèmes complexes à un très haut niveau d'abstraction. La principale idée est la décomposition des systèmes complexes dans des sous-systèmes plus faciles à maîtriser. Elles utilisent les avantages de l'approche objet pour la description, la visualisation et la documentation des systèmes. La méthode de ce type la plus populaire, est actuellement UML (Unified Modeling Language). Cette méthode représente l'unification des meilleurs concepts présentés dans les trois plus importantes méthodes OOA/OOD : Booch [11], OMT [65] et OOSE [46]. Le principal inconvénient de cette méthode est l'absence de modèle exécutable pour la synthèse et la vérification. StateCharts [41], ObjectTime [78] et SDL [104] sont présentés comme des modèles exécutables d'UML. Il s'agit de langages existants auxquels sont associées des méthodes de décomposition de spécification conformément à UML.

- Plusieurs langages ont été développés pour la **modélisation des systèmes divers,** à un niveau d'abstraction élevé. Ces langages fournissent de larges bibliothèques pour la description de systèmes appartenant aux domaines d'applications tels que la mécanique, l'hydraulique ou encore les applications à base de DSP. Les langages de ce type les plus connus sont Matlab [68] et Matrixx [69]. Certains langages sont dédiés à la modélisation des systèmes à base de DSP, les plus utilisés dans le monde de la conception des systèmes électroniques sont COSSAP et SPW [54].

- **Méthodes pour la modélisation des systèmes temps – réel**. Les systèmes temps- réel sont des systèmes dont le comportement doit respecter des contraintes temporelles : un temps maximum est imposé entre la variation des entrées et la réaction correspondante. L'importance de ce type de systèmes a entraîné le développement de langages synchrones tels que Lustre [40], Esterel [8], [9] et des langages asynchrones tels que SDL et ObjecTime [78].

- **HDL (Hardware Description Languages).** Les langages de description matériel sont des langages qui supportent les concepts spécifiques aux systèmes matériels tel que le concept de temps, de parallélisme, de communication interprocessus (signaux et protocoles), la réactivité et les types de données spécifiques (entiers signés et non signés, données en virgule fixe et structures des vecteurs de bits). Les deux langages de description matériel les plus utilisés sont VHDL et Verilog.

- **Langages de programmation**. Certaines approches ont étendu les langages utilisés en informatique pour la programmation (C, C++), par l'ajout des concepts présentés ci-dessus, spécifiques au matériel. Le choix de ces langages repose sur trois raisons principales : ils fournissent le contrôle et les types de

données nécessaires, la plupart des systèmes contiennent des parties matérielles et logicielles et pour la partie logicielle un de ces langages représente le choix naturel. De plus, les concepteurs sont familiers avec ces langages et les outils attenants. Les langages développés consistent en la définition des fonctions ou classes pour modéliser la concurrence, le temps et le comportement réactif. Les langages de ce type les plus utilisés sont N2C [111], SystemC [101], SpecC [33], [34], JavaTime [115] et OpenJ [118].

2.3.3.2 Capacités des langages pour la modélisation des concepts de base

Cette section analyse les langages de spécification selon leur capacité à modéliser les concepts de base à travers les niveaux d'abstraction présentés au paragraphe 2.3.1. En conformité avec ces critères, les langages sont regroupés différemment de leur présentation générale.

Les langages **SystemC 0.9 et HDL** modélisent un système comme un ensemble d'entités physiques interfacées par des ports physiques. Les opérations sur les ports sont du type *set* / *reset bits* ou assignations de données. La communication est représentée par des canaux physiques permettant le transfert des données dans une représentation fixe.

Ces langages sont très performants pour la modélisation des concepts matériels, mais il manque plusieurs concepts nécessaires pour notre flot de conception. Ainsi, le concept de port abstrait sur lequel peuvent s'effectuer des opérations indépendantes de protocoles, ne peut pas être modélisés. Du point de vue de la communication, ces langages ne modélisent pas le concept de canal abstrait, transférant des données de type générique.

Les méthodes de spécification **Coware, SC 1.0, Cossap et SPW** fournissent en plus de concepts des langages SystemC 0.9 et HDL, des nouveaux concepts **:** les systèmes sont découpés en modules abstraits qui ont des interfaces composées par des accès à des canaux de transmission. Les opérations effectuées par les accès sont des RPC ou du 'token flow'(pilot de jeton). Les canaux transmettent ou temporisent des données de type fixé. Certains concepts restent manquants : les ports acceptant des types génériques de données et les opérations afférentes à ces ports et la modélisation d'un canal de communication capable de réaliser plus qu'une simple transmission, par exemple une conversion de protocole.

Les langages synchrones modélisant les systèmes temps réel, représentent les systèmes comme un ensemble de modules abstraits, qui peuvent être interfacés par des ports logiques. Les modules sont interconnectés par des canaux de communication qui assurent le transfert d'un type fixé de données. La communication est réalisée par des assignations de données sur les ports. Ces opérations doivent respecter des contraintes de temps imposées.

L'inconvénient de ces langages est qu'ils supposent que la communication ne prend pas de temps. Ils ne peuvent pas modéliser une communication hiérarchique et distribuée et ne permettent pas l'utilisation des ports abstraits.

Les langages asynchrones modélisant les systèmes temps réel, apportent aussi leurs nouveaux concepts. Les modules abstraits composant le système communiquent au travers d'interfaces composées d'accès. Ces interfaces sont accessibles par des primitives de communication de haut niveau, comme par exemple *send, receive, put* ou *get*. L'interconnexion entre les sous-systèmes est réalisée par des canaux actifs, permettant une éventuelle conversion de protocole, et l'échange de données génériques.

Ces langages ne peuvent pas modéliser les concepts de bas niveau spécifiques au niveau RTL.

Le **langage UML et les ADLs** fournissent des concepts plus abstraits pour la modélisation d'un système. Ainsi, les interfaces des modules peuvent être des ports d'accès fournissant des services, accédés par des demandes de services. L'interconnexion entre les différents modules est réalisée par des réseaux abstraits qui effectuent le routage des demandes de services. Ces méthodes présentent le même désavantage que les

langages asynchrones pour la modélisation des systèmes temps réel : l'impossibilité de modéliser les systèmes au niveau RTL.

Les concepts manquants des méthodes de spécification considérées, sont présentés au Tableau 3.

Concept / Langage				**SystemC.9 HDL**	**Coware, SC1.0, Cossap, SPW**	**Temps réel Langages synchrones**	**Temps réel Langages asynchrones UML, ADLs**
Module	**Interface**	**Port**		Ports abstraits	Ports abstraits	Ports actifs	Ports physiques
		Opération		Protocoles indépendants des opérations	Opérations sur des données de type générique	Opérations sans retard et réponse fixe	Opérations sur les données en représentation régulière protocoles détaillés
	Contenu	**Processus/tâche**		Multitâche synchronisé	Exécution dépendante des données	Indépendance d'un cycle fixe d'exécution	Modèles au niveau cycle- près opérations spécifiques à une implémentation
		Canal de communication	**Media**	Canaux abstraits	Canaux actifs	Communication hiérarchique et distribuée	Signaux physiques
			Comportement	Comportement non transmission	Conversion des protocoles	Autres que 'broadcasting'	Flot de données uniforme fixé
			Donnée	Type générique de données	Type générique de données	Type générique de données	Représentation fixée d'une donnée

Tableau 3 : Concepts manquants des langages de spécification

Nous pouvons remarquer que chacune de ces méthodes présentent des points forts, mais aussi des points faibles pour la modélisation de certains concepts. Aucun langage ne présente les capacités nécessaires à la modélisation des systèmes électroniques à tous les niveaux d'abstraction. Ils offrent des solutions partielles pour la spécification des systèmes électroniques actuels. C'est pourquoi l'enjeu des industriels et de la recherche est depuis quelques années de proposer de nouvelles solutions pour une spécification complète des systèmes électroniques.

2.3.4 Solutions pour la spécification des systèmes électroniques

Afin de spécifier des systèmes électroniques, plusieurs solutions sont proposées, elles sont appliquées selon l'une des approches :

- Approche homogène : un seul langage est utilisé pour la modélisation d'un système électronique.
- Approche hétérogène : des langages spécifiques sont utilisés pour la modélisation des différents modules d'un système électronique.

2.3.4.1 Solutions pour la spécification homogène

La spécification homogène implique l'utilisation d'un seul langage pour la spécification d'un système électronique. Cette approche présente l'avantage de réaliser une combinaison plus cohérente des différents concepts, mais la définition d'un langage qui supporte les concepts nécessaires pour la spécification d'un système électronique est très difficile. Deux solutions peuvent être envisagées :

La définition d'un nouveau langage, il peut suivre une nouvelle syntaxe, comme le langage Rosetta [96] ou être l'extension d'une syntaxe existante. Cette dernière alternative a été adoptée pour la définition du langage SpecC [34].

L'extension exécutable d'un langage existant. Cette solution implique le développement de bibliothèques ou classes nécessaires au langage. Des exemples de langages obtenus par extension du langage C++ sont SystemC, C – level et IP – modeling.

2.3.4.2 Solution pour la spécification hétérogène des systèmes électroniques

Une spécification hétérogène implique une modélisation de chaque module du système dans un langage spécifique et approprié. Cela permet d'exploiter au mieux les performances des langages existants.

Le problème qui se pose est la spécification des interfaces et des interconnexions entre les différents modules du système. Cela implique un modèle de coordination qui peut être un langage de coordination ou une forme intermédiaire.

2.3.4.3 Analyse des solutions proposées pour la spécification des systèmes électroniques

Ce paragraphe présente une étude des solutions proposées pour la spécification de systèmes électroniques selon les exigences du flot de conception présenté au paragraphe 2.3.2 : la spécification des systèmes à tous les niveaux d'abstraction, validations et raffinements.

Les critères d'analyse sont :

- La puissance d'expression – ce critère fixe la difficulté ou la facilité à spécifier un système. Les principales composantes de la puissance d'expression d'un langage sont le modèle d'exécution et de communication.
- La puissance d'analyse – ce critère est lié à l'analyse formelle d'un langage et les possibilités de construire de nouveaux outils.
- La puissance commerciale – ce critère inclut différents aspects, les principaux sont la standardisation, la courbe d'apprentissage, les outils et les bibliothèques disponibles.

Le Tableau 4 présente une analyse des solutions pour la spécification des systèmes électroniques en fonction de ces critères.

Un nouveau langage ou une extension syntaxique d'un langage existant peut être envisagé pour faciliter l'analyse formelle et la construction de nouveaux outils. Cette solution est intéressante du point vue de la puissance d'analyse. Un tel choix entraîne généralement une puissance d'expression réduite pour la partie de calcul ainsi que pour la partie communication. En analysant ce langage du point de vue puissance commerciale, plusieurs problèmes sont posés : la standardisation est difficile à réaliser, la courbe d'apprentissage est longue et évidemment un nouveau langage ne dispose pas d'outils et de bibliothèques.

Une extension exécutable d'un langage est intéressante du point de vue de la puissance d'expression illimitée, du modèle d'exécution déjà existant, de la possibilité d'être simulé sans effort supplémentaire et de la standardisation. Un tel langage est très difficile à synthétiser et analyser, et nécessite une longue courbe d'apprentissage de sa sémantique.

L'approche multilangage, utilisation d'un ensemble de langages existants et d'un langage de coordination, présente une puissance d'expression illimitée. Du point de vue puissance de commercialisation, les problèmes qui se posent sont la nécessité d'un outil de synthèse pour la communication et l'apprentissage du langage de commercialisation. L'analyse formelle est difficile à réaliser.

Critère d'analyse	Solution de spécification	Approche homogène		Approche hétérogène	
		Nouveau Langage	**Extension exécutable D'un langage existant**	**Langage de coordination + ensemble de langages existants**	**Forme intermédiaire + ensemble de langages existants**
Puissance d'expression	**Modèle d'exécution**	Restrictif	Illimité	Illimité	Illimitée
	Communication	Restrictif	Illimité	Illimité	Illimitée
Puissance d'analyse	**Analyse Formelle**	Accessible	Non pratique	Difficile	Accessible
	Construction de nouveaux outils	Facile	Non pratique	- Facile pour la communication - Difficile pour le modèle d'exécution	Facile
Puissance commerciale	**Standardisation**	Difficile	Implicite	Solutions pratiques	Non requise
	Courbe d'apprentissage	Longue	- Implicite pour la syntaxe - Longue pour la sémantique	Longue pour le langage de coordination	Inexistante
	Outils existants et bibliothèques	—	- Oui pour l'exécution - Non pour la synthèse/analyse	- Oui pour l'exécution partielle - Non pour la communication	Non requit

Tableau 4 : Analyse de solutions pour la spécification de systèmes électroniques

La solution basée sur une forme intermédiaire ne présente aucune limitation du point de vue puissance d'expression. Cette solution est aussi préférable pour le raffinement, l'analyse formelle et la construction de nouveaux outils. Le format intermédiaire étant un format interne, transparent pour l'utilisateur, la mesure de la puissance commerciale est sans objet.

2.4 Conclusion

La première partie de ce chapitre à présenté une étude sur trois langages représentatifs des langages de spécification de systèmes électroniques. Les concepts sélectionnés pour l'étude sont les principaux concepts apparaissants dans la littérature, la concurrence, la hiérarchie, la communication et la synchronisation. Le Tableau 5 résume les particularités de chaque langage étudié.

	Modèle d'exécution	**Signaux**	**Concurrence**	**Hiérarchie**	**Communication**
COSSAP (traitement du signal)	Flot de données dynamique	Discrets, pas de temps	Issue de la non dépendance des données	Oui, modules de base C, Fortran ou VHDL	Une file d'attente par signal
SDL (modélisation système)	Machines d'états	Discrets, pas de temps	Processus concurrents	Oui	Une file d'attente en entrée de processus
Matlab / Simulink (modélisation physique)	Flot de données régulier (synchrone)	Continus, pas de simulation réglable	Issue de la non dépendance des données	Oui	Transmission à chaque fin de pas de simulation

Tableau 5 : Résumé des concepts par langage

L’étude des langages et environnements de cosimulation au travers des mêmes concepts permet de constater que leurs différences sont directement liées au domaine d’utilisation. Cependant la structure globale et ses connexions, les communications, sont tout à fait semblables d’un langage à l’autre.

La seconde partie du chapitre à présenté une approche globale, synthétique et pragmatique de la conception de systèmes électroniques. Les concepts structurels de base et leurs variations au cours de la conception au travers de quatre niveaux d’abstraction ont été présentés. Cette approche permet de proposer un flot de conception focalisé sur la notion d’assemblage de modules. Une étude des solutions actuelles montre qu’aucune n’apporte de réponse complète. Plusieurs voies sont explorées à la lumière de ces concepts et aboutissent à la conclusion qu’une forme intermédiaire liant les langages existant semble être une solution.

Ce travail propose une approche de la conception basée sur l’emploi des langages adaptés aux domaines d’applications et sur une forme intermédiaire explicitant la coordination entre les modules du système. L’avantage de la forme intermédiaire est de faire évoluer sa sémantique au fur et à mesure du raffinement du système.

Les perspectives à court terme sont le développement d’une forme intermédiaire adaptée à la conception des systèmes électroniques et validée au travers des concepts présentés. A moyen terme, les perspectives sont le développement d’environnements de simulation et de raffinement autour de la forme intermédiaire sélectionnée.

Chapitre 3

SYNTHESE DE LA COMMUNICATION DANS LE CADRE DES SYSTEMES HETEROGENES MULTILANGAGES

Ce chapitre présente une nouvelle méthodologie pour la synthèse de la communication dans le cadre des systèmes hétérogènes multilangages. Cette approche permet de raffiner les interconnexions entre les blocs décrits dans des langages différents et tient compte des différents formalismes de description, modèles de calcul et schémas de communication.

Ce chapitre correspond à un travail réalisé en commun par F. HESSEL et P. COSTE et fait partie des thèses des deux auteurs.

3.1 Introduction

La conception des systèmes actuels, d'une complexité sans cesse croissante, nécessite une combinaison de plusieurs langages de spécification adaptés aux différentes parties du système afin de couvrir toute la fonctionnalité d'un système embarqué. En effet, les équipes de développement disposent de flots de conception propres et de langages de spécification différents ce qui rend difficiles les tâches de validation et de synthèse du système global.

Les méthodes actuelles proposent aux différentes équipes de développer leurs modules suivant un cahier des charges défini lors de la spécification. Cependant, la validation du système complet et l'interconnexion des différents modules ne s'effectuent que lors de la construction du prototype.

Une méthode de conception multilangage permet la spécification et la validation d'un système en utilisant différents formalismes de description, modèles de calcul et schémas de communication. Cette méthode doit permettre de réduire considérablement le temps de conception en validant très tôt la spécification du système et doit également permettre de résoudre les problèmes d'interconnexions entre les blocs décrits dans des langages différents.

La principale contribution présentée dans ce chapitre est une nouvelle approche pour la synthèse de la communication dans le cadre des systèmes hétérogènes multilangages. Cette approche introduit un nouveau concept, définit un nouveau modèle de communication et permet le raffinement des interconnexions entre les blocs décrits dans des langages différents. La validation du système à ses divers stades de développement peut être réalisée en utilisant l'approche de cosimulation [43], [67].

3.1.1 Spécification des systèmes hétérogènes : définition du problème

La synthèse des communications devient essentielle pour la conception des systèmes multilangages puisque les différentes parties du système communiquent entre elles. Chaque partie du système peut utiliser différents schémas de communication, protocoles de communication et topologies d'interconnexion selon le langage de spécification utilisé et les besoins du système. La topologie d'interconnexion et le protocole de communication ont une grande influence sur les performances d'un système et peuvent mener à des solutions infaisables si le concepteur sous-estime le débit des communications. Les communications entre les différentes parties du système sont représentées par un modèle de haut niveau. Ce modèle est l'équivalent d'une « netlist » de haut niveau. Il spécifie les interconnexions entre les différentes parties du système au travers de canaux de communication abstraits. Dans le cas de spécifications multilangages, le problème qui se pose est celui de l'interface entre des ports de communications qui peuvent être extrêmement différents d'un langage à l'autre.

Le problème de la synthèse de communication se résume à produire une interface adaptée à chaque partie du système, selon leurs caractéristiques, de façon à obtenir un ensemble de modules communiquant par des bus, des signaux et éventuellement un composant de contrôle. La synthèse de la communication inter modules s'effectue en parallèle avec le raffinement des modules comme présenté en Figure 17.

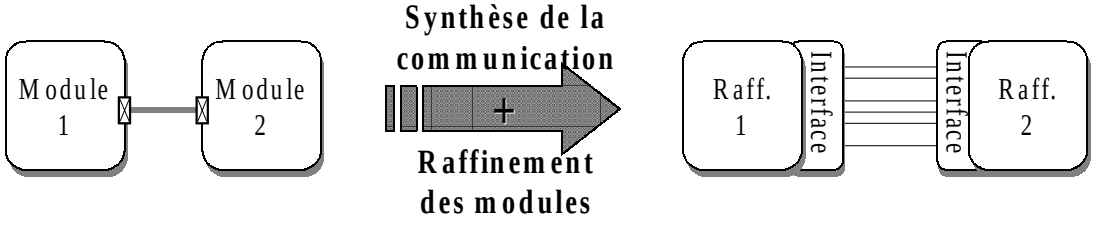

Figure 17. Flot de conception multilangage

3.1.1.1 Modélisation de la communication dans le cadre des systèmes hétérogènes

Le modèle de communication utilisé pour la modélisation de la communication des systèmes hétérogènes multilangages est une extension du modèle de communication présenté par [21].

3.1.1.1.1 Le canal de communication

Le canal de communication est une unité responsable des communications entre les sous-systèmes. Un canal offre un ensemble de primitives de communication (services) qui sont appelées par les sous-systèmes pour communiquer. Le modèle d'unité de canal combine le concept de moniteurs et de passage de messages, connu aussi comme l'appel de procédure à distance (RPC [2]). L'utilisation de RPC pour invoquer les services de canaux permet une représentation flexible, avec une sémantique claire et efficace, modélisant des schémas de communication existants. Ces schémas de communication peuvent être décrits séparément du reste du système, permettant ainsi de séparer les aspects de communication et de comportement d'une spécification.

Le modèle du canal est composé d'un ensemble de primitives de communication, d'une interface et d'un éventuel contrôleur. Le contrôleur assure la résolution des conflits d'accès au canal et l'adaptation des différents schémas de communication utilisés pour les sous-systèmes. Un sous-système qui désire communiquer au travers d'un canal fait appel à une primitive de communication du canal ou fait une lecture/écriture sur les portes d'interface dans le cas des sous-systèmes existants.

3.1.1.1.2 Le protocole de communication

Le protocole de communication offre une réalisation possible pour un ensemble de primitives de communication. La définition des protocoles de communication est réalisée dans une bibliothèque de communication et peut contenir plusieurs réalisations différentes du même protocole. Un protocole de communication contient aussi une interface sur laquelle les primitives de communication lisent et écrivent. Cette interface représente les portes et les interconnexions nécessaires aux différents sous-systèmes utilisant ce protocole.

Le processus de synthèse choisit dans la bibliothèque le protocole de communication qui est capable d'exécuter les primitives de communication requises par le canal. Le protocole de communication est distribué entre les primitives de communication et le contrôleur. Dans le cas d'interconnexions de modules existants, le contrôleur sera responsable de l'adaptation des protocoles de communication utilisés par les modules interconnectés par le même canal. Lorsqu'il n'y a pas de contrôleur, le protocole est complètement distribué entre les primitives de communications, cela peut être le cas pour le protocole «*handshake*». La complexité du contrôleur peut varier d'une simple file d'attente à un protocole complexe en couches ou il peut être un module existant réutilisé.

3.1.1.1.3 Le type d'interface

Le type d'interface de communication de chaque sous-système dépend du niveau d'abstraction de la description du sous-système. La Figure 18 présente trois niveaux d'abstraction.

Au niveau *message*, l'interface du sous-système est composée d'un ensemble des portes abstraites de communications (par la suite appelées accès) [50], accessibles au travers de primitives de communication de haut niveau, comme par exemple *send*, *receive*, *put* ou *get*. Les détails de la communication sont englobés par les primitives de communication et aucune hypothèse sur la réalisation de la communication n'est faite. La spécification du sous-système est réalisée sans prendre en considération le protocole de communication qui sera utilisé. L'interconnexion entre les sous-systèmes est réalisée par des canaux de communication actifs homogènes.

Au niveau *abstraction intermédiaire*, appelé « *driver* », l'interface peut être composée d'un ensemble de portes logiques, appelées « *connecteurs* » (voir section 3.3.1.2). La communication est exécutée au travers des opérations de lecture et d'écriture sur des registres d'entrée et de sortie. Le décodage d'adresses physiques et la gestion d'interruption sont cachés par les opérations. L'interconnexion entre les sous-systèmes est réalisée par des canaux abstraits.

	Net	Interface	Primitives de communication
Niveau message	Canaux actifs	Portes abstraites de communication (accès)	Primitives de communication de haut niveau (put, get, etc.)
Niveau driver	canaux abstraits	Portes logiques	Opération de lecture et d'écriture
Niveau RTL (φ)	Bus physique	Portes physiques	Opération de set et reset (bits)

Figure 18. Niveaux d'abstraction des communications inter modules

Au niveau *RTL*, l'interface est composée d'un ensemble de portes physiques. La communication est réalisée au travers de fils (par exemple, des signaux VHDL) et tous les détails de la mise en œuvre du protocole de communication sont décrits. Au niveau driver et au niveau RTL, le protocole de communication est défini et ne peut plus être modifié. L'interconnexion entre les sous-systèmes est réalisée par des bus physiques.

Une spécification multilangage peut combiner ces trois types d'interfaces. La spécification peut être composée, par exemple, d'un sous-système décrit au niveau message et de deux composants existants (un au niveau intermédiaire et l'autre au niveau RTL). Dans ce cas, un canal de communication peut connecter des primitives de communication avec des interfaces au niveau drivers et des interfaces au niveau RTL. Le canal est ici appelé canal de communication abstrait hétérogène (voir section 3.3.1.3). La synthèse de la communication multilangage génère les interfaces au niveau driver pour les sous-systèmes décrits au niveau RTL, adapte les interfaces au réseau de communication choisi, ajoute l'éventuel contrôleur et génère les interconnexions correspondantes entre les sous-systèmes.

La complexité de la synthèse de la communication dépend du niveau d'abstraction de la communication inter modules. Deux types de synthèse de la communication peuvent être nécessaires pour la conception d'un système hétérogène :

- Synthèse de la communication homogène, c'est-à-dire, le raffinement des interactions entre sous-systèmes spécifiés au même niveau d'abstraction. Cela consiste à transformer les canaux de communication abstraits qui connectent accès ou ports du même type. Ce type de synthèse est nécessaire pour le raffinement d'un modèle au niveau système (par exemple SDL) dans une réalisation.
- Synthèse de la communication hétérogène ou multiniveau, c'est-à-dire, le raffinement des interactions entre sous-systèmes spécifiés dans différents niveaux d'abstraction. Cela consiste à transformer des canaux de communication qui connectent accès ou ports de différents niveaux d'abstraction. Ce type de synthèse est nécessaire quand la spécification du système est composée de sous-systèmes spécifiés aux différents niveaux d'abstraction ou en différents langages.

3.1.1.2 Synthèse de la communication dans un système hétérogène

La synthèse de communication dans le cadre des systèmes hétérogènes multilangages prend en compte les différents types d'interfaces, les différents niveaux d'abstraction et les différents protocoles de

communication utilisés par les sous-systèmes. Une bibliothèque de communication est utilisée pour décrire les différentes réalisations possibles d'un protocole de communication. Le contrôleur est responsable du contrôle des échanges de données entre les sous-systèmes (par exemple, le contrôle d'accès à une FIFO) ou de l'adaptation des différents protocoles de communication utilisés par les sous-systèmes communicants.

3.1.2 Domaine d'application

La méthode de conception multilangage est utilisée pour la conception des systèmes hétérogènes dont les différentes parties du système appartiennent à différentes classes d'application, comme par exemple contrôle/données ou discret/continu, et utilisent différents langages de spécification. Les exemples de tels systèmes ne manquent pas : téléphones mobiles, interfaces utilisateur pour des produits de communication, téléviseurs numériques ou encore modules de commande dans une voiture, un avion ou un bâtiment.

3.1.2.1 Réutilisation de composants existants

La complexité croissante des systèmes embarqués nécessite la réutilisation de modules logiciel/matériel existants afin de réduire le temps de mise sur le marché. Ces modules ont une interface spécifique et figée, et des protocoles de communication spécifiques dont l'objectif est de satisfaire les contraintes de coûts et de performances. La plupart des méthodes de synthèse de la communication ne traitent pas ce type de systèmes.

La méthode de synthèse proposée dans ce travail permet l'intégration et la réutilisation des composants logiciels ou matériels existants. Cette méthode prend en compte l'interface définie pour le composant existant ainsi que le protocole de communication utilisé, et génère les interconnexions nécessaires à la communication entre les modules sans entrer dans les détails de la spécification du module existant. Ainsi, le concepteur peut modifier la fonctionnalité d'un composant existant ou remplacer un composant existant par un autre, sans être obligé de refaire le processus de synthèse. Pour cela, il est impératif de conserver l'interface et le protocole de communication du composant existant.

3.1.2.2 Interconnexion de modules décrits dans des langages différents

Un système multilangage est défini par un ensemble de modules basés sur des concepts différents et interconnectés via des canaux de communications et communiquant grâce à des schémas de communication. Chaque module peut utiliser un schéma de communication différent, selon sa spécification.

Les modules spécifiés au niveau système communiquent au travers d'un schéma de communication de haut niveau, tel que l'appel de primitives de communication. Dans ce cas, le protocole de communication sera défini au moment de la synthèse de communication. Par contre, les modules spécifiés à un niveau plus proche de la réalisation, ou composants existants, communiquent au travers d'affectations de signaux avec un protocole de communication bien défini. Le canal de communication a la charge d'offrir l'ensemble des primitives de communication utilisées par les modules et, éventuellement, de spécifier la conversion entre les protocoles de communication.

3.1.3 Contributions

Ce chapitre présente une nouvelle approche pour la synthèse de la communication qui prend en considération les systèmes hétérogènes multilangages. Cette approche est une extension de la méthodologie présentée par [21], adressant uniquement la synthèse de la communication des canaux abstraits homogènes portant sur des accès. La spécification d'entrée de l'étape de synthèse de la communication est composée d'un ensemble de sous-systèmes représentant les différentes parties du

système global (matérielle, logicielle et composant existant) qui communiquent via des canaux de communication abstraits. Les interconnexions entre les différentes parties qui composent le système global sont spécifiées par une liste de haut niveau. Le résultat de la synthèse est un ensemble de processus interconnectés par des signaux qui peuvent être pilotés par une unité de contrôle.

Les principaux objectifs de l'approche proposée dans cette thèse pour la synthèse de la communication des systèmes hétérogènes multilangages sont :

- Réaliser la synthèse de communication à partir d'une spécification d'un système hétérogène multilangage,
- Modéliser la communication indépendamment de la spécification du système afin de permettre des changements dans le modèle de communication sans modification de la spécification du système,
- Offrir le choix entre différents protocoles de communication décrits dans une bibliothèque de communication,
- Réutiliser des protocoles de communications existants.

Les principaux avantages de l'approche proposée pour la synthèse de la communication sont :

- Une synthèse de la communication des systèmes hétérogènes multilangages,
- Une synthèse complète de la communication :
  - Synthèse de réseau de communication et sélection de protocole de communication,
  - Synthèse d'interface et adaptation des différentes interfaces de communication,
- Une réutilisation des protocoles de communication.

Les limitations de l'approche proposée sont :

- La nécessité d'avoir une bibliothèque de protocoles de communication fournie par l'utilisateur.
- La nécessité de disposer de fonctions d'estimation et de coût permettant de guider la sélection de protocole et la synthèse de la communication.

La section 3.2 présente la modélisation de la communication dans le cadre des systèmes homogènes et l'extension de ce modèle pour les systèmes hétérogènes multilangages. La synthèse de communication pour les systèmes homogènes ainsi qu'une approche pour la synthèse de la communication des systèmes hétérogènes sont présentées dans la section 3.3. La section 3.4 illustre le flot de synthèse de la communication dans le cadre des systèmes hétérogènes multilangages en utilisant l'exemple du contrôleur de bras de robot. Finalement la section 3.5 présente des conclusions.

3.2 Travaux Antérieurs

Cette section présente les travaux portant sur la synthèse de la communication et développe les travaux de synthèse de la communication dans les systèmes homogènes [21].

La plupart des travaux sur la synthèse de la communication pour le codesign se sont concentrés sur la synthèse d'interfaces en partant du principe d'une spécification homogène du système [20], [64], [76], [25]. Dans [105], Vahid présente une bibliothèque de communication à objet pour le codesign. L'outil de cosimulation PIA [44] permet la spécification de plusieurs modèles de communication pour chaque interface du système. Les modèles de communication peuvent être échangés dynamiquement pendant la simulation du système. L'environnement CoWare [111] permet au concepteur de spécifier et simuler les interfaces de communication à plusieurs niveaux d'abstraction. Ces approches effectuent la synthèse de la communication à partir d'une spécification unifiée du système.

Plusieurs travaux abordent le problème de la synthèse d'interface. Dans [24], Ecker présente une méthode de transformation et d'optimisation des protocoles de communication. Narayan [75] aborde le problème de la génération d'interfaces entre deux modules matériels d'une spécification. L'objectif est d'optimiser l'utilisation d'un bus en y entrelaçant les différentes communications point à point. Dans [77], [60], Narayan et Lin considèrent le problème de la synthèse d'interfaces avec conversion automatique du protocole pour une ou deux interfaces fixées. Rowson et Sangiovanni-Vicentelli [93] proposent une nouvelle méthodologie basée sur la conception d'interfaces qui considère le comportement et la communication comme des éléments orthogonaux. Le but est la réutilisation des unités de communication. SpecSyn [31], [32] génère des bus pour la communication entre deux processus pour une technique de raffinement d'interface. Cette méthode analyse la taille des données qui seront transmises et le flux de données. Ainsi, il est possible de déterminer la bande passante du bus généré et de décider de fusionner les canaux de communication. Yen et Wolf [114] traitent de la synthèse de la communication dans les architectures hétérogènes distribuées sur une topologie arbitraire. Cette approche crée un nouvel élément de calcul et un bus lorsqu'il n'est plus possible d'affecter un processus à un élément de calcul déjà existant ou une communication à un bus tout en respectant les contraintes fixées. Ortega et Borriello [81] traitent du problème de la synthèse de la communication pour une topologie de bus arbitraire spécifiée par le concepteur. Dans cette approche, les fonctions de communication de haut niveau sont raffinées en des éléments de calcul spécifiques à l'architecture. Une extension de cette approche pour l'intégration de composants existants est présentée dans [15]. Polis [4] utilise un modèle de communication basé sur des machines d'états finis étendues pour le codesign. La communication est globalement asynchrone et localement synchrone avec des files d'attente bornées entre les machines d'états finis. Dans cette approche les communications entre des modules logiciels ou entre des modules logiciels et matériels sont réalisées au travers d'une mémoire partagée ou au travers de portes d'entrée et de sortie. Un seul bus peut être utilisé, l'utilisation de plusieurs bus n'est pas autorisée. Le projet Adéquation Algorithme Architecture et le projet MOCAT, [5] et [70], présentent des méthodes et outils d'étude des communications après la synthèse comportementale. Ceux-ci permettent de trouver le compromis entre la vitesse, le coût et la consommation par une approche « à grain fin » des communications. Daveau [22] part d'une description comportementale et sélectionne automatiquement un protocole de communication décrit dans une bibliothèque de communication. Cette approche résout le problème de synthèse de la communication pour les systèmes homogènes monolangages.

3.2.1 Modélisation de la communication dans le cadre des systèmes homogènes.

Cette section introduit le modèle de communication présenté dans [48] apportant une très grande flexibilité au schéma de communication dans le cadre des systèmes homogènes.

3.2.1.1 Modèle de communication homogène

Un système est représenté par un ensemble de processus communicants. Le modèle de communication présenté ici permet de dissocier la description des communications de la description de la fonction des processus composant un système. Pour cela, la notion de canal de communication est introduite.

Un canal abstrait est un canal de communication, qui spécifie les primitives de communication requises par les processus qui l'utilisent, mais qui ne spécifie ni le protocole suivi ni la réalisation de ce dernier. Il offre l'ensemble des primitives de communication utilisées par les processus lui étant rattachés. Un processus désirant communiquer au travers d'un canal abstrait appelle à distance la primitive de communication adaptée du canal abstrait.

Ce modèle de communication de haut niveau permet de s'abstraire de toute hypothèse sur la réalisation des communications lors de la conception d'un système. Les détails de la communication sont « englobés » et ne laissent apparaître qu'une transmission de données. La communication est alors

totalement transparente pour les processus appelants. Cette approche autorise la conception d'un système en deux temps, tout d'abord chaque processus et ensuite les communications.

L'utilisation d'appels de procédure à distance permet de séparer la spécification de la communication du reste du système. Les modèles de communication peuvent être décrits séparément et les détails d'implémentation sont cachés. Dans cette approche, les détails d'implémentation et le protocole sont cachés dans une bibliothèque de composants de communication.

3.2.1.2 Interconnexion des modules homogènes

Deux types d'interconnexions sont nécessaires pour représenter les communications dans la description d'un système : un type abstrait et un type proche de la réalisation correspondant aux deux niveaux de spécification : le niveau système et le niveau proche de la réalisation.

La description au niveau système correspond à un système défini comme un ensemble de processus communicants au travers de schémas de communication de haut niveau. Les détails du protocole de communication sont englobés dans des primitives de communication utilisées par les processus. Cette description ne s'intéresse pas à la description des communications. La description de la réalisation correspond à un ensemble de processeurs interconnectés via des signaux physiques, c'est-à-dire un système interconnecté. Ici les communications sont détaillées au plus bas niveau par l'affectation de signaux selon le protocole défini. La synthèse de la communication dans le cadre des systèmes homogènes consiste à transformer la spécification d'un système communicant en la spécification d'un système interconnecté.

L'interconnexion abstraite s'exprime par la connexion des accès spécifiés par les interfaces des modules interconnectés à l'aide d'un canal abstrait. L'interconnexion concrète s'exprime par la connexion des ports physiques des modules interconnectés à l'aide d'un canal physique (un fil).

3.2.1.3 Notion de canal abstrait homogène

Le concept de canal abstrait permet de représenter les schémas de communication sans faire d'hypothèses sur leur réalisation. En effet, la spécification initiale d'un système définit la fonctionnalité et non les détails de réalisation, ceci afin de ne pas soumettre prématurément le système à des contraintes spécifiques. Il est donc préférable de ne pas définir la technique (un protocole particulier) de communication employée pour chaque canal à ce niveau de spécification.

Un canal abstrait est un « objet » capable d'assurer la communication suivant un protocole. Il offre un ensemble de primitives de communication permettant son utilisation. Une primitive de communication est une procédure (une « méthode ») utilisée pour la communication entre processus. Un processus désirant communiquer au travers d'un canal fait appel à une primitive fournie par ce dernier. Les informations à transmettre sont données en paramètres à la primitive et l'appel de la primitive constitue un appel de procédure à distance [2], [10].

Les primitives fournies par un canal constituent l'unique partie visible du canal. L'ensemble des primitives d'un canal de communication est défini par les appels des processus connectés et est donc spécifique à chaque processus ou à chaque canal de communication.

L'appel à distance et l'exécution indépendante de la primitive de communication permettent de dissocier totalement les communications des processus dans le système. Ainsi il est possible de faire totalement abstraction des détails des communications en représentant le système sous la forme d'un ensemble de processus communiquant au travers de canaux abstraits.

La Figure 19 représente un ensemble de processus communiquant au travers d'un réseau de canaux abstraits. Le canal abstrait *c3* offre les primitives de communication *put* et *get*. Ces primitives sont utilisées par les processus *A, C* et *D* pour communiquer.

Cette représentation du système permet de ne pas spécifier explicitement les primitives fournies par un canal, celles-ci sont déduites des appels effectués au travers du canal abstrait. Un canal abstrait a donc la particularité d'offrir toutes les primitives de communication requises.

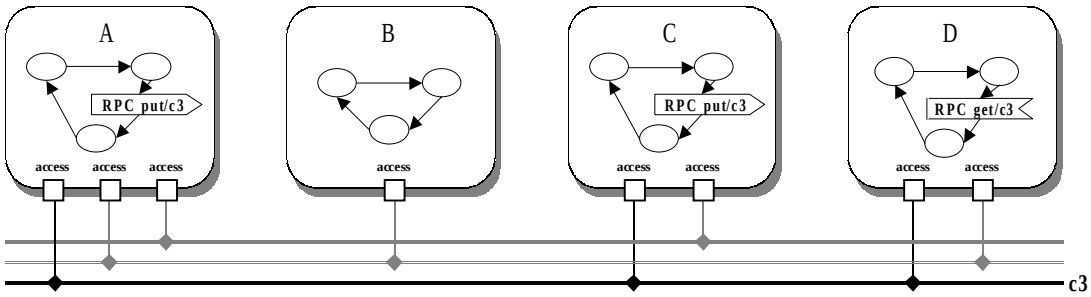

Figure 19. Spécification au niveau système

3.2.1.4 Modélisation des unités de communication homogène

Une unité de communication est l'abstraction d'un composant physique. Les unités de communication sont sélectionnées dans la bibliothèque et « instanciées » lors de la synthèse de la communication. La modélisation employée est présentée dans [48].

Une unité de communication est un objet fournissant un ensemble de primitives de communication (méthodes) réalisant un protocole spécifique. Elle est composée de l'ensemble des primitives de communication, un éventuel contrôleur de communication, une interface et un ensemble d'interconnexions.

Le contrôleur détermine le protocole de la communication, assure la synchronisation de la communication et l'absence de conflits d'accès. Il offre un ensemble de ressources de communication (bus, arbitres de bus, mémoire) qui sont utilisées par les différentes primitives de communication. La complexité du contrôleur peut varier d'une simple file d'attente « *first-in-first-out* » à un protocole complexe en couches. Certains protocoles de communication, tels que le rendez-vous (« *handshake* »), ne nécessitent pas de contrôleur, le contrôle est alors complètement distribué entre les primitives de communication. Les primitives de communication interagissent avec le contrôleur au travers de l'interface pour effectuer la communication.

Les primitives de communication sont le lien entre le canal abstrait et la réalisation finale. Une primitive prend en paramètre les données à communiquer et accède à l'unité de communication (le contrôleur éventuel ou la primitive duale) au travers de ports (fils ou bus) définis par l'interface. Une primitive de communication contient l'algorithme nécessaire au respect du protocole de l'unité de communication. Lors de la synthèse de la communication, la primitive de communication remplace l'appel à distance. La communication s'effectue alors selon le protocole choisi au travers des ports de chaque primitive.

L'approche modulaire permet la conception séparée du système et des unités de communications. Il est possible de développer plusieurs réalisations d'un même canal ou protocole et de cacher les détails des communications lors de la conception des différentes parties du système. Cette méthodologie autorise la réutilisation des unités de communication grâce à la modularité.

3.2.2 Synthèse de la communication dans le cadre des systèmes homogènes

Dans le cadre des systèmes homogènes, la synthèse de la communication consiste à transformer un système communicant à l'aide de primitives de haut niveau et au travers de canaux abstraits en un système interconnecté communicant au travers de signaux. A partir de la spécification système deux étapes sont nécessaires. La première étape vise à allouer un protocole à chaque canal de communication et à fixer la

topologie du réseau d'interconnexion. La seconde étape consiste à générer et adapter les interfaces des différents processus au réseau de communication construit.

Le flot global de synthèse de la communication est présenté par la Figure 20. Le modèle d'entrée est un ensemble de processus connectés par des canaux abstraits. Les processus communiquent par appel de procédure à distance. Les procédures appelées sont des primitives de haut niveau fournies par le canal abstrait connecté au processus appelant. Le résultat de l'étape de synthèse des communications est un système de processus communiquant par appels de procédures locales et connectés par le biais d'un ensemble de signaux de contrôle et de données. La synthèse de la communication utilise une bibliothèque de communication décrivant un ensemble d'unités de communication. Le comportement des primitives et du contrôleur de communication et leurs interfaces sont décrits dans la bibliothèque. La bibliothèque de communication est utilisée lors de l'allocation des unités de communication pour déterminer si l'unité de communication allouée est capable de fournir toutes les primitives requises par le canal abstrait. L'étape de synthèse s'appuie sur la bibliothèque pour raffiner la communication en une spécification de bas niveau.

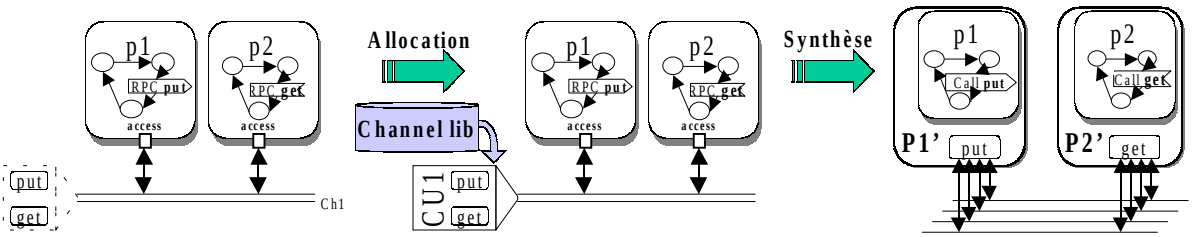

Figure 20. Flot global de synthèse de la communication homogène

3.2.2.1 Fusion de canaux abstraits

La fusion de plusieurs canaux abstraits en une seule unité de communication (après synthèse) permet de partager un même médium de communication entre plusieurs communications abstraites et permet ainsi de réduire les interfaces et les interconnexions nécessaires. Il s'agit d'un partage de ressources au niveau des communications; les canaux fusionnés s'exécutent par le biais du même contrôleur et les communications correspondant aux différents canaux abstraits restent indépendantes. Une approche pour déterminer la taille d'un bus qui met en œuvre un ensemble de canaux est présentée dans [29] et [75]. Vahid [105] propose une approche de synthèse d'interface permettant de réduire les interfaces en partageant les ports d'entrée/sortie entre différentes fonctionnalités.

3.2.2.2 Sélection de protocole et allocation des unités de communication

L'allocation d'unités de communication prend en entrée un ensemble de processus communiquant au travers de canaux abstraits et une bibliothèque d'unités de communication. Cette étape choisit dans la bibliothèque un ensemble d'unités de communication capables d'exécuter toutes les primitives de communication requises par les processus. Le choix d'une unité de communication particulière ne dépend pas seulement de la communication à exécuter (données à transférer), mais aussi des performances requises et de la réalisation des processus concernés. Cette étape fixe le protocole de chaque canal abstrait en choisissant une unité de communication avec un protocole défini et associe les primitives disponibles aux primitives requises par le canal abstrait. Elle fixe la topologie du réseau d'interconnexion en déterminant le nombre d'unités de communication allouées aux canaux abstraits de la spécification.

3.2.2.3 Synthèse d'interface

L'étape de synthèse consiste à appliquer au canal abstrait le protocole sélectionné dans la bibliothèque lors de l'allocation. La synthèse d'interface génère les interfaces et interconnexions requises par les processus

utilisant les unités de communication et introduit dans la spécification la réalisation du contrôleur éventuel de communication. Le résultat de la synthèse d'interface est un ensemble de processeurs interconnectés communiquant au travers de bus, de signaux et d'éventuels contrôleurs de communication provenant de la bibliothèque tels que des arbitres de bus, des files d'attente.

La synthèse d'interfaces se compose de quatre opérations :

- La génération des interfaces de chacun des processus accédant au canal abstrait. L'ensemble des ports associés aux primitives intégrées à un processus remplace le point de connexion au canal abstrait (l'accès).
- La génération de l'éventuel contrôleur de communication (file d'attente, arbitre de bus, etc.). Le corps de l'unité de communication est transformé en une unité logique lorsqu'un comportement y est décrit.
- La génération de l'ensemble des interconnexions entre les interfaces et les contrôleurs éventuels de communication. Les interconnexions précisées par l'unité de communication choisie remplacent le lien logique modélisé par le canal abstrait.
- La transposition des primitives fournies par l'unité de communication sélectionnée à l'intérieur de chaque processus requérant et la transformation des appels à distance de procédure en appels locaux. Les primitives (*méthodes*) contenues dans l'unité de communication choisie sont déplacées au sein de chaque processus requérant et sont transformées en procédures.

Cette approche apporte la souplesse du choix du protocole de communication, il est possible de transposer un canal abstrait en un protocole simple tel un rendez-vous ou un protocole complexe telle une file d'attente multi-point.

3.3 Modélisation et synthèse de la communication dans le cadre des systèmes hétérogènes

La synthèse de la communication a pour but de transformer un système composé de processus, qui communiquent au travers de canaux abstraits, en un ensemble de processeurs interconnectés via des signaux partageant le contrôle des communications. Cette section introduit la modélisation de la communication et le flot global de la synthèse de la communication pour les systèmes hétérogènes.

3.3.1 Modélisation de la communication dans le cadre des systèmes hétérogènes

Cette section introduit une extension du modèle de communication présenté dans la section 3.2.1 afin de mieux appréhender la communication dans le cadre des systèmes hétérogènes.

3.3.1.1 Modèle de communication pour les systèmes hétérogènes

La spécification de la communication dans le cadre des systèmes multilangages prend en considération le niveau d'abstraction de la spécification de chaque sous-système. Le système global est spécifié comme un ensemble de sous-systèmes communicants. Chaque sous-système peut être décrit à un niveau d'abstraction différent et peut avoir des types d'interfaces et schémas de communication différents. Les communications entre les sous-systèmes sont assurées par des canaux de communication abstraits hétérogènes. La communication reste totalement transparente pour les sous-systèmes.

Les deux modes de connexion disponibles pour un sous-système dans le modèle homogène sont l'appel de procédure à distance ou la connexion à un fil. Dans le modèle hétérogène un nouveau type d'interface apparaît : l'interface contrainte à un protocole fixé, par la suite appelé *connecteur* en référence aux

connecteurs normalisés (Centronic, RS232, SCSI, etc.). Ce type d'interface permet la modélisation des interfaces d'un composant existant ou d'un module décrit dans un langage différent. Aucune restriction n'est imposée au comportement ou à la spécification du module. Le connecteur est une interface qui enveloppe le module IP (Figure 21). Il permet la spécification de l'interface de communication d'un module IP avec le reste du système de façon homogène. Cela autorise l'utilisation conjointe des modules spécifiés à différents niveaux d'abstraction et, par conséquent, chaque module peut avoir différents types d'interface de communication et protocoles de communication.

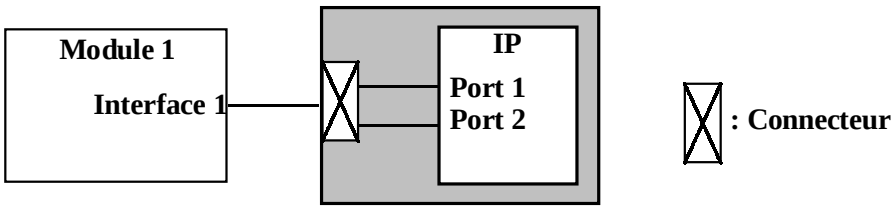

Figure 21. Exemple d'un connecteur

Afin de conserver un haut niveau d'abstraction, le canal abstrait supporte la connexion à un connecteur et doit être raffiné en fonction de cette extension. Le niveau d'abstraction atteint permet de s'abstraire des communications lors des premières étapes de conception d'un système, rejoignant ainsi l'approche des systèmes homogènes.

La mise en œuvre de telles communications passe par l'étape de synthèse des communications hétérogènes et par l'utilisation d'une bibliothèque de réalisation précisant les détails de mise en œuvre des différents protocoles.

3.3.1.2 Modèle d'interconnexion des modules hétérogènes : introduction du concept de connecteur

Afin de prendre en compte le concept d'interface contrainte, le connecteur est introduit à trois niveaux :

- **La définition du connecteur**, un connecteur est un ensemble d'éléments d'interface associés et sur lesquels porte un protocole fixé (ex. SCSI, RS232, etc.). La définition d'un connecteur est donnée par une bibliothèque de définitions accessible à l'utilisateur pour extension.
- **L'utilisation du connecteur,** dans la spécification des interfaces des modules d'un système ou d'une unité de communication, le connecteur est utilisé comme un nouveau type d'élément d'interface d'un module.
- **L'accès aux éléments d'un connecteur** est nécessaire lors de la spécification d'une unité de communication. Une primitive est introduite à cet effet et permet la sélection d'un élément du connecteur.

3.3.1.3 Notion de canal abstrait hétérogène

Le modèle homogène propose un canal abstrait limité aux interconnexions de points de connexion du type accès proposant l'ensemble des procédures appelées aux travers des connexions. Dans le modèle hétérogène, le canal abstrait est étendu aux interconnexions mixtes : un accès avec un connecteur ou un connecteur avec un autre connecteur. Le canal abstrait fournit alors l'ensemble de la logique de contrôle nécessaire au respect du protocole de chaque connecteur.

Un système est décrit par un ensemble de sous-systèmes communicants. Chaque sous-système communique avec les autres par des ports de communication. Un port de communication peut être de trois types :

- Un *fil* correspond à un port HDL,
- Un accès de haut niveau correspond à un canal abstrait,
- Un *connecteur* correspond à une vue abstraite d'une réalisation, par exemple, un bus. Un connecteur est un ensemble de fils contraints à un protocole.

La Figure 22 montre les types de canaux abstraits disponibles dans le modèle hétérogène :

- **C1** : Connexion abstraite entre deux accès. Le canal est laissé entièrement libre quant à sa réalisation. Il s'agit de la version issue du modèle homogène, la plus flexible.
- **C2, C4** : Connexions abstraites entre une interface contrainte (connecteur) et un accès. Le canal abstrait est entièrement chargé de la communication mais doit être adapté à la connexion contrainte.
- **C3** : Connexion abstraite entre deux interfaces contraintes (connecteurs). Le canal abstrait est ici chargé d'adapter les protocoles des deux connexions.

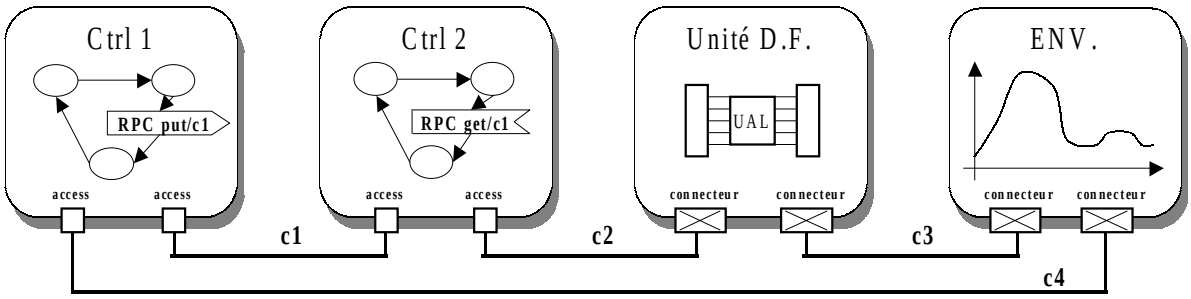

Figure 22. Les canaux abstraits hétérogènes

La validation d'un tel système peut être réalisée par cosimulation si le bus de cosimulation modélise le comportement du canal abstrait ou si le concepteur introduit des modules spécifiques pour la cosimulation afin de réaliser l'adaptation des interfaces de communication. Un exemple plus détaillé de l'utilisation des connecteurs est présenté dans la section 4.4.

3.3.1.4 Modélisation des unités de communication hétérogène

L'unité de communication issue du modèle homogène se propose de décrire une réalisation d'un canal abstrait. Tout comme le canal abstrait est étendu pour le modèle hétérogène, les unités de communication sont étendues aux canaux abstraits hétérogènes. Afin d'autoriser la description d'une réalisation de canaux mixtes, les connecteurs sont employés aux niveaux de l'interface et de la description du comportement. L'unité de communication hétérogène intègre les trois parties principales :

- *Les méthodes* sont appelées aux travers des points de connexion logique.
- *Le contrôleur* est employé si nécessaire afin d'assurer le protocole inhérent aux connecteurs de l'unité de communication. Il est parfois possible d'intégrer toute la logique de contrôle aux méthodes.
- *Les connexions internes* définissent les interconnexions entre les éléments de l'interface de l'unité de communication.

La Figure 23 montre une unité de communication utilisant un connecteur et une unité définissant uniquement l'interconnexion entre deux connecteurs série.

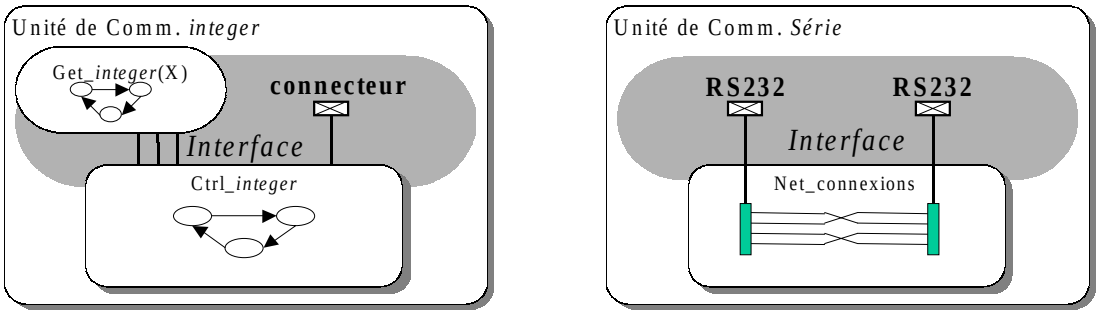

Figure 23. Unités de communication hétérogènes

3.3.2 Flot global de synthèse de la communication pour les systèmes hétérogènes

3.3.2.1 Modèle d'entrée

Le modèle d'entrée est un ensemble de modules connectés par des canaux abstraits, tel que présenté dans la section 3.3. Les canaux abstraits lient des points de connexion de types différents. Les modules peuvent être décrits dans des langages différents et présenter différents concepts. La Figure 22 présente un exemple de système avant synthèse de la communication proposant tous les types de canaux possibles.

3.3.2.2 Modèle produit par la synthèse de la communication

Le résultat de la synthèse de la communication est un système composé de modules interconnectés par des signaux sur lesquels porte un protocole défini lors de la synthèse de la communication. Des modules supplémentaires sont automatiquement créés afin d'assurer le contrôle des communications maintenant explicitées.

3.3.2.3 Méthode de synthèse de la communication

La synthèse de la communication utilise une bibliothèque de communication. La bibliothèque de communication contient un ensemble d'unités de communication permettant de choisir le protocole adapté au canal abstrait devant être synthétisé. Une unité de communication, comme présenté en 3.3.1.4, est constituée d'une interface, décrivant les méthodes et les connecteurs d'entrée / sortie, et d'un module décrivant les interconnexions et le contrôleur éventuel d'adaptation des protocoles.

Deux étapes sont nécessaires à la synthèse de la communication. La première étape, l'allocation, vérifie l'applicabilité de l'unité de communication sélectionnée et la paramètre. Cette étape est appelée sélection de protocole et allocation des unités de communications. La seconde étape consiste à transformer le canal abstrait en une réalisation selon l'unité de communication choisie. Cette étape est appelée synthèse d'interface.

3.3.2.3.1 Sélection de protocole et allocation des unités de communications

Le choix de l'unité de communication est laissé à l'expertise de l'utilisateur. L'allocation consiste à valider le choix par une vérification de la correspondance des besoins du canal abstrait et des services et connexions fournis par l'unité de communication sélectionnée. La seconde tâche de l'allocation consiste à paramétrer l'unité de communication en fonction de son emploi, en particulier la réutilisation des signaux de donnée et la duplication des signaux de contrôle.

3.3.2.3.2 Synthèse d'interface

L'étape de synthèse d'interfaces consiste à appliquer l'unité de communication au canal abstrait à transformer. La synthèse d'interfaces génère les interfaces et les interconnexions requises par les modules connectés au canal abstrait et introduit dans la spécification la réalisation du contrôleur de communication éventuel. Le résultat de la synthèse d'interfaces est un ensemble de modules interconnectés communiquant au travers de signaux ou bus.

La synthèse d'interface est en charge de trois étapes de transformation : l'expansion des connecteurs en un ensemble d'éléments physiques, la génération des interconnexions et la génération du contrôleur ou adaptateur éventuel.

Cette approche apporte la souplesse du choix du protocole de communication et rend possible l'interconnexion de tout type de module.

3.3.2.4 Outil de synthèse pour les systèmes hétérogènes

L'approche de la synthèse repose sur l'emploi de primitives permettant d'opérer la transformation par étapes successives. La section 3.3.2 a mis en évidence les étapes nécessaires à la synthèse de la communication hétérogène :

- *L'allocation d'une unité de communication,* cette étape met en correspondance l'unité de communication et le canal abstrait. Cette opération permet de vérifier l'adéquation entre le canal et l'unité de communication sélectionnée. Cette étape utilise la primitive développée dans le cadre des systèmes homogènes.
- *La synthèse d'interface,* cette étape génère les interfaces et les interconnexions selon l'unité de communication et les canaux abstraits choisis.

L'allocation étant résolue par l'emploi des primitives développées pour les systèmes homogènes, seule l'étape de synthèse est développée.

3.3.2.4.1 Extension du langage SOLAR

Afin de prendre en compte le concept d'interface contrainte dans le format intermédiaire SOLAR, le connecteur est introduit à trois niveaux :

```
(SOLAR librairie
 (LIBRARYDEF librairie

(CONNECTDEF typeA
  (PORT VALEURM1 (DIRECTION OUT) (INTEGER))
  (PORT ack_put      (DIRECTION OUT) (BIT))
 )

 (CONNECTDEF typeB
  (PORT CONSM1 (DIRECTION IN)  (INTEGER))
  (PORT donnee   (DIRECTION IN)  (BIT))
  (PORT ack_get  (DIRECTION OUT) (BIT))
 )
)
)
```

Figure 24. Bibliothèque de définitions des connecteurs

- **La définition du connecteur** est donnée par une bibliothèque de définitions accessible à l'utilisateur pour extension. La Figure 24 présente un exemple de bibliothèque de définitions, ici le connecteur *typeB* contient trois ports physiques.
- **Le connecteur** est ajouté aux types d'éléments d'interface. La Figure 25 présente la spécification d'un module comportant le connecteur présenté en Figure 24.

```
(DESIGNUNIT   Fd_controler
   (VIEW  Fd_controler_behaviour
     (INTERFACE
       (CONNECTOR connecteurG1 typeA)
       (PORT  data_int (DIRECTION  IN) (INTEGER) )

     (CONTENTS
      (STATETABLE controlleur
       (STATE gerer_put
         …
         (ASSIGN (CONNECTFIELD  connecteurG1 ack_put ) '1')
         …
       )
       (STATE aprouve_put
         …
```

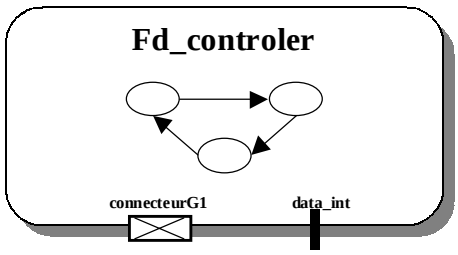

Figure 25. Module muni d'un connecteur

- **L'accès aux éléments d'un connecteur** est assuré par la primitive *connectfield* permettant la sélection d'un élément du connecteur. La Figure 25 présente la méthode d'utilisation de la primitive *connectfield* pour la spécification d'une unité de communication employant le connecteur présenté en Figure 24.

3.3.2.4.2 *Synthèse de la communication hétérogène*

La synthèse de la communication hétérogène consiste en deux points principaux : la synthèse des points de connexion, de type accès ou de type contraint (connecteur), et la synthèse de la logique d'interconnexion.

3.3.2.4.2.1 Synthèse d'un point de connexion de type accès

La synthèse d'un point de connexion de type accès est issue de la synthèse homogène et consiste à transformer les appels de procédures à distance du module en appels locaux. Pour cela, les primitives appelées sont incorporées au module et l'accès est transformé en l'ensemble de ports logiques constitué par l'union des interfaces des procédures incorporées. Le canal abstrait est transformé en un bus logique. La Figure 20 présente l'intégration des procédures *put* et *get* aux processus *P1* et *P2* du système.

3.3.2.4.2.2 Synthèse d'un point de connexion de type connecteur

La synthèse d'un point de connexion contraint consiste à « déployer » le connecteur. Ce type de point de connexion étant destiné aux modules externes, aucune transformation n'est appliquée au module, seule l'interface, la vue externe, est modifiée. Le « déploiement » consiste à transformer un connecteur en chacun de ses éléments.

3.3.2.4.2.3 Synthèse de la logique d'interconnexion

L'unité de communication sélectionnée fournit les informations nécessaires à la construction des connexions entre les modules connectés au canal abstrait et à la génération d'un contrôleur s'il s'avère nécessaire. Les deux étapes de la synthèse de la logique d'interconnexion sont :

- *La synthèse de l'éventuel contrôleur* consiste à construire un nouveau bloc de contrôle à partir de la spécification fournie par l'unité de communication sous la forme d'une machine d'états finis ou sous la forme d'un module externe.
- *La synthèse des interconnexions* directes consiste à construire les connexions directes entre les éléments des connecteurs ou les ports des procédures selon la spécification fournie par l'unité de communication (Figure 23).

3.3.2.4.3 La primitive map hétérogène

La primitive *map hétérogène* transforme un ensemble de modules communiquant au travers de canaux de communication en un ensemble de modules interconnectés communiquant au travers de signaux ou bus (étape de synthèse d'interface, cf. section 3.3.2.3.2). Pour les modules en conception, la primitive *map hétérogène* réalise les opérations suivantes :

- La génération des interfaces pour chaque module accédant au canal de communication. Les ports utilisés par les primitives de communication ainsi que l'interface du contrôleur de communication sont décrits dans la bibliothèque de communication ;
- L'*instanciation* des éventuels contrôleurs de communication ;
- La génération de la *netlist* d'interconnexion reliant les interfaces des contrôleurs et des modules IP ;
- L'expansion en ligne des primitives de communication pour chaque module en conception. Le corps des primitives et le corps du contrôleur sont décrits dans la bibliothèque de communication.

La Figure 26 représente le système de la Figure 22 après l'application de la primitive *map hétérogène* en utilisant les unités de communications de la Figure 23. Le système est composé de quatre modules : *ctrl1*, *ctrl2* (modules en conception) et *unité D.F.*, *env* (modules IP). L'interconnexion entre les modules est spécifiée par quatre canaux de communication : *c1* (interconnecte deux points de connexion de type accès), *c3* (interconnecte deux points de connexion contraints) et *c2*, *c4* (interconnecte un point de connexion de type accès et un point de connexion contraint).

La primitive *map hétérogène* est appliquée aux canaux *c2* et *c3*. Seuls les ports nécessaires à chaque primitive de communication sont inclus dans l'interface de chaque module en conception. Ceux-ci sont connectés soit aux ports d'autres modules, soit aux ports de l'éventuel contrôleur de communication. Aucune transformation n'est appliquée aux modules IP. Dans ce cas, la primitive *map hétérogène* décompose le connecteur en chacun de ses éléments et interconnecte ces derniers soit avec le contrôleur, soit avec les autres modules du système.

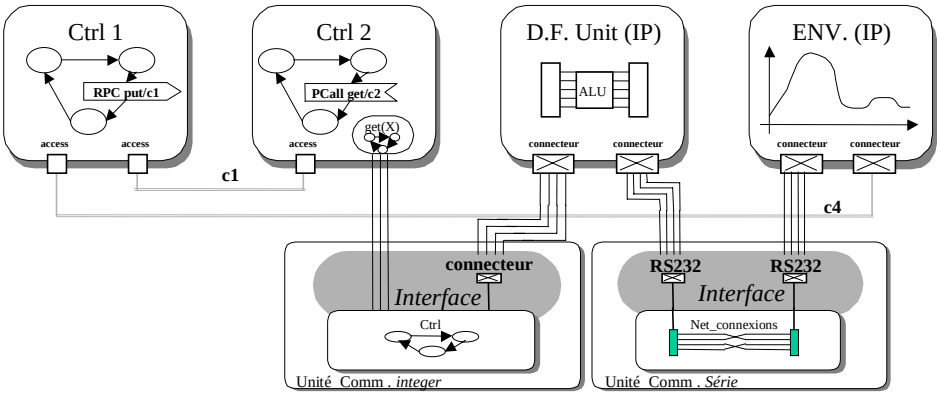

Figure 26. Résultat de l'application de la primitive *map hétérogène*

3.4 Résultat : l'exemple du contrôleur de bras de robot

Cette section présente une application du flot de synthèse de la communication pour un système hétérogène multilangage. L'exemple est ici un contrôleur de bras de robot. Le but de ce système est de valider la fonctionnalité d'un contrôleur de moteurs basé sur le principe du correcteur PID (Proportionnel Intégral Dérivé). Au travers de cet exemple sera démontré le flot de synthèse de la communication présenté dans la section 3.3.2 à partir d'une spécification fournie en SDL pour la partie électronique, et en

Matlab pour la partie mécanique, vers une architecture distribuée matérielle et logicielle sous la forme de modules C, VHDL et Matlab.

3.4.1 Modélisation du système de contrôle de moteur

Le contrôleur de bras de robot est un système qui ajuste les positions et les paramètres de vitesse jusqu'à dix-huit moteurs commandant le bras d'un robot à trois doigts. Ce système reçoit, d'une machine hôte, les informations liées au mouvement que le bras du robot doit exécuter et ajuste la vitesse de chaque moteur de façon à ce que le mouvement du bras soit doux et que les contraintes physiques d'accélération et de freinage soient respectées. La tâche du contrôleur est d'éviter la discontinuité dans le fonctionnement de chaque moteur. Cet exemple sera limité à deux moteurs afin de simplifier la présentation et de rendre l'exemple plus clair. Le schéma global du contrôleur est donné sur la Figure 27.

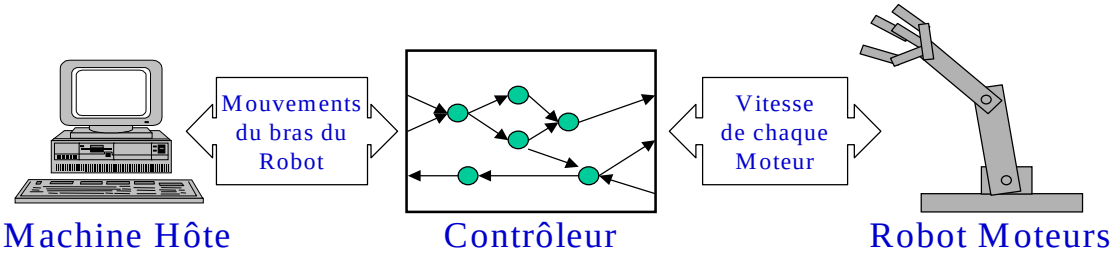

Figure 27. Schéma global du contrôleur de bras de robot

Le système peut être divisé en deux parties, les moteurs du bras et le contrôleur. Le langage SDL est utilisé pour la description du contrôleur et le langage Matlab est utilisé pour modéliser le comportement physique des moteurs.

3.4.1.1 Flot de conception du système de contrôle de moteur

La conception du système de contrôle de moteurs a été réalisée à l'aide de l'environnement de conception conjointe MUSIC [17], [100], [108]. La Figure 28 présente le flot de conception utilisé.

La conception du système de contrôle de moteurs commence avec une analyse des besoins du système et de la définition à haut niveau de leurs fonctions. Ensuite, un découpage manuel du système en plusieurs sous-systèmes (électronique, mécanique, etc.) est réalisé car il n'existe pas encore une spécification formelle. Le résultat du découpage est une spécification du système composée de deux sous-systèmes : un sous-système électronique appelé « contrôleur », spécifié en SDL, et un sous-système mécanique appelé « moteurs », spécifié en Matlab. Ces sous-systèmes communiquent par des canaux de communication abstraits. A cette étape de conception, le concepteur obtient un modèle multilangage de l'application au niveau système. La validation est réalisée à travers la cosimulation fonctionnelle du modèle multilangage. L'outil de cosimulation interprète les communications abstraites entre les sous-systèmes.

La prochaine étape de conception est le raffinement du sous-système SDL et la synthèse de la communication. Le sous-système SDL est raffiné dans un modèle composé de modules matériels (spécifiés en VHDL) et de modules logiciels (spécifiés en C). La communication est raffinée du niveau application au niveau registre (voir section 3.1.1.1.3). Chaque canal de communication abstrait sera réalisé en conformité avec le protocole de communication choisi par le concepteur. Après cette étape de conception, la cosimulation au niveau architecture est réalisée afin de valider le modèle produit (VHDL, C, Matlab) et les protocoles de communications utilisés.

Après le modèle validé au niveau architecture, la prochaine étape de conception consiste à raffiner le modèle VHDL en un modèle RTL et le modèle C sera exécuté sur un modèle de processeur. Dans ce cas, le processeur choisi est le ST10. La communication sera raffinée du niveau registre au niveau physique.

La dernière étape de conception consiste à valider le système par cosimulation au niveau cycle (VHDL-RTL, C, Matlab).

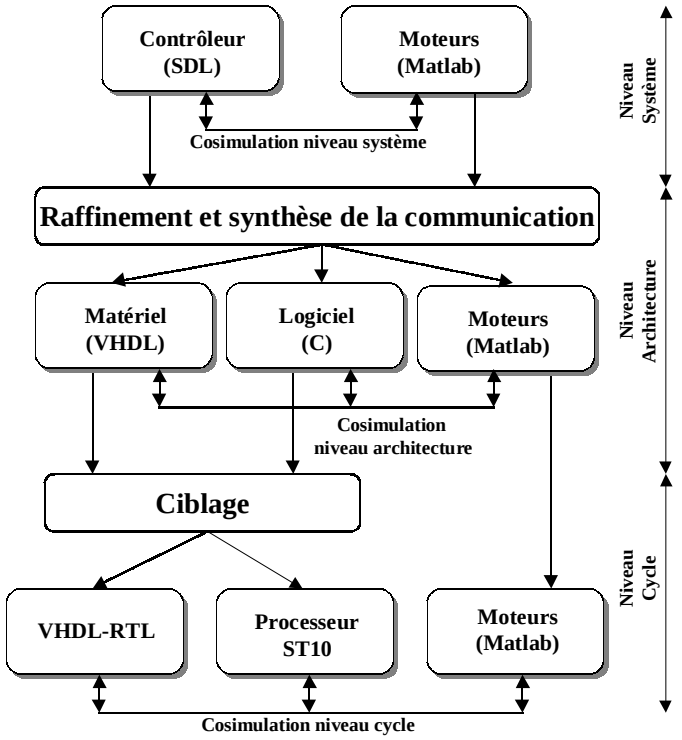

Figure 28. Flot de conception du système de contrôle de moteurs

3.4.1.2 Modélisation du contrôleur PID

La modélisation du contrôleur PID est composée de trois parties, la spécification mathématique du contrôleur PID, l'algorithme et la spécification SDL. La spécification mathématique décrit les équations du correcteur de type PID. L'algorithme montre une vue synoptique du comportement du contrôleur PID et la spécification SDL montre le diagramme des blocs de base du système et le processus SDL, ainsi qu'une brève description de la fonctionnalité.

3.4.1.2.1 Spécification mathématique

Le correcteur de type PID reçoit la vitesse d'un moteur et la compare à la consigne donnée par le moteur. Une fois que la comparaison est effectuée, il calcule alors un nouvel ordre pour le moteur. Cet ordre est proportionnel à l'erreur, proportionnel à la dérivée de l'erreur et proportionnel à l'intégrale de l'erreur. Ce correcteur est un des correcteurs fondamentaux en asservissement. La Figure 29 schématise le principe de la correction.

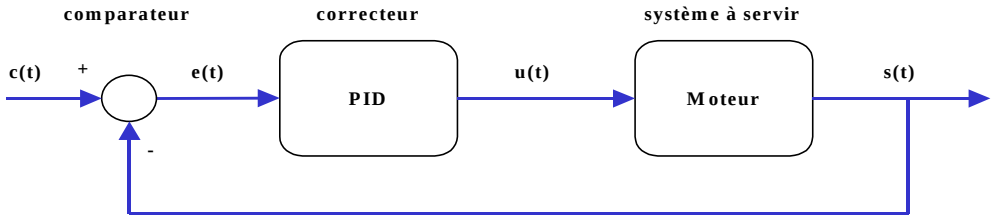

Figure 29. Principe de la correction

Données :

- c(t) : consigne
- s(t) : sortie du moteur
- e(t) : erreur ; e(t) = c(t) – s(t)
- u(t) : consigne

Equations du PID (3-1):

$$u(t) = K_p * e(t) + \frac{K_i}{\tau_i} \int e(t)dt + K_d * \tau_d * \frac{de(t)}{dt} \qquad (3\text{-}1)$$

Le terme K_p détermine les performances en terme de vitesse. Le terme K_i détermine les performances en terme d'erreur. Le terme K_d détermine les performances en terme de stabilité.

Après la transformée en *p*, on obtient la fonction de transfert du contrôleur PID donnée par l'équation (3-2) :

$$H(p) = K_p + \frac{K_i}{\tau_i * p} + K_d * \tau_d * p \qquad (3\text{-}2)$$

Après la transformée en *z*, on obtient la fonction de transfert donnée par l'équation (3-3) :

$$H(z) = K_p + \frac{K_i * T_e}{\tau_i} * \frac{z}{z-1} + \frac{K_d * \tau_d}{T_e} * \frac{z-1}{z} \qquad (3\text{-}3)$$

Enfin, on a l'équation (3-4) aux différences permettant de programmer le correcteur :

$$u[n] = u[n-1] + K_1 * e[n] + k_2 * e[n-1] + k_3 * e[n-2] \qquad (3\text{-}4)$$

En réglant les différents coefficients K_1, K_2, K_3 on obtient la correction répondant au mieux aux spécifications de l'asservissement. Un asservissement est un compromis entre la vitesse, l'erreur et la stabilité.

3.4.1.2.2 Algorithme

La vue synoptique de l'algorithme du contrôleur est présentée sur la Figure 30. Le but de cet exemple n'est pas de concevoir un système complexe, ni d'innover dans le domaine de l'asservissement, mais de valider l'outil de synthèse de la communication et les protocoles de communication.

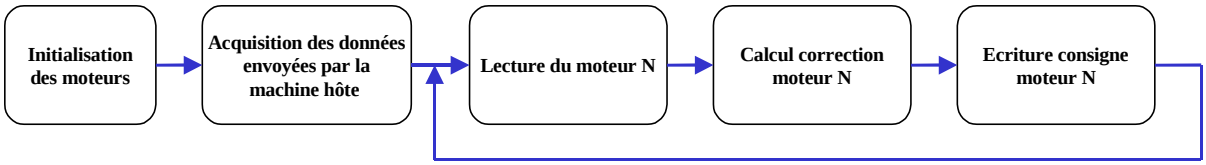

Figure 30. Algorithme du contrôleur

3.4.1.2.3 Spécification SDL

La spécification SDL du contrôleur est constituée de 3 blocs interconnectés. Le bloc *GENERATEUR* calcule une trajectoire pour l'ensemble des moteurs. La trajectoire de chaque moteur est transmise au bloc *CONTROLEUR* sous la forme d'un couple (consigne , identification du moteur). Le bloc *ECHANTILLONNAGE* reçoit les valeurs de vitesse de chaque moteur et les transmet au contrôleur. Le bloc *CONTROLEUR* est chargé de commander les moteurs, de façon à ce que tous arrivent à la position désirée dans le même espace de temps. Cette génération est basée sur l'exécution de l'algorithme du correcteur PID présenté dans la section 3.4.1.2.1. La Figure 31 présente la spécification SDL représentant le contrôleur.

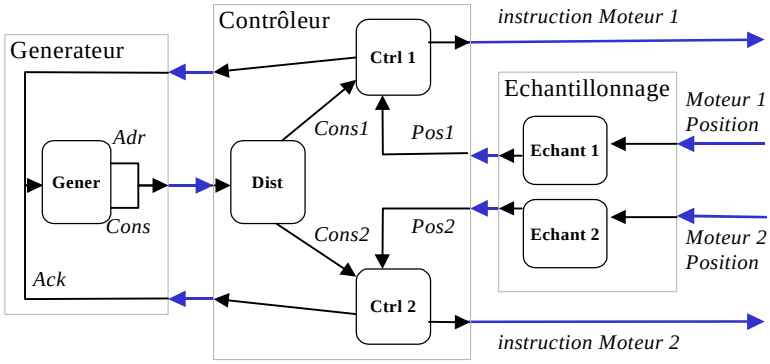

Figure 31. Représentation SDL du contrôleur

Les principaux processus de cette spécification sont :

- *Gener* : assure l'interface entre l'utilisateur et le système. Il est responsable du stockage de segments et du calcul des consignes des moteurs,
- *Dist* : reçoit des consignes et des adresses concernant le mouvement que le bras du robot doit effectuer. Il identifie le moteur qui doit tourner à la vitesse maximale et envoie des commandes en échelle pour les autres contrôleurs,
- *Ctrl 1 et 2* : gardent les informations actualisées sur la distance à parcourir pour chaque moteur à partir de la position actuelle du bras. Ce processus génère les ordres pour les moteurs,
- *Echant 1 et 2* : reçoivent les valeurs de vitesse du moteur sous forme de signaux de contrôle. Ces processus réalisent la conversion, en échantillons, des signaux continus en provenance des moteurs.

3.4.1.3 Modélisation du moteur électrique

La modélisation du moteur électrique est composée de la spécification mathématique qui décrit l'équation aux différences permettant de simuler le moteur, la vue synoptique de l'algorithme du moteur et la spécification de la fonctionnalité en Matlab.

3.4.1.3.1 Spécification mathématique

Les moteurs sont modélisés par une équation aux différences du premier ordre. Cette équation permet de simuler le comportement des moteurs.

Equation de départ (3-5) :

$$s(t) = \left[1 - \exp\left(\frac{-t}{\tau}\right)\right] u(t) \qquad (3\text{-}5)$$

Après la transformée en *p*, on obtient la fonction de transfert du moteur (3-6) :

$$H(p) = \frac{1}{1 + \tau * p} \qquad (3\text{-}6)$$

Après la transformée en *z*, on obtient la fonction de transfert (3-7) :

$$H(z) = \frac{1 - \exp\left(\frac{-T_e}{\tau}\right)}{z - \exp\left(\frac{-T_e}{\tau}\right)} \qquad (3\text{-}7)$$

Enfin on a l'équation aux différences permettant de simuler le moteur (3-8) :

$$s[n] = s[n-1] + K[[s[n-1] - u[n-1]]] \qquad (3\text{-}8)$$

3.4.1.3.2 Algorithme

La synoptique de l'algorithme des moteurs est présenté sur la Figure 32. Le moteur fait la lecture de la consigne envoyée par le contrôleur et calcule sa position et sa vitesse. Ces paramètres sont envoyés au contrôleur afin qu'il calcule la prochaine consigne pour le moteur.

Figure 32. Algorithme du moteur

3.4.1.3.3 Spécification Matlab

La spécification Matlab est composée des deux processus indépendants qui décrivent le comportement de chaque moteur électrique. Chaque processus exécute l'algorithme présenté dans la section 3.4.1.3.2. Ce modèle se comporte comme un testbench pendant toutes les étapes de validation du système. La Figure 33 présente le schéma Matlab pour un des moteurs. Matlab communique avec l'extérieur en utilisant un

protocole de communication défini par les portes *ExIn* et *ExOut* (cellule Matlab qui exécute un programme C) et, en conséquence, l'interface de communication est figée.

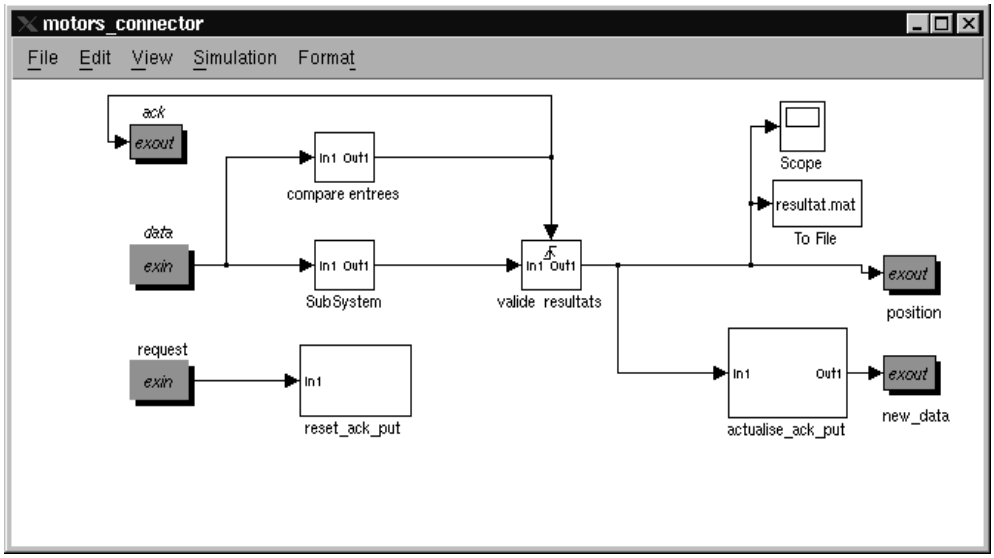

Figure 33. Spécification Matlab d'un moteur

Dans cet exemple, le moteur Matlab reçoit une nouvelle donnée au travers de la porte *data* et un signal au travers de la porte *request* qui informe de l'arrivée de la nouvelle donnée. Ce signal indique à Matlab s'il peut ou non calculer une nouvelle valeur. La porte *ack* indique si Matlab a bien reçu la nouvelle donnée. Le calcul fait, Matlab met la valeur calculée sur la porte *position* et signale sur la porte *new_data* qu'il y a une nouvelle valeur. Les portes *data*, *request* et *ack* spécifient un connecteur (type13), et les portes *position* et *new_data* un autre connecteur (type14). La Figure 34 présente la bibliothèque de définitions de connecteurs SOLAR pour cette application.

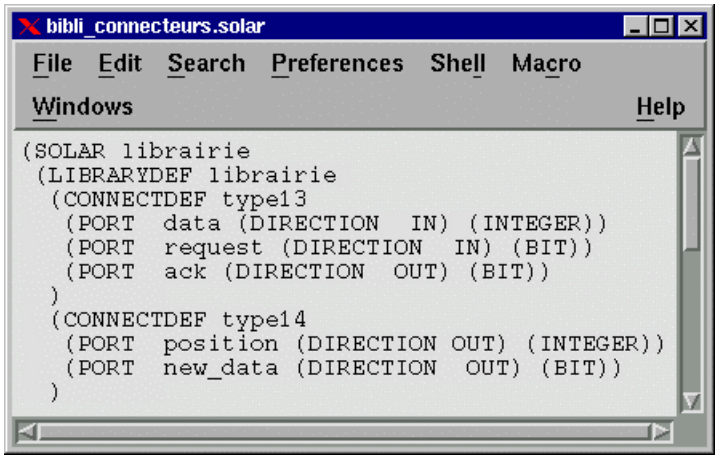

Figure 34. Bibliothèque de connecteurs pour l'exemple de moteurs

La spécification pour le deuxième moteur est identique aux caractéristiques du moteur près.

3.4.1.4 Spécification du système global

Le système complet est composé du contrôleur modélisé en SDL et de son environnement modélisé en Matlab. Le système global est décrit par le biais d'un fichier de configuration précisant les interconnections des signaux entre les deux modèles. Le format intermédiaire SOLAR [48] est utilisé

pour la description de la configuration. SOLAR permet la spécification de systèmes mixtes logiciel/matériel à plusieurs niveaux d'abstraction.

3.4.1.4.1 Spécification hétérogène dans le format SOLAR

Un système hétérogène est décrit en SOLAR comme un ensemble de sous-systèmes interconnectés via des canaux de communication abstraits. Les sous-systèmes sont représentés comme des boîtes noires. Pour chaque sous-système est spécifiée l'interface qui désigne les signaux entrant et sortant du sous-système, ainsi que le type et la direction de chaque signal. Si un ensemble de signaux représente une fonctionnalité spécifique, comme par exemple un port série, les signaux seront spécifiés dans l'interface comme un connecteur (voir section 3.3). Le fichier de description du comportement du sous-système est indiqué au travers d'un lien. Les interconnexions entre les sous-systèmes sont représentées par une liste de haut niveau qui spécifie les échanges de données. Cette liste est représentée en SOLAR comme des canaux de communication abstraits. Les protocoles de communications qui coordonnent les échanges de données seront explicités au moment de la synthèse des communications. A ce niveau d'abstraction, SOLAR est utilisé comme un langage d'interconnexion décrivant une représentation unifiée du système.

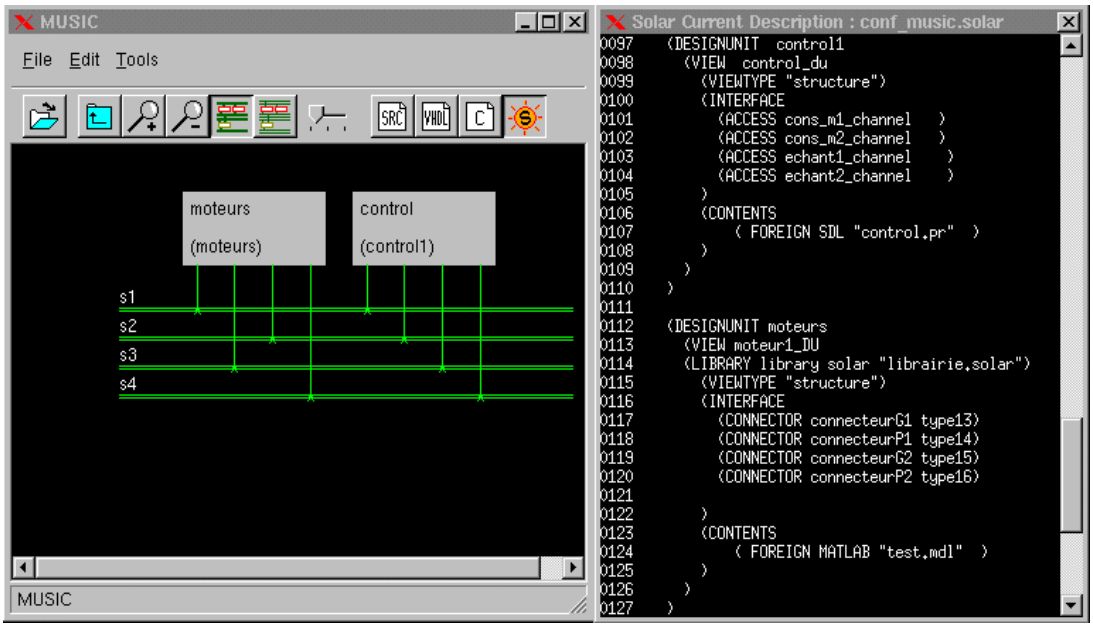

Figure 35. Spécification hétérogène SDL-Matlab

La Figure 35 montre la représentation graphique et textuelle du fichier de configuration utilisé pour le contrôleur. Il est composé de deux sous-systèmes : un sous-système en SDL, nommé « control », composé de six processus parallèles (Figure 31) et un sous-système en Matlab, nommé « moteurs », composé de deux processus qui modélisent le comportement des deux moteurs. Les deux sous-systèmes communiquent au travers de quatre canaux de communication abstraits hétérogènes (s1, s2, s3, s4). L'interface du sous-système SDL est spécifiée par des accès accessibles par des primitives de haut niveau, et l'interface du sous-système Matlab est décrite par des connecteurs qui spécifient un protocole de communication figé (connecteur G1 et P1 pour le premier moteur et connecteur G2 et P2 pour le deuxième moteur). Le sous-système SDL sera raffiné en une architecture C-VHDL, ainsi que les communications entre les processus SDL (communications internes) et les communications entre les processus SDL et le sous-système Matlab seront raffinées du niveau d'abstraction système vers le niveau d'abstraction architecture. Aucun raffinement ne sera appliqué au sous-système Matlab qui est spécifié comme un module composant existant [80].

3.4.2 Synthèse des communications du système de contrôle de moteurs

La synthèse des communications transforme un système composé de processus qui communiquent via des canaux abstraits en un système composé de processus interconnectés qui communiquent via des signaux [21]. Cette transformation est réalisée en se basant sur le fichier de configuration SOLAR qui décrit le système global (Figure 35) et sur une bibliothèque de communications composée d'un ensemble de protocoles de communications fournie par l'utilisateur. L'environnement MUSIC [17] assiste l'utilisateur pour assigner à chaque canal abstrait un protocole de communication concret.

3.4.2.1 Protocoles utilisés pour la synthèse des communications

Dans l'exemple du contrôleur, les canaux abstraits sont réalisés en utilisant deux protocoles de communication disponibles dans la bibliothèque de communication fournie par l'utilisateur. Les communications entre les processus du contrôleur (communications internes) sont transformées par l'application d'un protocole du type rendez-vous et les communications entre les moteurs et les processus du contrôleur sont transformées par l'application d'un protocole de communication appelé hétérogène. Le choix du type de protocole est adapté aux besoins de la communication et doit prendre en considération les performances nécessaires et le coût de la réalisation.

3.4.2.1.1 Protocole de communication hétérogène (SDL-Matlab)

Le protocole de communication hétérogène est utilisé pour réaliser les canaux de communication abstraits qui spécifient les communications entre les processus du contrôleur et les moteurs. La caractéristique de ce type de canal est d'interconnecter des processus qui ont des interfaces de communication différentes. Un contrôleur est utilisé pour connecter les différents processus et pour adapter les différentes interfaces de communications. Ainsi, le contrôleur décrit la logique nécessaire pour mettre en œuvre la communication. Dans cet exemple, la logique décrite par le contrôleur est simple. En effet, les interfaces de communication sont compatibles et les processus communiquent au travers d'un rendez-vous point-à-point. Une méthodologie pour décrire la logique dans le cas où les interfaces entre les processus seraient incompatibles est présentée par [77].

3.4.2.2 Transformation du modèle hétérogène SDL-Matlab en une architecture C-VHDL-Matlab

La spécification du système au niveau système est composée de deux modules. Un module SDL qui représente le contrôleur et un module Matlab qui représente l'environnement (moteurs). L'objectif est de réaliser les étapes de conception et de synthèse système pour générer une description architecturale mixte composée de sous-systèmes matériels (algorithmes VHDL), de sous-systèmes logiciels (algorithmes C) et de sous-systèmes de communication (pouvant être logiciels ou matériels) afin d'obtenir un prototype du système. Toutes les étapes de transformation sont basées sur la forme intermédiaire SOLAR, qui représente à la fois les concepts du niveau système et les concepts utilisés par les langages de description logiciels ou matériels. Elles sont réalisées à l'aide de l'environnement MUSIC.

3.4.2.2.1 Raffinement du module SDL en une architecture C-VHDL

La première étape de raffinement consiste à traduire le sous-système SDL en SOLAR. Cette étape est réalisée à l'aide de la primitive **expand.** L'**expand** fait la conversion du sous-système SDL, indiqué par le lien du fichier de configuration (voir section 3.4.1.4.1), dans le format intermédiaire SOLAR. Pendant cette conversion, les opérations suivantes sont réalisées [21] : les processus sont transformés en unités de conceptions, le comportement des processus est transformé en tables d'états et les interconnexions entre les processus sont transformées en canaux abstraits. La description SOLAR générée remplace la spécification initiale du sous-système SDL dans le fichier de configuration ainsi que les interconnexions

entre les processus SDL et le sous-système Matlab qui sont actualisées en fonction de la description SOLAR générée.

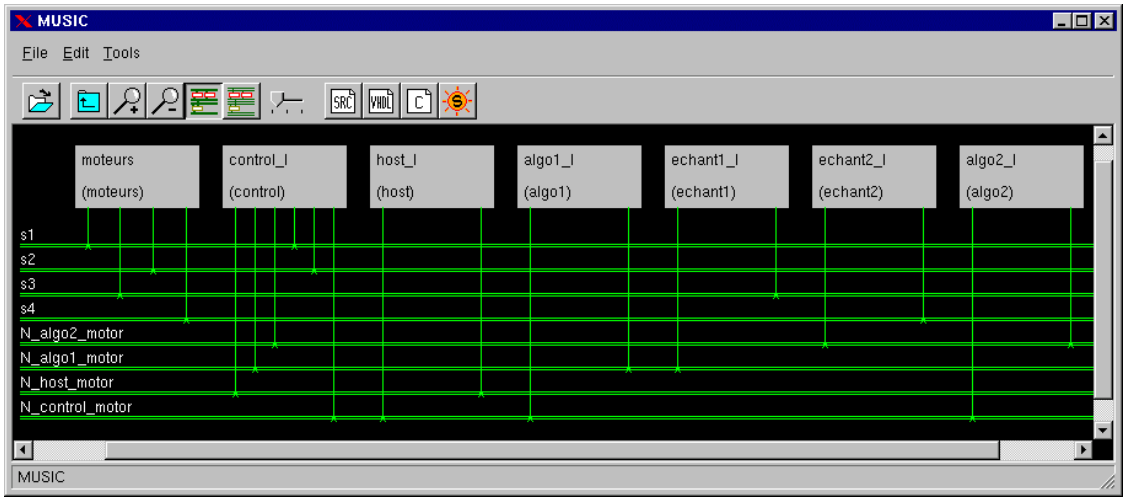

Figure 36. Spécification du système mis à plat

La prochaine étape de raffinement consiste à mettre à plat la structure hiérarchique des unités du système. Cette opération facilite la visualisation et le traitement des communications par étapes de raffinement des communications. La spécification obtenue après cette étape de raffinement est présentée par la Figure 36, elle est composée de sept unités de conception, six proviennent du modèle SDL et la septième contient les moteurs Matlab. Les six blocs constituent le système en cours de conception et le septième l'environnement.

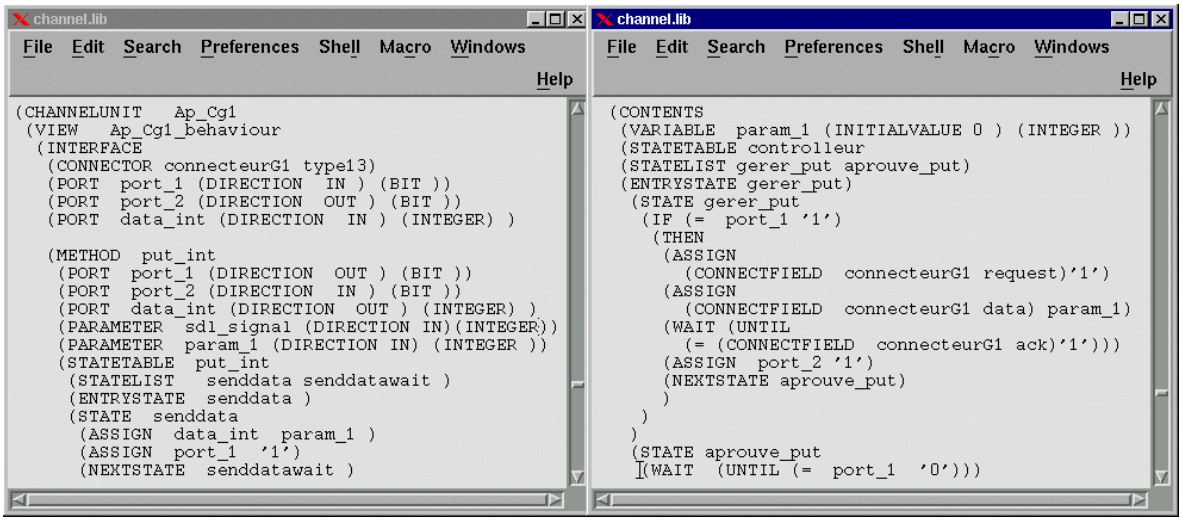

Figure 37. Bibliothèque de protocoles de communication

La transformation structurelle appliquée, la primitive **map hétérogène** est utilisée pour remplacer les canaux abstraits par le protocole choisi dans la bibliothèque de communication et pour générer les signaux et l'interface correspondante. Les protocoles rendez-vous et rendez-vous multiple sont utilisés pour réaliser les canaux qui interconnectent les unités de conception en provenance du modèle SDL et le protocole hétérogène est utilisé pour réaliser les canaux qui interconnectent l'environnement Matlab et le système en cours de conception. La Figure 37 présente la spécification du protocole de communication hétérogène (Ap_Cg1) utilisé pour réaliser le canal abstrait *s1*. Ce canal est responsable pour la

transmission d'une donnée du modèle SDL pour le modèle Matlab. Le protocole de communication est composé d'une méthode responsable pour la réalisation de la connexion logique (SDL) et d'un connecteur responsable pour l'interconnexion du contrôleur de la communication avec le modèle Matlab (voir section 4.3.1.4).

La Figure 38 montre le résultat de la primitive **map hétérogène.** Sept nouvelles unités ont été automatiquement insérées, elles correspondent aux connexions nécessitant un contrôle externe pour mettre en œuvre la communication. Trois correspondent au contrôle du protocole rendez-vous multiple (les trois dernières) et quatre correspondent au protocole hétérogène (les quatre premières). Le fichier SOLAR textuel présente le résultat des modifications qui sont réalisées pour la primitive **map hétérogène** sur les connecteurs. Dans ce cas, les connecteurs G1, P1, G2, P2 sont remplacés pour les ports définis dans la bibliothèque de connecteurs.

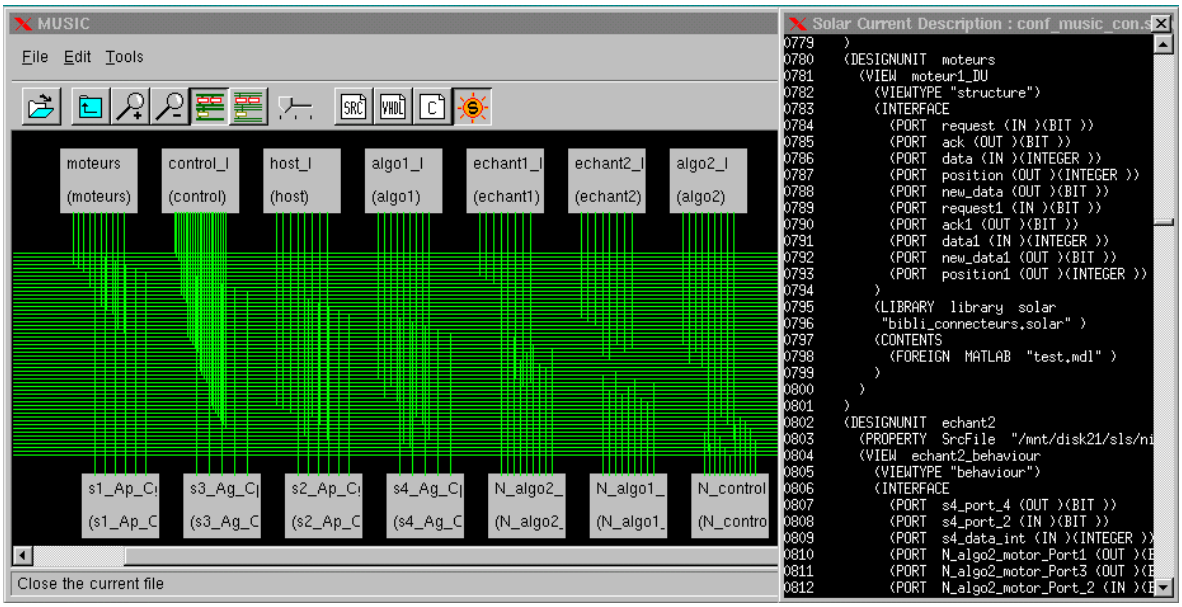

Figure 38. Spécification au niveau architecture

L'étape suivante consiste à sélectionner les unités de conception destinées à une réalisation logicielle et à une réalisation matérielle. Ainsi, le système sera composé des parties matérielles et logicielles et des composants existants. Suite à ces transformations, MUSIC peut générer le code C-VHDL pour chaque processus, sauf pour le sous-système composant existant. Le comportement des processus logiciels est traduit en langage C, et le comportement des processus matériels en VHDL. Le modèle obtenu est composé des modules C, VHDL et Matlab, et peut être utilisé pour une validation de l'architecture. L'étape suivante est le raffinement de la spécification du niveau architecture vers le niveau cycle. A ce niveau, les parties matérielles sont raffinées dans un modèle RTL, les parties logicielles sont exécutées sur un modèle du processeur et le composant existant en Matlab. L'étape finale est la mise en œuvre du modèle sur un prototype réel. A ce niveau, le prototype de la partie électronique a besoin d'être construit et le composant existant Matlab a besoin d'être émulé.

3.4.2.2.2 Raffinement des communications

Le raffinement des communications, dans cet exemple, consiste à transformer un système multilangage spécifié au niveau système et communiquant au travers de primitives de haut niveau en un ensemble de processus spécifiés au niveau architecture qui communiquent au travers de signaux. Ce raffinement est réalisé à l'aide de la primitive **map hétérogène** en se basant sur la bibliothèque de protocoles de communication fournie par le concepteur. Cette primitive affecte un protocole de communication de la

bibliothèque à chaque canal de communication de haut niveau et génère les interfaces entre les unités interconnectées.

Au niveau système, les interconnexions sont décrites, au niveau d'une coordination de type application, par des canaux abstraits accessibles par des primitives de haut niveau. Au niveau architecture, les primitives de haut niveau sont transformées en appels de procédure qui sont des services fournis par le canal abstrait. La dernière étape de raffinement consiste à remplacer les appels de procédure par le code équivalent et le canal abstrait par des fils.

3.4.3 Covalidation du système

La validation de la spécification aux niveaux système et architecture du contrôleur s'effectue par cosimulation [50], à l'aide de l'environnement de cosimulation MCI [43], [67]. La solution de cosimulation permet la connexion des simulateurs propres à chaque langage et permet ainsi de profiter des possibilités de chacun d'eux. L'environnement MCI utilise le fichier de configuration présenté dans la section 3.4.1.4.1 afin d'exécuter une cosimulation où les différents sous-systèmes sont simulés de manière concurrente et répartie sur des machines distantes.

3.4.3.1 Covalidation du système au niveau système

La spécification du système contient deux sous-systèmes : l'environnement (Matlab) et le contrôleur (SDL). Afin de valider la spécification globale du système, MCI est utilisé pour réaliser la cosimulation SDL-Matlab. La Figure 39 montre une cosimulation en cours d'exécution du système SDL-Matlab. Les débogueurs et les interfaces graphiques des simulateurs permettent d'interagir avec chaque sous-système puis d'analyser les résultats.

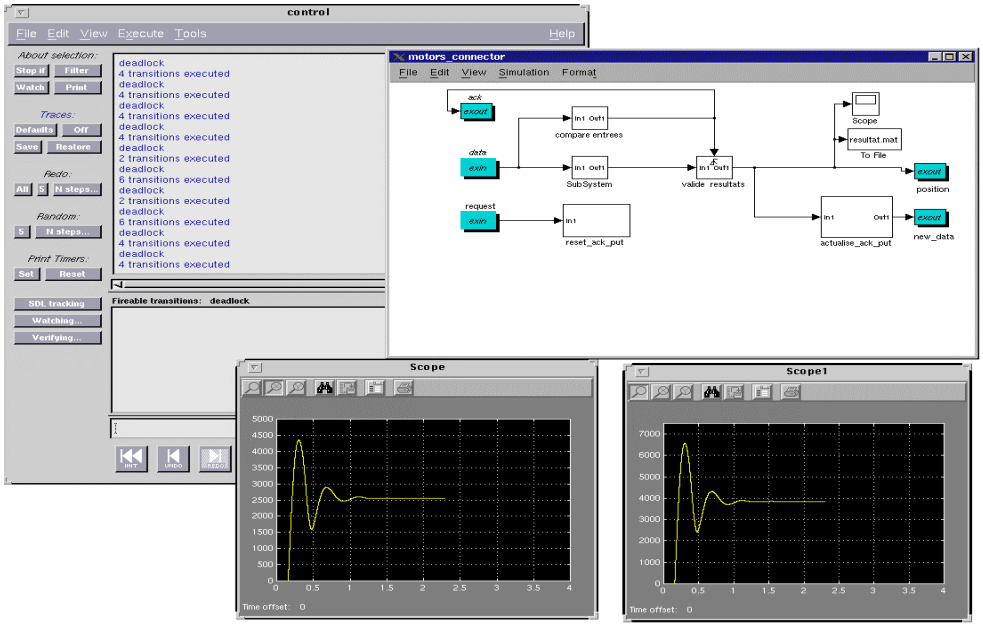

Figure 39. Cosimulation SDL-Matlab

La partie gauche de la figure montre le simulateur SDL (ObjectGeode) et la partie droite le simulateur Matlab (Simulink). Les graphiques montrent l'avancement de la vitesse des moteurs.

La cosimulation au niveau système permet la validation très tôt du système et d'ajuster les paramètres du système afin de fixer la spécification finale. Après avoir réalisé la validation du système global au niveau

système, les étapes de raffinement peuvent être appliquées sur la partie électronique du système (voir section 3.4.2.2.1).

3.4.3.2 Covalidation du système au niveau architecture

Afin d'obtenir un prototype, plusieurs étapes de raffinements de la spécification SDL-Matlab sont nécessaires (voir section 3.4.2.2). Le résultat obtenu est une spécification au niveau architecture composée de modules C, VHDL, Matlab et d'un fichier de configuration, qui spécifie les interconnexions entre les modules et qui est utilisé pour la cosimulation. Le nouveau modèle ainsi généré est utilisé pour une validation au niveau architecture utilisant MCI. La Figure 40 montre la cosimulation C-VHDL-Matlab en cours d'exécution.

La partie supérieure de la figure montre le simulateur VHDL (Synopsys) et la partie inférieure le simulateur Matlab (Simulink). L'exécution des modules C n'est pas visible puisque n'ayant pas d'interface graphique. Les deux graphiques montrent les résultats de l'avancement de la vitesse des moteurs. A ce niveau, la cosimulation permet la validation des étapes de raffinements, comme le découpage logiciel/matériel, la synthèse des communications, les protocoles de communication et quelques vérifications temporelles.

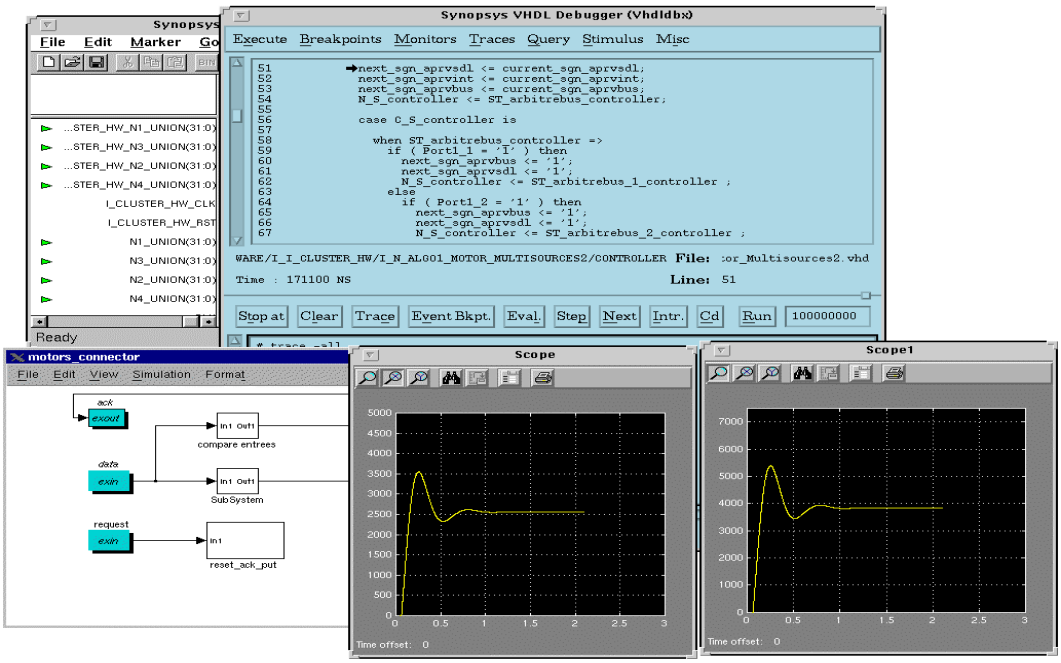

Figure 40. Cosimulation C-VHDL-Matlab

3.4.3.3 Covalidation du système au niveau cycle

La prochaine étape consiste à raffiner la spécification au niveau cycle. La partie matérielle est raffinée dans un modèle RTL, la partie logicielle est exécutée dans le modèle du processeur et la partie mécanique peut être exécutée sur le simulateur Matlab. A ce niveau, toutes les interfaces sont raffinées aux niveaux physique et drivers. Quelques adaptateurs sont nécessaires pour réaliser la liaison entre le processeur et l'application. La cosimulation peut être utilisée pour valider au niveau cycle le système. La Figure 41 montre la cosimulation au niveau cycle en cours d'exécution. Le processeur ST10 a été utilisé pour la partie logicielle, le simulateur VHDL (Synopsys) pour la partie matérielle et le simulateur Matlab (Simulink) pour la partie mécanique.

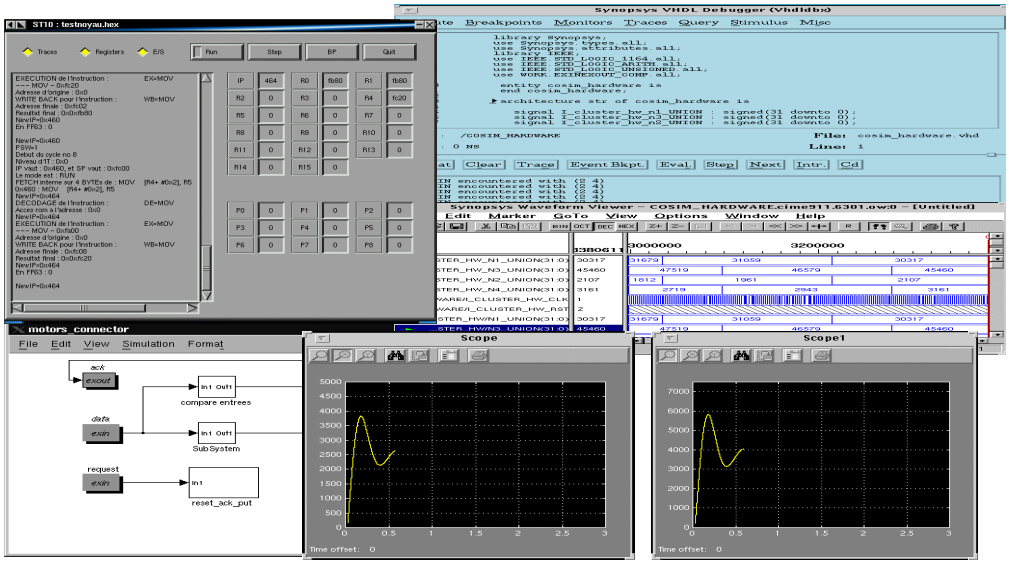

Figure 41. Cosimulation au niveau cycle

3.4.4 Résultats

L'étape suivante consiste à analyser les résultats de la synthèse de la communication obtenus par la cosimulation au niveau système, au niveau architectural et au niveau cycle. La Figure 42 montre un graphique qui compare les résultats de la cosimulation au niveau système, au niveau architecture et au niveau cycle pour cet exemple.

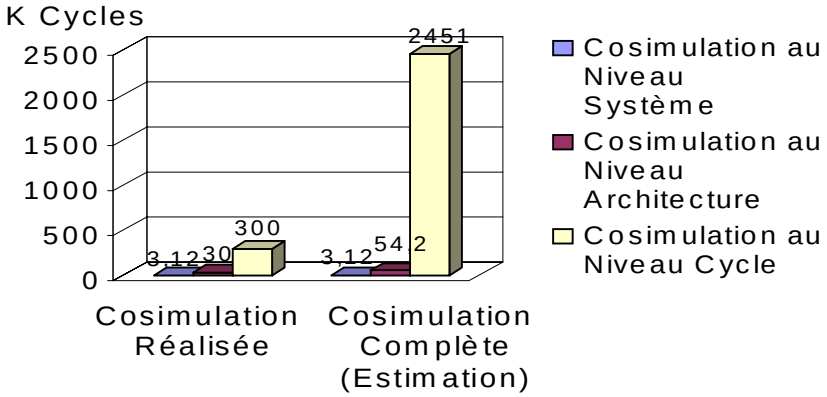

Figure 42. Résultats de la Cosimulation

Au niveau système, le pas de simulation correspond à la transaction entre les processus parallèles, et la simulation complète du système est réalisée en environ deux minutes. Au niveau architecture, le pas de simulation représente le pas de calcul d'une description en VHDL comportemental, et la simulation complète du système est réalisée en environ une heure et dix-sept minutes. Au niveau cycle, le pas de simulation correspond à un cycle d'horloge et la simulation d'une petite partie du système est réalisée en environ quatre heures. La simulation complète du système au niveau cycle est estimée à environ trente heures et vingt minutes.

La Figure 43 montre les résultats de la synthèse de la communication hétérogène. Ces résultats sont donnés en fonction de l'augmentation de la taille du code entre une spécification au niveau système et sa

réalisation au niveau architecture. Ces résultats sont fortement liés au type de protocole de communication utilisé, au type de spécification du protocole de communication et à l'application elle-même.

Pour l'application du contrôleur du bras du robot, la communication au niveau système représente 5% de la spécification (SDL) et 23% de la spécification au niveau architecture. Cela est dû au fait que SDL utilise seulement une instruction pour représenter la communication. Le protocole de communication, l'acheminement du signal et la file d'attente sont implicites. Lors de la réalisation de la spécification, la communication devenant explicite requière un protocole spécifique, un contrôleur et des bus. La taille du code du modèle au niveau architecture est environ 962% celle du modèle au niveau système.

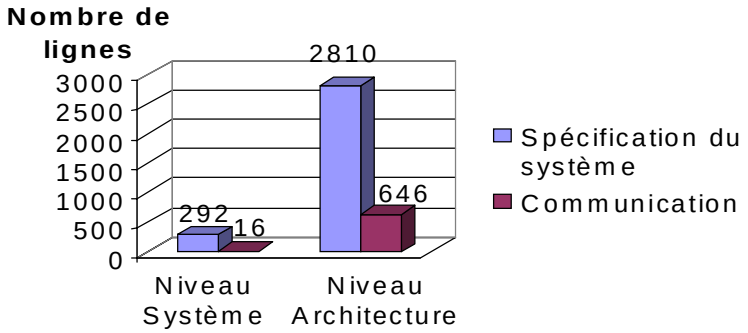

Figure 43. Résultats de la synthèse de la communication

3.5 Conclusion

Dans ce chapitre a été présentée une méthodologie de synthèse de communication pour les spécifications de systèmes multilangages. La méthodologie de synthèse proposée peut être exécutée automatiquement et permet au concepteur de transformer les primitives de communication de haut niveau, décrites dans différents formalismes et notations, en une réalisation physique. Aucune restriction n'étant imposée aux modèles de communication et aux langages de description, ce procédé peut être appliqué à une large classe d'applications multilangages.

Cette approche a été illustrée sur l'exemple du système de contrôle de moteurs. La spécification initiale de la partie électronique du contrôleur a été réalisée en SDL et la spécification de la partie mécanique en Matlab. La communication de la partie électronique est réalisée par des primitives de communication de haut niveau, et son interface est définie au moment de l'attribution d'un protocole de communication. La partie mécanique est spécifiée comme un module existant qui par définition a une interface figée et un protocole de communication défini, spécifiés au travers de connecteurs.

L'approche proposée dans ce chapitre, en contraste avec d'autres approches de synthèse de communication, apporte des solutions de synthèse des communications pour les systèmes hétérogènes multilangages.

Chapitre 4

COSIMULATION GEOGRAPHIQUEMENT DISTRIBUEE

Ce chapitre présente l'environnement de validation MCI développé en commun avec Ph. Le Marrec et Fabiano Hessel. Un état de l'art est dressé et une étude des spécificités dues à la distribution est présentée. Le prototype MCI et ses extensions sont détaillés. Enfin, une expérience menée en collaboration avec STmicroeletronic, le CNET et AREXSYS est présentée.

4.1 Introduction

Les systèmes actuels complexes font appel aux compétences de plusieurs équipes de développement. Chaque équipe de conception emploie son propre langage et environnement de conception. La complexité entraîne l'utilisation de modèles abstraits afin d'obtenir une vue globale du système à concevoir. Il en résulte que la conception d'un système est modulaire, multilangage et réalisée à partir d'une spécification abstraite et avec des outils de conception spécifiques. D'autre part, ces équipes peuvent être géographiquement éloignées. Des exemples de tels systèmes peuvent être un lecteur de DVD, un téléphone mobile ou encore un téléviseur numérique.

Le principal problème soulevé par cette approche de conception est qu'il est difficile de s'assurer du fonctionnement correct du système avant la construction du premier prototype. Une solution est la validation globale du système très tôt dans le planning de conception qui permet d'optimiser les coûts de conception en évitant les boucles prototype - essai – recherche de l'erreur très coûteuses. Une telle validation pose les problèmes de puissance de calcul, de disponibilité des licences de chacun des simulateurs nécessaires et des problèmes de confidentialité des modules en conception. La simulation d'un système composé de modules décrits dans des langages différents est appelé cosimulation.

Les premiers environnements de cosimulation sont apparus à la suite des environnements de conception conjointe dans le but de faciliter la validation des systèmes mixtes (VHDL/C) ainsi conçus. L'étude et la généralisation à tout type de langage de ces environnements permet d'appréhender la validation globale de tels systèmes. La cosimulation permet de simuler conjointement les différents modules qui composent le système. La répartition géographique permet de résoudre les problèmes de puissance de calcul, de disponibilité des simulateurs et de confidentialité. Plusieurs thèses du groupe SLS ont abordé les techniques de cosimulation [67], [43], [106]. Cette thèse porte sur l'utilisation et l'adaptation de la cosimulation aux environnements géographiquement distribués.

Ce chapitre s'articule autour de huit sections, la seconde présente l'état de l'art dans le domaine de la cosimulation ainsi que les principaux outils du marché. La section suivante présente une synthèse des solutions employées dans les approches de cosimulation locale et géographiquement distribuée. La quatrième section présente le prototype MCI issu du laboratoire TIMA-SLS. La cinquième section dépeint les extensions de prototype MCI réalisées dans le cadre de ce travail. La section suivante présente des exemples d'application du nouvel environnement MCI. La septième section présente un travail d'application sur un projet industriel en collaboration avec AREXSYS, CNET(FT) et STmicroelectronic. La dernière section présente les conclusions.

4.2 Etat de l'art de la cosimulation

Le principe de la cosimulation multilangage est l'exécution en parallèle des simulateurs nécessaires pour un système. Chaque simulateur exécute un module du système spécifié dans un langage spécifique au domaine traité. Les modules ou sous-systèmes peuvent appartenir à différents modèles de calcul (orienté contrôle/données, synchrone/asynchrone, discret/continu). De plus, les sous-systèmes peuvent être spécifiés à différents niveaux d'abstraction. Deux grands axes caractérisent le mode d'exécution d'un environnement de cosimulation : le modèle de synchronisation et le modèle temporel utilisés.

4.2.1 Modèles de synchronisation

La communication entre les différents simulateurs est un point essentiel dans la définition d'un environnement de cosimulation. Les simulateurs sont exécutés en parallèle et l'environnement de cosimulation doit assurer la cohérence des données (c.-à-d. pas d'inversion ou d'écrasements de données). Ainsi, il est nécessaire que l'environnement de cosimulation dispose d'un modèle de synchronisation fiable et performant en vitesse afin de garantir la correction du système. La synchronisation entre les

simulateurs peut être réalisée selon deux modèles : le modèle maître – esclave et le modèle distribué [109], [74].

4.2.1.1 Modèle maître / esclaves

L'architecture maître / esclaves consiste à désigner un simulateur comme maître de la simulation. Il a la charge d'invoquer les autres simulateurs lors de l'activation de modules externes. Cette méthode a l'avantage d'être facile à mettre en œuvre. Les simulateurs commerciaux fournissent une bibliothèque de fonctions permettant l'échange de données avec l'extérieur.

Cependant, elle contraint l'architecture du système aux systèmes du type maître / esclaves, formés d'un processeur maître auquel sont associés des coprocesseurs ou circuits programmables esclaves. Ce modèle de synchronisation élimine toute forme de parallélisme. Par analogie, un ensemble de programmes parallèles est transformé en un programme principal séquentiel appelant des sous programmes.

Ce modèle est bien adapté aux applications orientées données mais n'est pas adapté aux applications orientées contrôle. Dans le cas d'un système composé d'une partie matérielle et d'une partie logicielle, ce modèle exige que la partie logicielle soit découpée en « pas » de calcul dont le temps d'exécution est déterminé statiquement. Cette contrainte fondamentale est nécessaire afin d'éviter les interblocages dus aux boucles intermédiaires.

4.2.1.2 Modèle distribué

L'architecture distribuée permet aux simulateurs de s'exécuter de manière concurrente. Chaque simulateur peut envoyer ou recevoir des données sans contraintes. Les protocoles de communications sont chargés de la résolution des conflits. Cette méthode présente l'avantage de ne pas contraindre l'architecture du système et ainsi proposer un large champ d'applications. Cependant la cohérence entre les données partagées par les simulateurs est plus difficile à assurer.

La principale approche employée pour ce modèle consiste à connecter chaque simulateur à un bus de cosimulation (bus logiciel) chargé de l'échange des données et des synchronisations nécessaires. Ce bus agit comme un serveur de communication.

Ce modèle a plusieurs avantages comparé au modèle maître / esclaves. Tout d'abord il permet l'exécution parallèle et concurrente de plusieurs simulateurs. En outre, les différents sous-systèmes peuvent être conçus de façon concurrente en utilisant différents outils et méthodes de conception. Il permet la simulation de différents sous-systèmes à plusieurs niveaux d'abstraction au cours du processus de conception tout en utilisant le même banc d'essai.

Néanmoins, les inconvénients de ce modèle sont la complexité du bus de cosimulation, plus particulièrement, la synchronisation entre des simulateurs à différents niveaux d'abstraction et la performance du bus de cosimulation.

4.2.2 Modèles temporels

La cosimulation d'un système hétérogène multilangage est basée sur l'exécution de plusieurs simulateurs communiquants. Chaque simulateur a une notion différente du temps et une vitesse de simulation propre. Selon le degré d'abstraction voulu, deux approches existent : une validation fonctionnelle, sans modélisation du temps, et une approche temporelle, avec modélisation du temps et synchronisation des différents acteurs de la cosimulation [39].

4.2.2.1 Validation fonctionnelle

La validation fonctionnelle a pour but de vérifier la fonctionnalité d'un système. Pour cela, seules les données échangées entre les simulateurs sont considérées, l'outil de Cosimulation a la charge de transporter les données d'un simulateur vers l'autre.

La validation fonctionnelle consiste à vérifier la fonctionnalité d'un système sans synchronisation temporelle des simulateurs pendant l'exécution de la cosimulation et sans prise en compte du temps de propagation. Ce type de validation échange les données dans l'ordre de leurs modifications. Il est possible de perdre des données si un simulateur était plus rapide que les autres et écrasait des données non lues. Ce problème peut être résolu par un protocole de communication bloquant. Dans ce cas, le simulateur envoie une donnée et reste bloqué jusqu'à ce que le simulateur destinataire ait lu cette donnée. Une variante de cette solution est l'utilisation d'une mémoire intermédiaire pour stocker les valeurs envoyées (une file d'attente).

La fidélité de la validation fonctionnelle est donnée par la précision utilisée dans la modélisation de la relation valeur/temps des variables et des signaux. Ainsi, le temps avance pour tous les simulateurs indépendamment et la relation valeur/temps n'est vérifiée qu'à certains intervalles de temps [39].

4.2.2.2 Validation temporelle

La validation temporelle permet de vérifier de façon plus réaliste un système en considérant le temps et consiste à échanger les données à des instants précis dans le temps (échantillonnage temporel de données) [39]. Ainsi les différents simulateurs sont synchronisés et possèdent tous la même date de simulation. Ce type de Cosimulation permet d'évaluer les performances d'un système et de vérifier les contraintes temporelles du système.

La Cosimulation temporelle est d'une plus grande complexité que la cosimulation fonctionnelle. La solution est différente selon la méthode de Cosimulation employée :

- Lorsqu'un simulateur maître est chargé de l'activation des modules, la solution n'est qu'une extension de la méthode. En effet, le maître supporte alors la gestion du temps. Il gère l'horloge globale du système et contrôle facilement l'avancement de chaque simulateur.
- L'intégration du temps dans une architecture distribuée est une tâche sensiblement plus difficile. La synchronisation des simulateurs passe par la mise en place d'une horloge globale. Il s'agit de synchroniser l'avancement dans le temps de chacune des simulations sans dégrader la précision et les performances par un coût de synchronisation trop élevé.

Le modèle « lock-step », autrement appelé barrière de synchronisation, est une solution de validation temporelle pour l'architecture distribuée. Une horloge globale est gérée par un module externe aux simulateurs, parfois nommé « serveur de temps ». Le temps est, comme dans tout modèle de simulation, rendu discret par une division en segments, appelés « pas ». Les différents simulateurs sont indépendants et possèdent une longueur (durée) de « pas » de simulation propre. Le « pas » de l'horloge globale est inférieur au « pas » le plus petit des simulateurs présents et doit être un diviseur commun à tous les « pas ». Un simulateur est libre d'effectuer un « pas » s'il ne dépasse pas la date globale courante. L'horloge globale avance d'un « pas » lorsque tous les simulateurs ont atteint la date globale courante. Cette méthode est une solution de Cosimulation non contraignante pour l'architecture du système et facile à mettre en œuvre. Le surcoût dû à ces étapes de synchronisation est important pour obtenir une précision importante. Toutefois il est acceptable si la différence entre les « pas » de chaque simulateur n'est pas trop importante.

Tous les modules d'un système ne possèdent pas de notion de temps, certains niveaux d'abstraction d'un module interdisent une validation temporelle juste. Un module logiciel décrit en langage C ne possède pas d'informations sur son temps d'exécution. Une solution pour la simulation temporelle est de s'appuyer sur

un modèle du processeur utilisé. Un modèle de processeur est un programme ou un module HDL simulant le fonctionnement du processeur. Il existe plusieurs type de modèles de processeurs. Le niveau de précision peut varier de l'instruction, au cycle ou aller jusqu'au niveau de la porte logique.

Ce type de validation est très utilisé pour la validation à très bas niveau où il est nécessaire d'avoir une simulation plus précise.

4.2.3 Outils existants

De nombreux outils de cosimulation matérielle/logicielle existent, aussi bien sur le marché que dans le monde universitaire [6], [38], [26], [51], [103], [12], [56], [18], [107]. Les environnements complets de conception conjointe disposent également de moyens de cosimulation. Deux principales approches se détachent : une architecture cible fixe ou libre.

Les approches basées sur une architecture cible fixe imposent un modèle de communication et permettent l'emploi de simulateurs de processeur. Il est ainsi possible d'obtenir des simulations très précises. Le principal inconvénient est l'emploi d'une architecture peu flexible.

Les approches non basées sur une architecture cible fixe n'imposent ni de modèle de communication, ni de processeurs spécifiques. Il est alors difficile de proposer une simulation très précise par l'emploi de simulateurs de processeurs et de modèles de structures de communication. L'intérêt d'une telle approche est la flexibilité apportée et la possibilité de s'appliquer aux méthodes de conception employées jusqu'alors. Le principal inconvénient est le manque de précision de simulation dû au manque d'informations temporelles. La cosimulation distribuée multilangage est une approche générique. La méthodologie consiste à exécuter plusieurs simulateurs en parallèle à diverses étapes du processus de conception conjointe sans se préoccuper du modèle de communication. Les détails des communications sont introduits par raffinement tout au long de la synthèse du système. Cette approche est une solution émergente et les environnements de cosimulation multilangage sont encore rares.

La suite de cette section présente quelques environnements de cosimulation.

4.2.3.1 Cosyma

Cosyma [27], [42], [82] est un environnement de conception conjointe de systèmes mixtes logiciel / matériel. La spécification initiale est réalisée en langage Cx, une extension du langage C permettant de décrire des contraintes temporelles, de débits d'information et de parallélisme entre les processus du système.

L'environnement permet de raffiner la description initialement purement logicielle en une architecture mixte. Pour cela, les parties critiques du système sont repérées puis transformées en partie matérielle. Les nouvelles parties matérielles utilisent le langage HardwareC [57], [37] et les parties logicielles, le langage C. L'architecture du système est contrainte à un processeur associé à un circuit intégré spécialisé (ASIC) et une mémoire, un bus assurant toutes les communications. L'environnement fourni un outil de cosimulation fonctionnelle répartie permettant de valider le système à chaque niveau de spécification. Un simulateur du processeur et du bus de l'architecture permet d'obtenir une simulation à grain très fin. La finesse de simulation interdit une simulation rapide du système et le système est contraint à l'architecture proposée.

4.2.3.2 Ptolemy II

Ptolemy II [83], [86] est un environnement de simulation et de modélisation de systèmes hétérogènes issu de l'université de Berkeley. L'environnement permet la description de systèmes embarqués et composés de parties très différentes (traitement du signal, dispositifs mécaniques ou électroniques, environnement physique du système, etc.).

Pour cela, Ptolemy II dispose d'un modèle orienté objet basé sur Java proposant plusieurs modèles de calcul (continu, discret, machine d'états, flot de données, …) appelés « domaines ». Cet environnement peut être qualifié d'outil de conception conjointe car il est possible de raffiner la spécification jusqu'à obtenir une description très fine du système. La simulation est une validation fonctionnelle pouvant être distribuée. Chaque module basé sur un « domaine » est concurrent des autres et est exécuté au travers de « threads » Java. La distribution sur un réseau du système s'effectue par l'emploi d'un environnement distribué tel que CORBA ou Java RMI. Ptolemy II permet de modéliser et simuler un système hétérogène complexe mais les outils de raffinement sont rares.

4.2.3.3 MCSE

L'approche MCSE (Méthodologie de Conception des Systèmes Electroniques) définit une méthodologie complète qui couvre les étapes de développement de systèmes [13], [14]. La spécification initiale est composée de spécifications fonctionnelles, exprimées dans un langage graphique basé sur le modèle de processus communicants, et de spécifications non fonctionnelles exprimées sous forme de contraintes (temps, performances, technologie, etc.).

Les estimations de performances recueillies à partir du modèle de performance de MCSE permettent la sélection des parties logicielles et matérielles. Les simulations VHDL et C++ permettent d'assister le découpage matériel/logiciel. Le modèle, dit non interprété, est formé de deux parties : la structure et le comportement. Le modèle structurel résulte de la composition d'une structure fonctionnelle et de l'architecture matérielle donnée par le processus d'allocation. Le modèle comportemental de chaque fonction est une abstraction du comportement algorithmique. Ce modèle de performance est utilisé pour la vérification du système dans une étape de cosimulation matérielle/logicielle au niveau modèle (simulation VHDL dite non interprétée) [72]. Le modèle de performances et l'architecture matérielle sont ensuite employés pour obtenir les descriptions matérielles et logicielles. Le système obtenu est validé par cosimulation fonctionnelle distribuée, interprétée et détaillée.

4.2.3.4 VCI

L'environnement VCI [107], [109], [106] est un outil de cosimulation fonctionnelle logicielle/matérielle (C/VHDL).

L'outil utilise un fichier de configuration décrivant les communications entre les parties logicielles (C) et les parties matérielles (VHDL) du système. Il génère automatiquement les interfaces de communication entre les parties du système et autorise tout type d'architecture distribuée de système. Les communications sont réalisées au travers de primitives simples de type « send » et « receive » permettant différents niveaux d'abstraction de la communication. La validation du système s'effectue par cosimulation fonctionnelle distribuée. Le fichier de coordination permet de spécifier la sensibilité de chaque signal et ainsi de construire un mode de synchronisation spécifique au système.

4.2.3.5 Seamless

Seamless [73] est un environnement de cosimulation logicielle / matérielle. La partie logicielle est confiée à un simulateur de processeur permettant d'atteindre une précision de simulation au niveau du jeu d'instruction.

L'environnement standard permet de simuler immédiatement un système mixte logiciel / matériel car les simulateurs sont interconnectés dans ce but et la mémoire nécessaire à la partie logicielle est prise en charge par le simulateur de processeur. La prise en charge de la mémoire par le simulateur de processeur permet à l'outil de proposer plusieurs niveaux d'optimisation des échanges entre ces derniers en modifiant le compromis entre précision et vitesse de simulation. Cette méthode permet au simulateur d'atteindre une vitesse de l'ordre de 100 k. instructions par seconde. Une interface de programmation standard est

disponible autorisant la connexion d'autres simulateurs, cependant, l'intégration d'un simulateur de processeur n'est possible qu'au travers de Mentor Graphics. L'architecture du système n'est contrainte qu'au type de processeurs disponibles dans l'outil et l'optimisation des accès mémoires n'est possible que dans le cas où chaque processeur dispose d'une mémoire locale. Le modèle de cosimulation adopté est une validation temporelle distribuée.

4.2.3.6 Cossap

Cossap [94] est un environnement de conception de systèmes de traitement du signal proposé par Synopsys. La spécification initiale est réalisée par assemblage d'éléments de bibliothèque standard ou construits par le concepteur et sous forme logicielle (C ou FORTRAN) ou matérielle (VHDL).

L'environnement permet de simuler le système à tous les stades de développement et de générer un module matériel et un module logiciel par « assemblage » des différents blocs. Pour la partie logicielle il est possible d'employer un simulateur de processeur afin d'obtenir des simulations précises. L'architecture du simulateur limite son usage aux applications de traitement du signal mais autorise des simulations abstraites très rapides. Le modèle de cosimulation adopté est un modèle maître – esclave fonctionnel basé sur une partie logicielle maître. Cossap est un environnement très intéressant pour la mise au point de chaînes de traitement du signal pour sa vitesse de simulation.

4.2.3.7 Coware N2C

Coware N2C [4], [111], [19], [30], initialement développé comme outil de recherche à l'IMEC puis industrialisé par CoWare Inc., est un environnement de conception conjointe à haut niveau de systèmes mixtes logiciel / matériel. Les langages de spécification employés sont C, C++ et VHDL et rendent l'environnement ouvert aux outils de développement standards.

La spécification initiale purement logicielle est composée de processus concurrents et interagissants en début ou fin de cycle. La simulation du système initial permet de guider la sélection des parties matérielles et logicielles. La connexion d'un simulateur de jeu d'instructions du processeur permet d'obtenir une validation précise du système. L'architecture du système consiste en un seul processeur auquel sont associées des parties matérielles.

L'environnement Coware N2C est particulièrement adapté au développement rapide des systèmes embarqués. La disponibilité de cœurs de processeurs et des interfaces correspondantes, aussi bien en VHDL qu'en C, permet d'obtenir facilement un prototype réaliste. Les possibilités de cosimulation à tous les niveaux d'abstraction autorisent une validation continue tout au long de la conception, et ce dans un environnement unifié. La validation fonctionnelle du système et la cosimulation C-VHDL intervenant avant le choix du processeur et de l'interface, peuvent être effectuées très tôt dans le flot de conception. L'utilisation des outils de développement standards permet une conception rapide du système.

4.3 Cosimulation géographiquement distribuée vs locale

Cette partie présente les éléments principaux séparant les approches locale et distribuée de la cosimulation. La première section présente les principales classes de support matériel pour les environnements de cosimulation. Les sections suivantes présentent les moyens techniques disponibles sur lesquels s'appuient les environnements de cosimulation. La dernière section présente un bilan des deux approches.

4.3.1 Exemples de supports

L'exécution d'une cosimulation est possible au travers de plusieurs topologies de réseau :

- Une seule machine est utilisée, la cosimulation n'est pas répartie sur un réseau. Tous les simulateurs doivent être présents sur la machine et se partagent le temps machine. L'environnement de cosimulation exploite aussi la même machine. Cette solution peut être intéressante si le système ne possède que peu de modules, peu de canaux de communication et beaucoup d'informations à échanger, ainsi le coût des communications est réduit au minimum entre les simulateurs. Cependant, le temps « CPU » est partagé entre les simulateurs.
- « Local area network », un réseau local est utilisé, la cosimulation est répartie sur plusieurs machines du réseau. Il est possible de répartir les modules à simuler sur les machines disponibles afin d'accéder aux licences des simulateurs et en profitant du parallélisme intrinsèque au système d'utiliser la puissance de calcul disponible. Cette solution est intéressante lorsque les simulateurs ne sont disponibles que sur certaines machines et lorsque la puissance de calcul nécessaire est importante. Le coût des communications devient important car basé sur le temps de transfert du réseau. Il est nécessaire d'employer des techniques d'optimisation adaptées au profil du système à simuler afin de réduire le coût des communications et d'obtenir de bonnes performances en temps de simulation.
- « Wide area network », un réseau régional voire mondial tel qu'Internet est utilisé. Les contraintes d'utilisation sont les même que dans le cas du réseau local mais plus fortes. Il convient de différencier le cas du réseau local à très haut débit du réseau Internet mondial ne possédant bien évidemment pas les mêmes performances. Cette solution permet de simuler un système dont certaines parties doivent être maintenues secrètes, ou nécessitent un simulateur peu répandu.

Chaque solution dispose d'avantages et d'inconvénients, mais aucune ne possède tous les avantages. Il est nécessaire qu'un environnement de validation ait la possibilité de s'adapter au support et de profiter de ses avantages.

4.3.2 Techniques de communication

Cette section présente les principales techniques de communication existantes entre plusieurs programmes. Cette énumération n'est bien entendu pas exhaustive mais tente d'être représentative.

4.3.2.1 Communication inter-processus (I.P.C.)

Les IPC sont un ensemble de moyens de communication inter–processus fournis par le système d'exploitation, UNIX en particulier. Les méthodes habituellement disponibles sont :

- **Les messages,** par le biais de primitives transportent des données entre deux processus et avec une synchronisation dépendante des primitives employées. Le système d'exploitation gère le protocole de communication et garantit la distribution et le stockage des messages via une pile système.
- **La mémoire partagée**, permet de mettre en commun un espace mémoire partagé par les processus connectés. Ce mode de communication est le plus simple et autorise les performances les plus élevées. Cependant, ce mode nécessite la mise en place de synchronisations en lecture ou écriture dans la mémoire partagée, non gérées par le système.
- **Les primitives de haut niveau** (e.g. JAVA), sont des outils qui surchargent les primitives de bas niveau présentées en offrant des primitives très flexibles dans le but de traiter des structures de données pouvant aller jusqu'au transport de classes d'objets. Ces primitives gèrent automatiquement la communication et les modes de synchronisation en utilisant des primitives de bas niveau de communication et autorisent des performances de toutes natures.

4.3.2.2 Sockets

Les "*sockets*" sont un mode de communication bas niveau qui existe sur tous les systèmes d'exploitation et assurent une compatibilité entre les machines et systèmes différents. Les deux principaux types de "*sockets*" sont différenciés par leur domaine d'application : Internet ou local. Les "*sockets*" de domaine Internet permettent de connecter un processus au réseau mondial mais bénéficient de performances en rapport. Les "*sockets*" de domaine local exploitent au mieux la communication engagée et autorisent des performances très élevées lors de communication au sien d'une même machine. Le principal intérêt des "*sockets*" est le fonctionnement identique quel que soit le domaine choisi, la programmation et l'optimisation des communications en sont simplifiées.

4.3.3 Techniques de synchronisation

La communication entre deux ou plusieurs processus peut être qualifiée en terme de synchronisation. Le mode de communication asynchrone est caractérisé par une communication non sure, il n'est pas assuré que l'information ait été reçue. Le mode synchrone assure, à contrario, la bonne réception de l'information. Le mode de communication est construit par l'emploi de primitives bloquantes ou non bloquantes, en émission ou en réception. Le blocage est assuré selon deux techniques distinctes :

- **L'attente active** aussi nommée « polling » en anglais, consiste à scruter de manière répétitive l'arrivée d'une information. Cette technique simple à mettre en œuvre est applicable dans les situations de processus locaux à une machine ou de processus répartis au travers d'un réseau de machines. Le principal travers de la technique est l'occupation du temps machine à l'attente, ce qui ne va pas dans le sens de bonnes performances en termes de vitesse.
- **Les sémaphores ou moniteurs**, fournis par le système d'exploitation, ils permettent de décharger les processus de la tâche d'attente. Ces outils permettent de bloquer un processus sans consommation du temps machine et ainsi obtenir des performances élevées. La contre-partie de cette technique est qu'elle est difficile à mettre en œuvre dans le cas de processus répartis au travers d'un réseau.

4.3.4 Bilan de la cosimulation géographiquement distribuée

L'approche géographiquement distribuée propose des avantages intéressant en termes de flexibilité d'emploi et de partage de charge. Les difficultés de mise en œuvre de cette approche résident dans la mise en œuvre des techniques de synchronisation et dans l'obtention de performances acceptables en terme de communication. En effet, les techniques de synchronisation présentées ne s'appliquent que dans l'approche locale et les communications locales proposent des performances très difficiles à égaler dans une approche distribuée. Les extensions présentées en section 4.5 tentent de répondre à ces difficultés.

4.4 Prototype MCI

Le prototype MCI [67], [66] est un environnement de cosimulation géographiquement distribuée. Ce prototype fait suite aux travaux sur VCI [109] et sur XXI [43] et reprend ou étend leurs caractéristiques principales :

- Cosimulation fonctionnelle distribuée,
- Cosimulation géographiquement répartie,
- Cosimulation de modules hétérogène, en particulier autres que C ou VHDL.

Ce prototype a donné naissance à l'environnement de cosimulation « CosiMate », actuellement commercialisé par la société AREXSYS [3]. Cette section présente le prototype MCI, ses caractéristiques, son architecture, les interfaces disponibles et son intégration à un environnement de conception conjointe.

4.4.1 Caractéristiques du prototype MCI

Les systèmes actuels sont complexes et composés de plusieurs modules réalisés par différentes équipes employant des langages différents. L'hétérogénéité de ces systèmes fait qu'il est difficile de simuler l'ensemble du système avant la production d'un prototype. L'environnement MCI apporte une solution de validation préalable par cosimulation de l'ensemble du système au travers des simulateurs commerciaux employés par les différentes équipes. Les principales caractéristiques du prototype MCI sont : Une validation fonctionnelle, distribuée et géographiquement répartie.

4.4.1.1 Validation fonctionnelle distribuée

La validation fonctionnelle consiste à assurer l'échange des données entre les simulateurs sans prise en compte de la notion du temps. La validation distribuée consiste à respecter la concurrence entre les modules composant le système. Les communications étant du type asynchrone, l'utilisateur doit veiller à ce que la communication réalisée soit représentative du système final sous peine de ne pas respecter le fonctionnement du système. L'ajout de protocoles est parfois nécessaire afin de ne pas perdre de données ou réutiliser des données périmées.

Ce type de validation est employé pour vérifier le respect du cahier des charges au niveau de la spécification système puis au niveau de la spécification architecture. L'avantage d'une telle cosimulation est les faibles ressources nécessaires à son exécution expliquant la rapidité de simulation d'un système décrit à ce niveau d'abstraction.

4.4.1.2 Conception concurrente du système

L'intrusion de la cosimulation dans la spécification étant restreinte au minimum, la possibilité de cosimulation de modules décrits à des niveaux d'abstraction différents et la répartition géographique de la cosimulation autorisent une conception de chaque module à des rythmes différents. Les équipes d'un même projet global ont une liberté de travail accrue, tant du point de vue des outils utilisés que de la synchronisation de la conception avec le projet global.

4.4.1.3 Validation géographiquement répartie

L'environnement de cosimulation MCI permet de répartir les modules du système à valider au travers d'un réseau. Il est alors possible de simuler un système entier en employant le simulateur propre à chacune des équipes de développement. Les simulateurs n'ont plus besoin d'être tous regroupés sur une machine ou sur un site particulier et la puissance de calcul disponible peut être colossale. Les équipes peuvent alors collaborer aussi bien au niveau de la simulation qu'au niveau du développement. Bien entendu, les performances de simulation sont dépendantes du réseau utilisé et de sa charge lors de la simulation du système. Cependant, la puissance de calcul disponible et les économies dégagées par l'utilisation d'un réseau de machines permet quelques compromis.

4.4.1.4 Description de la coordination inter modules du système

Afin de décrire la coordination des différents module d'un système il est nécessaire de la modéliser et d'intégrer celle ci à la cosimulation du système global. Selon le degré d'abstraction de la spécification, la description de la coordination doit être adaptée. L'environnement de cosimulation MCI utilise un fichier de coordination afin d'organiser la simulation globale. Ce fichier est composé des éléments suivants :

- Déclaration des modules à simuler ainsi que leurs propriétés. Les noms des modules et le type de simulateur à employer doivent être précisés. Le nom de la machine sur laquelle la simulation doit être exécutée peut être indiqué. Bien entendu, l'interface (les entrées et sorties) de chaque module est à préciser : nom du port, type et direction.

- Déclaration des interconnexions entre les ports des modules du système. L'environnement MCI analyse et regroupe les liens entre les simulateurs de façon à minimiser le nombre de connexions et économiser les ressources.

4.4.2 Architecture globale du prototype MCI

L'environnement de cosimulation MCI est composé des éléments suivants :

- Une interface avec chaque simulateur ou langage, construite à partir d'une interface de programmation générique du bus de cosimulation.
- Un bus de cosimulation arbitré par un 'routeur' chargé de la mise en place des liens entre les simulateurs et de la distribution des données.

La Figure 44 décrit l'architecture globale du prototype MCI et présente la répartition possible des éléments au travers d'un réseau de machines. Les paragraphes suivants présentent les éléments composant l'environnement MCI.

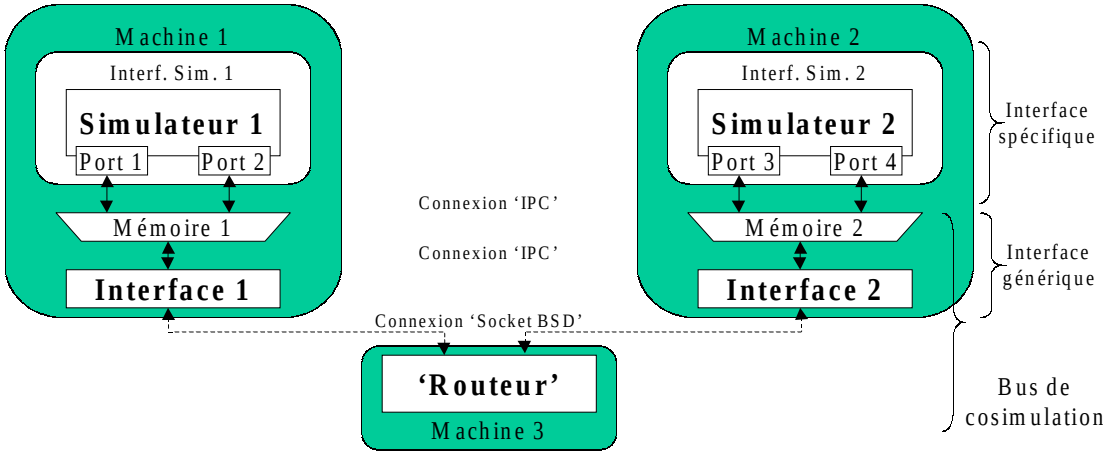

Figure 44 : Architecture du prototype MCI

Les éléments de l'environnement de cosimulation attenants au simulateur sont en partie spécifiques au simulateur, les ports d'entrée et de sortie, et en partie génériques, le programme d'interface et l'espace mémoire intermédiaire.

Les ports d'entrée et de sortie constituent l'interface spécifique au simulateur et sont construit à partir d'une bibliothèque permettant l'accès à la mémoire intermédiaire. Le programme 'Interface' assure la connexion entre la mémoire intermédiaire et le programme 'Routeur'. Toutes les connexions entre les éléments présents sur la même machine sont assurées par la technique de communication inter-processus fournie par le système d'exploitation UNIX.

L'échange de données suit le chemin suivant : Simulateur, Port de sortie du simulateur, Mémoire intermédiaire, Programme d'interface puis la donnée est acheminée vers un autre programme d'interface par le 'Routeur'. Le chemin inverse est suivi par une donnée entrante.

Le programme de routage, appelé 'Routeur', a la charge de transmettre les données d'un programme d'interface à un autre suivant le fichier de coordination fourni.

Les informations sont transmises au travers du réseau via une connexion par 'Sockets BSD' fournie par le système d'exploitation. Cette approche permet de s'abstraire du type de réseau employé et ainsi élargir l'application au réseau Internet.

L'architecture 'en couches' de l'environnement MCI permet de limiter le temps de développement de nouveaux ports lors de l'ajout de nouveaux simulateurs ou de ports spécialisés et d'optimiser les performances par des communications adaptées à chaque niveau.

4.4.3 Simulateurs commerciaux acceptés par le prototype MCI

Plusieurs langages et simulateurs sont actuellement acceptés par le prototype MCI. Cette section en énumère la liste :

- VSS [98] de synopsys pour le langage VHDL, l'interface permet de décrire des ports du type : Bit, entier et vecteur de bits. Ces ports sont sensibles en entrée et ne sont pas synchronisés avec le temps de simulation écoulé. Lors de l'arrivée d'une donnée, celle ci est intégrée par le simulateur immédiatement, il convient donc de mettre en œuvre un protocole de communication avec le module émetteur ou destinataire de l'information.
- GeodeSim [7], [26], [28], [36] de VERILOG pour le langage SDL, l'interface permet de décrire des ports de type entier ou réel. Ces ports sont connectés à la file d'attente SDL en entrée et subissent le protocole du module destinataire en sortie.
- Simulink [68] sous Matlab de Mathwork, l'interface permet de décrire des ports de type réel. Ces ports sont connectés aux flux de données Simulink et ne sont pas bloquant. La simulation Simulink utilise en entrée la nouvelle valeur ou à défaut, l'ancienne valeur. En sortie, les données sont envoyées à chaque cycle de simulation et interprétées selon le protocole mis en place avec le module destinataire.
- Modules en langage C ou C++, l'interface est fournie sous la forme d'une bibliothèque de fonctions et permet la description de ports de tous les types simples et composés C. Les ports en entrée comme en sortie sont du type non bloquants et nécessitent la mise en place d'un protocole afin d'assurer la transmission des données si besoin est.

Les interfaces actuellement disponibles permettent les connexions de tout type entre ces 4 types de modules. Cependant, l'adjonction de protocoles est nécessaire, et reste à la charge du concepteur, dans le but de ne pas perdre de données, réutiliser des données périmées, ou synchroniser les simulateurs.

4.4.4 Lien avec l'outil de conception conjointe MUSIC

L'environnement MUSIC permet le raffinement d'une spécification SDL composée de processus concurrents en un système mixte logiciel / matériel composé de modules C et VHDL. L'ultime étape de synthèse du code C et VHDL génère aussi un fichier de coordination compatible avec MCI. Ce fichier permet une cosimulation immédiate du système.

L'environnement MUSIC emploie le même format intermédiaire de fichier que le fichier de coordination. Il est alors possible d'employer MUSIC sur une spécification mixte, SDL et d'autres langages, afin de raffiner la partie SDL et les communication de tout le système (voir chap. synthèse hétérogène). Cette approche apporte une solution pratique d'adjonction de protocoles entre des modules incompatibles directement.

4.5 Extension du prototype MCI aux langages COSSAP et VHDL (RTL)

Le prototype MCI supporte les langages C, VHDL, SDL et SimuLink. Cette section présente deux nouvelles extensions pour les environnements COSSAP et VHDL-RTL sous VSS de synopsys. Ces extensions autorisent des simulations mêlant des modules décrits à des niveaux d'abstraction différents et autorisent des performances élevées.

4.5.1 Interface COSSAP

L'environnement COSSAP propose un simulateur orienté flot de données très performant. L'interface proposée permet d'intégrer ce dernier lors de la cosimulation d'un système complexe et ainsi profiter des capacités de l'environnement (paragraphe 4.4.1).

4.5.1.1 Présentation de l'interface COSSAP

L'interface COSSAP – bus de cosimulation se présente sous la forme d'une bibliothèque de ports d'entrée et sortie. Cette bibliothèque est intégrée aux bibliothèques standards COSSAP. L'interface dispose de ports de types entier et réel, les types de données habituels d'un schéma COSSAP. Selon l'emploi voulu, trois versions de chaque port est disponible : le port simple, le port bloquant avec acquittement et le port optimisé en terme de débit d'informations.

L'interface sous la forme d'élément de bibliothèque permet un utilisation simple de la cosimulation en disposant les éléments d'entrée et sortie de la même façon que les éléments standards COSSAP. Cette approche permet au concepteur de positionner un port facilement et à l'emplacement voulu. La Figure 45 montre la bibliothèque d'entrée – sortie et sont utilisation sur un schéma COSSAP.

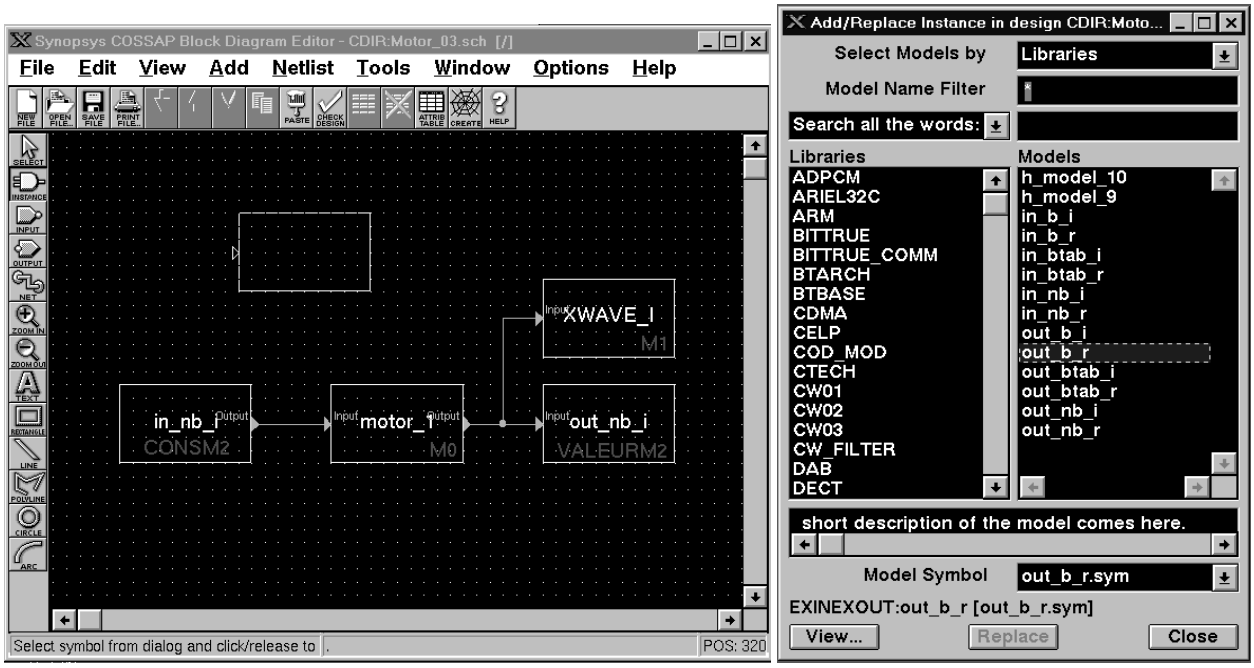

Figure 45 : Bibliothèque interface COSSAP - bus de cosimulation

4.5.1.2 Modèle de comportement de l'interface COSSAP

L'interface suit un comportement différent selon le type de port utilisé :

- Le port simple en entrée est une lecture bloquante du bus de cosimulation afin de suivre le modèle d'exécution COSSAP. En effet, un module COSSAP n'est exécuté qu'une fois toutes les données nécessaires présentes.
- Le port simple en sortie est une écriture non bloquante en direction du bus de cosimulation. Ce port n'assure pas de la bonne lecture de la donnée précédante, et peut donc, écraser des données.

- Le port avec acquittement en sortie est une écriture bloquante en direction du bus de cosimulation. Le blocage permet de s'assurer que la donnée précédante a bien été lue par le destinataire. Ainsi, il n'est plus possible de perdre des données.
- Le port avec acquittement en entrée suit le même comportement que le port simple mais dispose d'un signal d'acquittement en retour et facilite la connexion avec d'autres simulateurs nécessitant un tel signal.
- Le port optimisé en entrée et en sortie permet d'améliorer profondément les performances en termes de vitesse de communication par groupement de données. Ces ports constituent des blocs de données de la dimension fournie en paramètre au module COSSAP et communiquent le bloc entier au bus de cosimulation. Ainsi, même au travers d'un réseau peu rapide, il est possible d'obtenir de très bonnes performances en limitant le nombre d'échanges.

4.5.1.3 Limitations de l'interface COSSAP

La première limitation de l'interface COSSAP est due à son modèle d'exécution orienté flot de données avec abstraction du signal d'horloge. Ce type d'exécution implique des ports bloquants en entrée et rend complexe l'emploi de signaux de contrôle en entrée. En effet, ces signaux irréguliers risque de perturber l'exécution et bloquer le simulateur par manque d'information.

La seconde limitation porte sur les port optimisés et concerne le réglage de la dimension des blocs de données. Ce réglage doit être mené par le concepteur et ne doit pas être bloquant dans le cas d'un circuit en boucle, la dimension des blocs de données ne peut être supérieure au nombre de données contenues dans une boucle.

4.5.2 Interface VHDL-RTL

L'interface VHDL existante, issue de MCI [43], est bien adaptée aux modules VHDL comportemental car ils ne disposent pas de signal d'horloge. Une description VHDL RTL est régulée par un signal d'horloge et l'interface actuelle [43], ne peut pas synchroniser les signaux d'entrée avec l'horloge. Cette extension fournie une solution pour la simulation de modules VHDL-RTL.

4.5.2.1 Présentation de l'interface VHDL-RTL

L'interface se présente sous la forme d'un ensemble d'entités VHDL rassemblées dans une bibliothèque. Les types de données disponibles sont tous les types VHDL et des optimisations sont apportées afin d'obtenir des performances élevées. Les ports d'entrée et sortie sont connectés par génération d'instances des entités VHDL de la bibliothèque d'interface fournie. Plusieurs modèles de ports sont disponibles, avec ou sans acquittement, synchronisés ou non avec l'horloge.

4.5.2.2 Modèle de comportement de l'interface VHDL-RTL

Selon le type de port employé, le comportement de l'interface est différent. Les propriétés disponibles des ports sont les suivantes et peuvent être assemblées pour constituer un port au comportement adéquate :

- Entrée / Sortie, le port en entrée ou en sortie est par défaut non bloquant. Un événement est généré lors de l'arrivée d'une donnée et lorsqu'un événement atteint un port en sortie, la donnée est écrite sur le bus de cosimulation.
- Avec acquittement / sans acquittement, il est possible d'ajouter un signal d'acquittement de l'arrivé d'une donnée ou de signalisation de l'arrivée de la donnée à destination.

- Bloquant / Non bloquant, en entrée, le port bloquant ne libère le simulateur que suite à l'arrivée d'une donnée. En sortie, le port bloquant de l'interface attend la lecture de la donnée précédante avant d'écrire la nouvelle donnée.
- Synchrone / Asynchrone, le port synchrone possède un signal VHDL supplémentaire permettant la connexion d'une horloge. En entrée, la donnée n'est distribuée au simulateur qu'au front montant du signal d'horloge. En sortie, la donnée n'est prise en compte par l'interface qu'à l'instant du front montant. Le port asynchrone est le port par défaut, la donnée est reçue ou émise à l'instant où elle arrive.
- Optimisé / Simple, le port optimisé permet de constituer des blocs de données afin de minimiser le nombre de connexions. La dimension du bloc est un paramètre du port.

Les ports employés dans le cadre de la simulation VHDL–RTL sont de type synchrones, bloquants, avec acquittement et si les performances sont importantes, optimisés. Le mode bloquant permet de synchroniser les simulateurs. Le mode synchrone permet de synchroniser la mise à disposition des données avec l'horloge et donc le circuit VHDL-RTL. L'acquittement permet de préserver les données dans le bus de cosimulation et l'optimisation permet d'obtenir des performances de simulation élevées.

Les différents paramètres des ports permettent de connecter des simulateurs de natures totalement différentes dans les meilleures conditions de performance.

4.5.2.3 Limitations de l'interface VHDL-RTL

La principale limitation de cette interface est le réglage de la dimension des blocs de données dans le cadre d'une optimisation. Seul le concepteur peut régler ce paramètre qui ne doit pas entrer en conflit avec le nombre de données présentes dans les boucles de circuit (voir 4.5.1.3).

4.6 Exemples d'utilisation de MCI étendu

Cette section présente plusieurs exemples d'utilisation de la validation distribuée et répartie. Les deux premiers exemples présentent les techniques de cosimulation employées. Un exemple montre l'emploi de la cosimulation lors de la conception d'un contrôleur de moteurs. Le dernier exemple est une étude de cas industriel.

4.6.1 Cosimulation de systèmes hétérogènes SDL – COSSAP

Cette section présente un exemple de simulation d'un système composé de modules SDL et COSSAP. Cet exemple montre un système spécifié à haut niveau par l'emploi de deux langages abstraits.

4.6.1.1 Le système hétérogène

Le système s'articule autour de deux modules échangeants deux signaux. Le module SDL produit un signal qui est transformé par le module COSSAP puis retourné au module SDL. Le retour de donnée sert principalement d'acquittement de la bonne réception des données par le module COSSAP. La Figure 46 présente synthétiquement le système hétérogène.

Le module SDL suit le comportement d'un système SDL, basé sur la réaction de processus concurrents à l'arrivée de données (événements). De même, le module COSSAP suit le comportement d'un module flot de données. Chaque bloc du module COSSAP est activé lors de l'arrivée de données. Globalement, les deux modules utilisent le même modèle de calcul. La solution de connexion proposée consiste à transmettre les événements issus du simulateur SDL au simulateur COSSAP et inversement par l'intermédiaire du bus de cosimulation et ses interfaces SDL et COSSAP. Les ports utilisés sont les ports

standards, non bloquant en émission et bloquants en réception de données. De cette façon, les donnés sont transmises sans risque et le mode de transmission respecte les modèles de calcul des simulateurs.

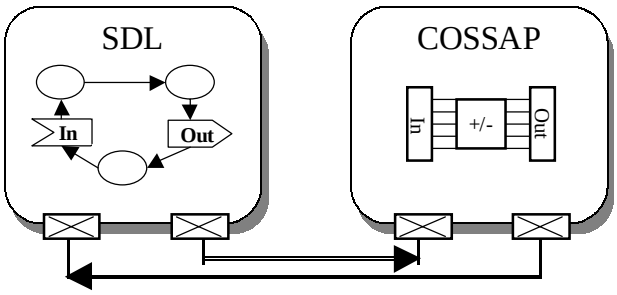

Figure 46 : Système hétérogène SDL – COSSAP

4.6.1.2 Cosimulation SDL-COSSAP

Le système entier est simulé à l'aide de l'environnement MCI. L'outil MCI utilise le fichier coordination contenant toutes les informations nécessaires sur l'interface de chaque module et les connexions. Les différents simulateurs sont exécutés et connectés automatiquement. Les simulateurs SDL et COSSAP sont exécutés sur des machines différentes au travers du réseau local. La copie d'écran (Figure 47) montre le système complet en cours d'exécution.

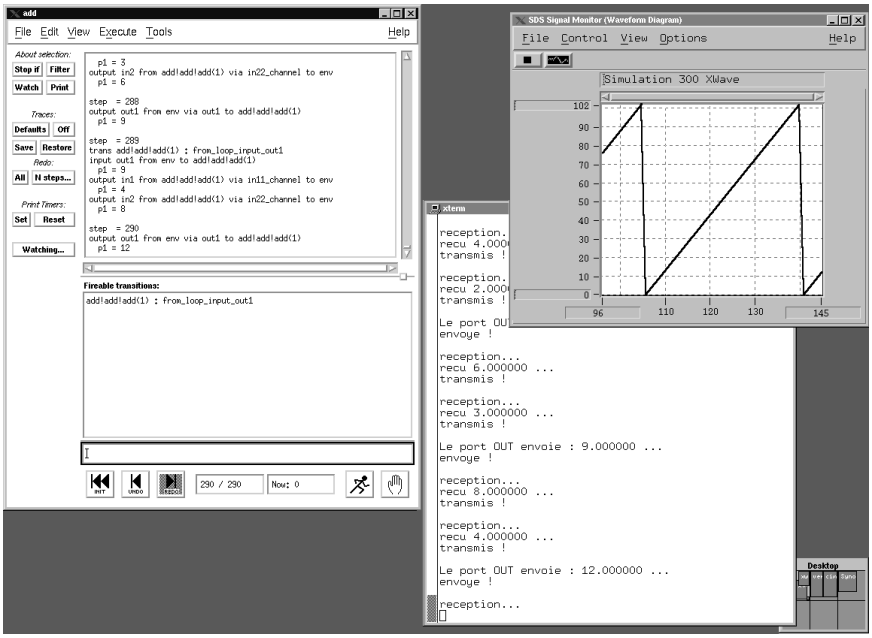

Figure 47 : Cosimulation SDL – COSSAP en cours d'exécution

Les outils d'analyse et les interfaces graphiques des simulateurs permettent d'interagir avec chaque module du système puis d'exploiter les résultats. La partie gauche de la Figure 47 montre le simulateur SDL et la partie droite le simulateur COSSAP.

La construction du prototype passe par des étapes de raffinement des modules et des communications entre ces derniers. L'étape finale est la mise en œuvre d'un modèle précis au niveau du cycle d'horloge sur un prototype. Ce travail peut être effectué par les techniques habituelles basées sur la synthèse comportementale [84].

4.6.2 Cosimulation de systèmes hétérogènes COSSAP – VHDL-RTL

Cette section présente un exemple de système hétérogène composé de module définis à des niveaux d'abstraction différents, abstrait pour COSSAP, plus concret avec le langage VHDL – RTL.

4.6.2.1 Le système hétérogène

Le système est composé de trois module, voir la Figure 48. Le module COSSAP de gauche fourni un signal qui est transmis au module VHDL – RTL pour traitement puis repris par un module COSSAP en vue d'analyse. Les données entrantes sont synchronisées avec l'horloge cadençant le module VHDL – RTL par l'interface spécifique VHDL – RTL. Les simulateurs COSSAP et VHDL sont synchronisés au travers de l'échange des données.

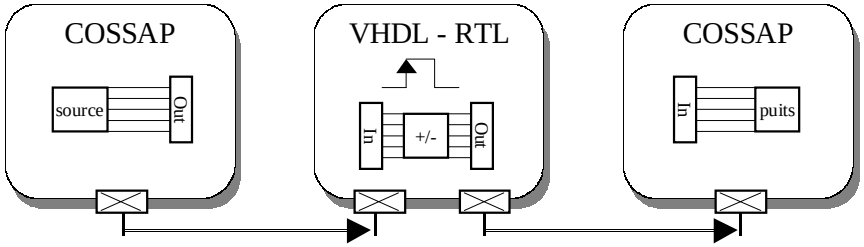

Figure 48 : Système hétérogène COSSAP – VHDL RTL

Cet exemple montre la possibilité de connexion entre deux simulateurs ne possédant pas la notion du temps et un simulateur la possédant. La simplicité de mise en ouvre par le biais des bibliothèques d'interface permet d'envisager une utilisation sur un exemple industriel (paragraphe 4.7).

4.6.2.2 Cosimulation VHDL-COSSAP

Le système est validé au travers de l'environnement MCI. Le fichier de coordination reprend les éléments de la Figure 48. Les trois simulateurs sont exécutés au travers d'un réseau de manière concurrente et les interconnexions assurent la transmission et la synchronisation des données. La Figure 49 montre le système en cours d'exécution, en haut le signal de départ et final COSSAP et en bas le simulateur VSS et la trace du signal synchronisé sur l'horloge du module VHDL – RTL.

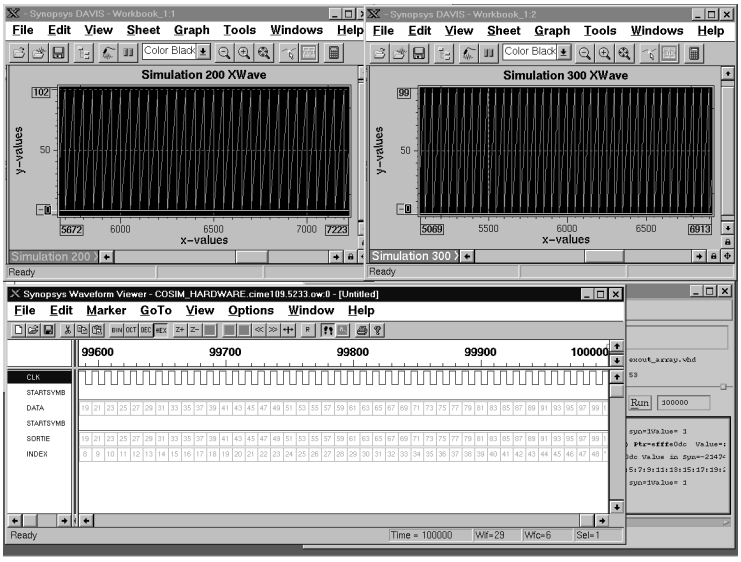

Figure 49 : Système hétérogène COSSAP / VHDL – RTL en cours d'exécution

4.7 Expérimentation industrielle, un modem VDSL

Cette section présente une expérimentation menée en commun entre le laboratoire TIMA, la société AREXSYS, le CNET(FT) et STmicroelectronic. L'expérience porte sur un modem VDSL. Une première partie tente une introduction rapide aux techniques xDSL. La seconde partie présente le système sur lequel porte l'expérience. Enfin, la dernière partie présente les expériences menées et en tire les conclusions.

4.7.1 Introduction aux techniques xDSL (x Digital Subscriber Line)

La jonction entre abonné et central est constituée de fils de cuivre dont les possibilités sont sous - utilisées car le réseau téléphonique a d'abord été conçu pour transporter la voix. La bande passante utilisée par les équipements de communication classiques est de l'ordre 3.3 kHz. Or, les caractéristiques physiques des lignes d'abonnés en cuivre autorisent en réalité la transmission de signaux pouvant atteindre des fréquences de 1 MHz . En modifiant les modems, il est donc possible d'optimiser l'utilisation de ces lignes en fonction de la distance séparant l'abonné de son central téléphonique, les paires de cuivre peuvent supporter des débits allant de 1.5 Mbits/s à 10 Mbits/s. Les technologies qui permettent cette astuce sont appelées "xDSL" et sont toutes dérivées de la technologie DSL utilisée dans le cadre de liaisons numériques RNIS (le type de codage utilisé est le même).

Le terme xDSL se décompose en quatre groupes : ADSL, HDSL , SDSL , et VDSL . A chacun de ces sous-groupes correspondent une utilisation et des caractéristiques particulières :

- Actuellement, l'ADSL (Asymmetric Digital Subscriber Line) est la technologie la plus au point et commercialement prête.
- Le VDSL (Very high Data rate digital Subscriber Line) est une technologie voisine, permettant des débits plus élevés encore (jusqu'à 58 Mbps). Alcatel a dévoilé cette technologie au salon Télécom 99 en décembre 1999
- Les autres groupes correspondent à des appellations encore imprécises et voisine de VDSL.

La technologie ADSL (Asymmetric Digital Subscriber Line) ou ligne asymétrique numérique utilise les fréquences supérieures à celles qui sont affectées au transport de la voix pour transmettre les données numériques. La traditionnelle modulation du courant électrique est remplacée par un procédé de transmission numérique DMT (Discrete Multitone) qui tronçonne la bande passante admissible du réseau téléphonique classique de 1.2 MHz en section de 4 kHz. L'utilisation de l'ASDL nécessite dans un premier temps une augmentation du spectre des fréquences reconnues pour passer à 1.1 MHz. La bande de fréquence est découpée en 256 bandes indépendantes plus petites de 4 kHz chacune. Le travail d'un modem ADSL consiste à additionner ces canaux pour atteindre le débit maximum.

Il s'agit d'une technologie asymétrique, le débit des données émises est plus faible que celui des données reçues. De plus le débit, dans les deux cas, varie avec la distance à parcourir. La liaison se trouvant entre l'abonné et le central, est divisée en trois canaux de transmission :

- Le haut de la bande (1MHz) est réservé au canal descendant unidirectionnel (central – abonné) à débit élevé (8 Mbits/s au maximum). Les possibilités de cette technique dépendent de la qualité de la ligne (longueur), ainsi seulement 50 % de la population française peut obtenir une liaison à 8 Mbits/s, le reste devra se contenter d'une liaison à 4 Mbits/s.
- En milieu de bande (entre 300 et 700 kHz), se trouve un canal bidirectionnel à débit moyen (entre 640 et 800 kbits/s) utilisé pour émettre les données.
- Le troisième canal est réservé soit à la téléphonie analogique classique (entre 0 et 4 kHz) soit au RNIS (entre 0 et 80 kHz). Avec cette technologie, l'abonné peut téléphoner et se connecter à l'Internet en même temps sur une seule prise téléphonique classique. A l'arrivée de la ligne de cuivre, les

fréquences vocales sont acheminées vers le réseau téléphonique classique tandis que les données sont dirigées vers le réseau Internet.

La technologie VDSL emploie des procédés voisins de codage et propose une simplification du protocole. Cependant, cette technique encore en développement n'est pas finalisée.

4.7.2 Présentation du modem VDSL expérimental

La Figure 50 présente le modem expérimental. Il est composé des trois parties principales :

- Deux circuits ASIC assurants une partie des tâches de contrôle (1), et une remise en forme des signaux en émission (2),
- Un DSP chargé du protocole et de certaines tâches de contrôle, tel que la reconnaissance du début d'un symbole (une trame) dans la suite de données,
- Un circuit de traitement du signal reçu et émis (bloc en pointillés).

Le chemin de données est représenté par les flèches, les données entrantes arrivent par le haut, via le signal RX data et les données sortantes sont modélisées en bas par le signal TX data.

La partie abordée dans cette section est le circuit de traitement du signal, et, plus précisément, les deux blocs RX et TX pour la réception et la transmission. Ces parties sont des modules de traitement du signal fonctionnant sur des vecteurs de 2048 données cadencées à 44 MHz ou 22 MHz. Les autres modules du circuit sont des mémoires pourvues d'accès spécifiques à leurs emploi.

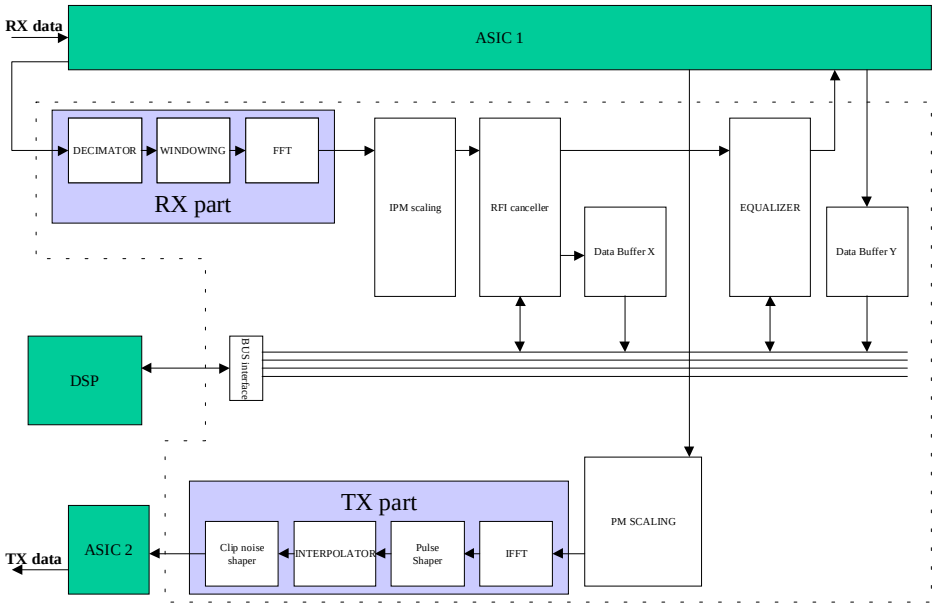

Figure 50 : Schéma global du modem VDSL

La spécification du circuit s'appuie sur deux environnements : COSSAP de SYNOPSYS et VHDL – RTL sous VSS de SYNOPSYS. La conception du circuit s'articule autour de trois étapes de spécification pour chaque module :

- Une spécification COSSAP (C) utilisant des données de type réel. Une telle spécification permet de vérifier le bon fonctionnement des algorithmes après écriture et de fournir une référence de résultat en terme de précision de traitement.

- Une spécification COSSAP (C) utilisant des données de type entier. Cette spécification permet de modéliser le traitement du signal par des unités de calcul en virgule fixe et ainsi mesurer la dimension adéquate des données à chaque niveau du circuit. La précision de simulation est améliorée et par conséquent, la vitesse de simulation diminuée. La perte de performance est cependant négligeable.
- Une spécification VHDL – RTL basée sur des vecteurs de bits. Elle est la traduction manuelle ou automatique des algorithmes C sur entiers. Cette spécification introduit le cycle d'horloge qui cadence le traitement et donc introduit une notion de temps. Cette simulation est extrêmement lente. Lorsque elle est appliquée à la totalité d'un système, la vitesse de simulation peut être inférieure à un cycle par seconde.

Cette expérience arrivant en fin de développement du circuit, tous les blocs du circuit sont disponibles sous les trois formes, COSSAP réels, COSSAP entiers et VHDL – RTL. Le principal intérêt de la cosimulation est ici de proposer une validation des blocs VHDL – RTL par cosimulation mixte du système global. L'approche par cosimulation autorise à simuler conjointement le bloc VHDL – RTL à valider avec le reste du circuit validé et spécifié en COSSAP. Une autre approche est de simuler parallèlement les différentes versions d'un bloc afin d'analyser les résultats en temps réel.

Les principaux obstacles à surmonter sont :

- La synchronisation de modules abstraits COSSAP avec des modules beaucoup plus concrets et détaillés VHDL – RTL.
- La performance de la simulation globale des blocs hétérogènes. Le rythme des données à l'intérieur du véritable circuit est de 44 MHz ou 22 MHz, il est donc important que les performances de simulation soient élevées afin de simuler quelques ms de fonctionnement du circuit.

4.7.3 Expérimentation menée

L'exemple d'utilisation de l'environnement de cosimulation proposé permet ici de valider l'approche et l'outil dans le cadre de la conception d'un système de dimension industrielle. L'expérience menée porte sur une partie du circuit présenté Figure 50. Les modules de réception et d'émission, amputés des blocs FFT et IFFT, sont chaînés afin d'obtenir une trace en sortie comparable au signal d'entrée. Ce système permet de valider les différents blocs du modem et dispose d'une approche très modulaire, facilitant les expériences. La Figure 51 présente le système réaménagé pour l'expérience.

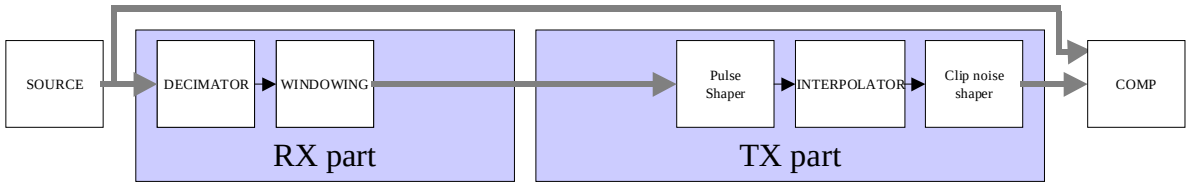

Figure 51 : Expérimentation menée sur le modem VDSL

Le système est composé de deux parties :

- Le récepteur, reçoit un signal cadencé à 44 MHz et produit un signal cadencé à 22 MHz. Le rôle principal de cette partie est de condenser l'information par élimination de détails.
- L'émetteur, reçoit un signal cadencé à 22 MHz et produit un signal cadencé à 44 MHz. Le rôle de l'émetteur est de générer les données intermédiaires afin d'obtenir le rythme initial de 44 MHz.

Le signal source provient d'un fichier de test du modem et contient une « trame » de 20480 éléments, il est utilisé en boucle pour une simulation en continu. Le résultat est comparé en sortie avec le signal original et un ratio signal à bruit est calculé. Une première simulation est menée entièrement en COSSAP afin

d'obtenir une référence fiable de résultat. La Figure 52 présente une copie d'écran de la simulation COSSAP du système construit pour l'expérience. La courbe gris clair représente le signal en entrée du « DECIMATOR » et la courbe en pointillés noirs le signal en sortie. Cette simulation de référence permet de voir que les deux courbes sont très semblables et présentent un rapport signal à bruit faible.

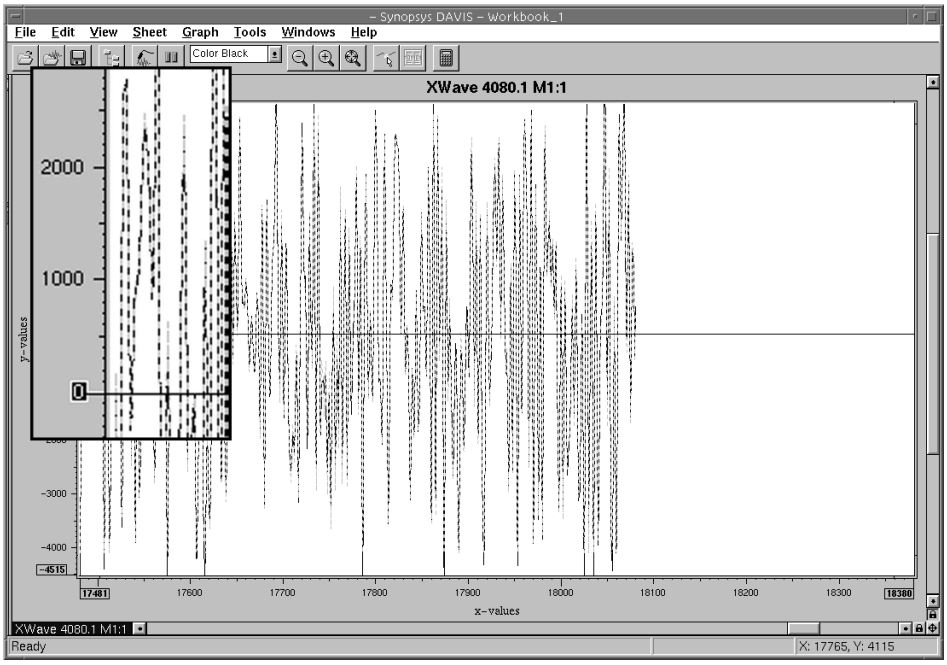

Figure 52 : Simulation de référence

Les performances de simulations sont indiquées au tableau comparatif Tableau 6.

4.7.3.1 Répartition géographique de la simulation

La première expérience consiste à valider l'emploi de l'environnement de cosimulation sur un exemple industriel et analyser les performances lors de la répartition géographique des blocs du système construit pour l'expérience. Cette expérience est réalisée par la séparation en deux partie du système : l'émetteur est exécuté sur une machine et le récepteur sur une autre. L'environnement de cosimulation est chargé des communications et de la synchronisation des deux simulateurs. La Figure 53 schématise la mise en place de l'expérience.

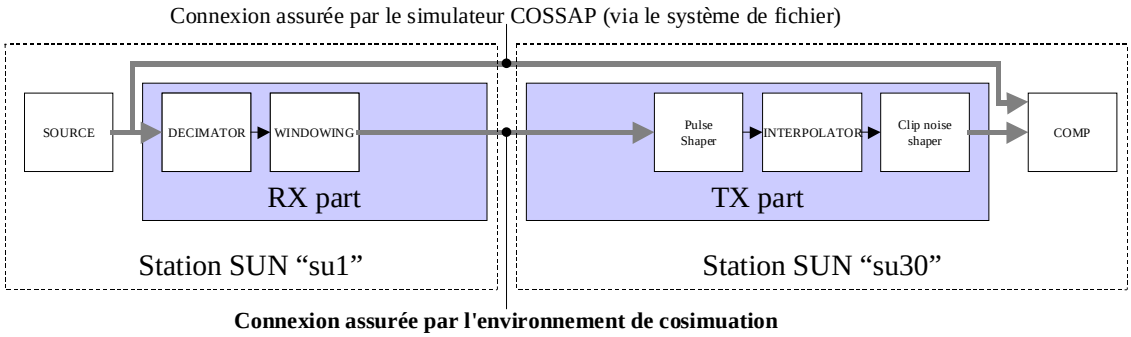

Figure 53 : Expérience avec deux simulateurs COSSAP

La connexion du signal initial est une « trace » gérée par l'interface graphique du simulateur de la partie récepteur. La connexion principale est assurée par l'environnement de cosimulation au travers d'un canal

de communication et via deux connecteurs COSSAP compatibles. Le canal est composé de deux signaux, un vecteur de données et un signal d'acquittement. Les ports d'entrée et sortie COSSAP utilisés sont du type signaux de type réel, avec acquittement et optimisés. Les données sont transmises par paquets et la liaison est protégée par un signal d'acquittement, les deux simulateurs sont ainsi synchronisés.

L'expérience est réalisée sur deux topologie réseau :

- Le mode local consiste à exécuter les deux simulateurs sur la même station de travail et profiter des performances des communications locales entre deux processus.
- Le mode réparti consiste à exécuter les deux simulateurs sur les station de travail précisées sur la Figure 53 et profiter des performances de calcul de chacune des stations et du parallélisme intrinsèque au système.

La Figure 54 montre une simulation locale en cours d'exécution sur une station de travail SUN Ultra-30. Les signaux présentés à gauche de la figure sont les signaux d'entrée et de sortie du modem VDSL réduit pour l'expérience. Le graphique de droite montre la superposition des deux courbes et montre la cohérence des résultats avec la simulation de référence.

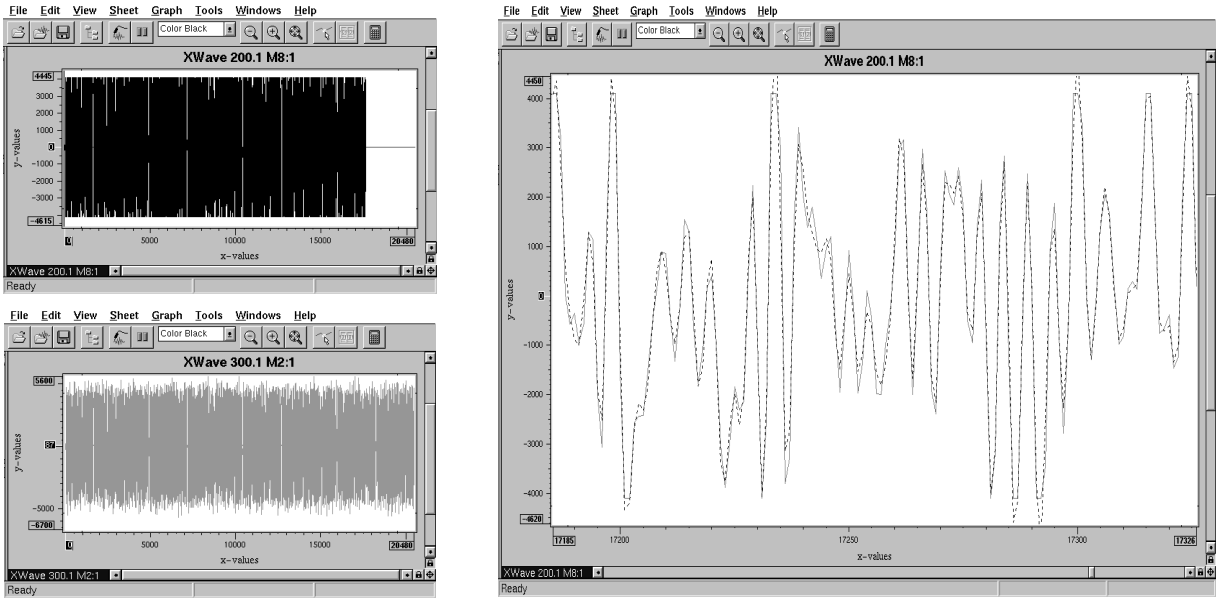

Figure 54 : Simulation COSSAP – COSSAP en mode local et en cours d'exécution

Afin de s'abstraire des phénomènes de charge du réseau et des stations utilisées, dix simulations ont été réalisées dans chaque catégorie pour produire une performance moyenne présentée au Tableau 6.

	Local	Réparti	COSSAP seul
Durée simulation	**3mn27**	**3mn33**	**4mn10**

Tableau 6 : Performances comparées de simulation

Ce tableau montre que le temps de simulation est meilleur lors de l'utilisation de l'environnement de cosimulation, pour le mode local comme pour le mode réparti. Ceci est principalement dû à une meilleure gestion des ressources par le système que par le simulateur COSSAP. En mode local, les communications ne sont pas pénalisantes et la répartition du temps machine entre les deux simulateurs est géré par le système d'exploitation. En mode réparti, chaque partie est plus « légère » à simuler tandis que les communications sont plus coûteuses mais peuvent être masquées par du temps de calcul.

4.7.3.2 Intégration d'un module VHDL-RTL à l'expérience

La seconde expérience consiste à valider l'environnement de cosimulation multiniveau. L'expérience porte sur le système modifié par le remplacement du module « INTERPOLATOR » COSSAP par un modèle VHDL – RTL. La Figure 55 montre l'architecture globale de l'expérience.

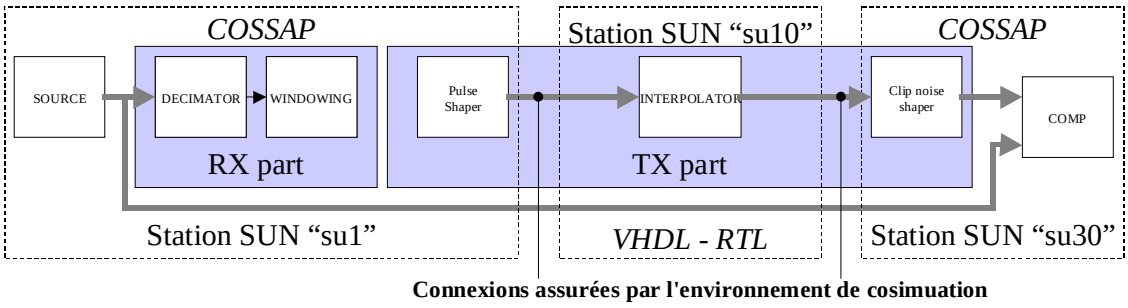

Figure 55 : Expérience avec deux simulateurs COSSAP et un simulateur VHDL (VSS – SYNOPSYS)

L'environnement de cosimulation supporte deux connexions : la connexion COSSAP – VHDL et la connexion VHDL – COSSAP. Les deux connexions utilisent le même canal de communication à deux composantes : un vecteur de données et un signal d'acquittement. Les points de connexion COSSAP utilisent les même connecteurs et ports que lors de l'expérience précédante. Le port d'entrée VHDL – RTL est de type entier, avec acquittement, bloquant, synchrone et optimisé. Ainsi, les données sont communiquées au simulateur VHDL à chaque front montant de l'horloge interne et ne sont ni perdues ni dupliquées entre le simulateur COSSAP et le simulateur VHDL.

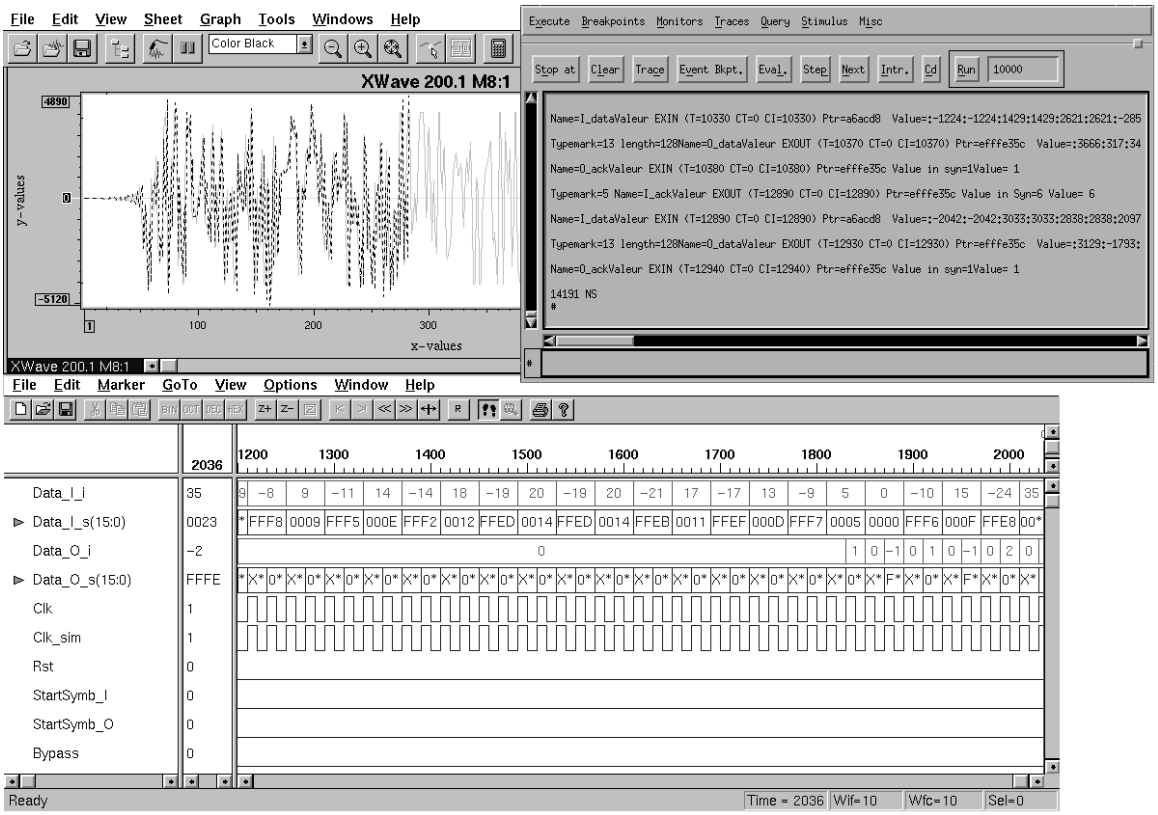

Figure 56 : Simulation avec deux simulateurs COSSAP et un VHDL en mode local

La Figure 56 montre une simulation locale en cours d'exécution sur une station de travail SUN Ultra-30.

Les simulations réalisées ont produit une performance moyenne présentée au Tableau 7.

La simulation en mode réparti est plus rapide car bien que le temps de communication soit élevé, il est entièrement recouvert par le temps de calcul du simulateur sur les éléments précédants.

L'expérience montre la possibilité d'une simulation multiniveau, entre COSSAP et VHDL – RTL, avec des performances autorisant la simulation de systèmes complexes sur de grandes durées.

	Local	Réparti
Durée simulation	**40 mn**	**26 mn**

Tableau 7 : Performances comparées de simulation

4.7.4 Conclusion

Lors de cette expérience, il a été démontré les éléments suivants :

- La faisabilité de la cosimulation multilangage par une cosimulation entre les simulateurs COSSAP et VSS pour VHDL.
- La faisabilité de la cosimulation multiniveau par une expérience de cosimulation entre des modules abstraits COSSAP et un module plus concret VHDL – RTL.
- La faisabilité de la cosimulation répartie au travers d'un réseau de stations de travail. L'intérêt de la répartition géographique apparaît lors de la simulation de modules conséquents tel que le module VHDL – RTL.
- La dégradation limitée des performances de simulation lors de l'utilisation de la cosimulation. L'expérience portant sur deux simulateurs COSSAP a démontré l'intérêt positif en termes de performances de la cosimulation.

Cette expérience a permis de développer des techniques et un savoir faire dans le domaine de la cosimulation multilangage et géographiquement répartie. Les résultats produits permettent d'envisager des perspectives très intéressantes pour ce projet. A court terme, il serait intéressant de poursuivre l'expérience et de l'étendre à tout le circuit puis tout le modem avec les circuits extérieurs. A plus long terme, deux voies se dessinent aux vues des résultats de l'expérience : la connexion avec un accélérateur de simulation VHDL et une expérience orientée performances.

4.7.4.1 Interface avec un accélérateur Voyager - IKOS

Afin d'améliorer les performances de simulation VHDL, il est possible d'utiliser un accélérateur de simulation. La société IKOS propose un outil nommé Voyager et proposant des performances de tout premier ordre : simulation d'environ un million de portes à une vitesse de l'ordre du MHz et avec un détail temporel.

L'avantage principal d'un tel environnement est de proposer des simulations rapide à un niveau très détaillé d'un circuit de dimensions importantes. Au sein de l'environnement de cosimulation, un tel outil permet d'éliminer le goulot d'étranglement que peut constituer le module VHDL.

Les travaux à approfondir sont une étude de l'interface d'un tel accélérateur afin de développer la même interface que celle employée pour le simulateur VSS pour VHDL – RTL, ainsi que l'application à une expérimentation de même complexité que le modem VDSL. La clef de la réussite est sans doute l'utilisation de ports optimisés et bloquants tels qu'employés pour le simulateur VHDL – RTL.

4.7.4.2 Amélioration des performances des simulateurs VHDL actuels

L'expérience menée permet d'espérer des performances de cosimulation élevées. Tout circuit orienté vers le traitement du signal peut être divisé en modules. Ce nouveau système peut être distribué et géographiquement réparti sur plusieurs stations de travail. Ceci permet de profiter du parallélisme généré par le « pipeline » construit. Cette approche est néanmoins limitée aux systèmes de la famille « traitement du signal » de par leur nature modulaire. Les boucles de circuit nécessitent une attention particulière, il convient de s'assurer que la dimension des vecteurs employés dans le cadre de l'optimisation des communications ne vienne pas contrarier le fonctionnement du circuit par une privation de données.

4.8 Conclusion

Ce chapitre a présenté un état de l'art dans le domaine de la cosimulation, sur les techniques, les outils existants et sur les outils développés au laboratoire TIMA. Deux extensions ont été présentées afin d'aborder le domaine des systèmes orientés vers le traitement du signal. Des exemples d'applications ainsi que le travail en collaboration avec plusieurs industriels ont conclu sur des résultats enthousiasmants.

Ce chapitre a présenté une étude de la cosimulation géographiquement distribuée et a démontré l'intérêt de l'approche même dans des domaines où les performances sont un critère important. Il reste encore beaucoup à faire dans ce domaine considéré comme peu intéressant par le milieu industriel de part les performances obtenues.

Chapitre 5

CONCLUSION

Ce chapitre présente les conclusions et les perspectives à moyen et long termes pour ce travail de recherche dans le domaine de la conception des systèmes hétérogènes multilangages.

Cette thèse s'inscrit dans le domaine de la conception multilangage pour les télécommunications. Un système actuel est composé de modules de natures différentes. Les équipes de développement emploient des environnements différents ce qui rend difficile la validation et la synthèse du système global. Les méthodes de conception actuelles proposent aux équipes de développer les modules indépendamment et la validation du système complet ne s'effectue que lors de la construction du prototype. Cette thèse a proposé une méthode de conception multilangage s'appuyant sur un environnement de synthèse d'interface et de validation multilangage afin de fournir aux équipes de conception le support à un travail coopératif au travers de leurs outils commerciaux. Ce document a présenté une étude des environnements de conception employés dans le domaine des télécommunications, un environnement de synthèse de la communication et un environnement de cosimulation. Ces éléments sont les piliers d'une méthode de conception multilangage adaptée aux nécessités actuelles.

La méthode de conception consiste à raffiner la spécification d'un système en une version moins abstraite. Pour ce faire, le raffinement est décomposé en deux parties : les modules et les communications. Un environnement de cosimulation permet de valider le système entier à chaque étape de raffinement et un environnement de synthèse de la communication guide le concepteur dans le raffinement des communications.

Le chapitre 2 a présenté une approche de la conception basée sur l'emploi des langages adaptés aux domaines d'application et sur une forme intermédiaire explicitant la coordination entre les modules du système. L'avantage de la forme intermédiaire et de faire évoluer sa sémantique au fur et à mesure du raffinement du système. La première partie de ce chapitre à présenté une étude sur trois langages représentatifs des langages de spécification de systèmes électroniques au travers de concepts issus de la littérature : la concurrence, la hiérarchie, la communication et la synchronisation. La seconde partie du chapitre a présenté une approche globale, synthétique et pragmatique de la conception de systèmes électroniques. Les concepts structurels de base et leurs variations au cours de la conception au travers de quatre niveaux d'abstraction ont été présentés. Cette approche permet de proposer un flot de conception focalisé sur la notion d'assemblage de modules. Une étude des solutions actuelles montre qu'aucune n'apporte de réponse complète. Plusieurs voies ont été explorées à la lumière de ces concepts et aboutissent à la conclusion qu'une forme intermédiaire liant les langages existants semble être une solution.

Le chapitre 3 a présenté une méthodologie de synthèse de communication pour les spécifications de systèmes multilangages. La méthodologie de synthèse proposée peut être exécutée automatiquement et permet au concepteur de transformer les primitives de communication de haut niveau, décrites dans différents formalismes et notations, en une réalisation physique. Aucune restriction n'étant imposée aux modèles de communication et aux langages de description, ce procédé peut être appliqué à une large classe d'applications multilangages. Cette approche a été illustrée sur l'exemple du système de contrôle de moteurs. L'approche proposée, en contraste avec d'autres approches de synthèse de communication, apporte des solutions de synthèse des communications pour les systèmes hétérogènes multilangages.

L'étude des différentes formes de spécification montre des divergences de concepts sur les plans du modèle d'exécution et de la communication. Les communications exprimées de manière très abstraite au niveau de spécification *service* ont besoin de raffinements afin de simuler ou de synthétiser le système entier. Pour cela nous proposons un environnement de synthèse de la communication articulé autour des deux éléments :

- **Un langage d'interconnexion** permet de spécifier les différents types d'interface d'un module et les interconnexions abstraites. Les trois types d'interface sont : le port abstrait (access) qui permet d'exprimer un point de connexion logique, le port physique qui permet d'exprimer le fil électrique porteur d'un signal et la prise (connector) qui permet d'exprimer l'interface rassemblant plusieurs ports physiques contraints à un protocole.

- **Un outil de synthèse des communications** permet le raffinement d'une communication abstraite en une réalisation. L'outil transforme les ports abstraits, les prises et les interconnexions abstraites en une réalisation par le biais d'une bibliothèque de communication comportant une description de la réalisation.

Le domaine d'application de cet environnement est vaste et s'étend de la réutilisation de composants existants (IP) à la connexion de modules hétérogènes.

Le chapitre 4 a présenté un état de l'art dans le domaine de la cosimulation, sur les techniques, les outils existants et sur les outils développés au laboratoire TIMA. Deux extensions ont été présentées afin d'aborder le domaine des systèmes orientés vers le traitement du signal. Des exemples d'applications ainsi que le travail en collaboration avec plusieurs industriels ont conclu sur des résultats enthousiasmants.

L'expérience menée en collaboration avec le CNET et ST – microelectronic nous a permis de developper l'environnement de cosimulation géographiquement distribuée dans le cadre d'une application industrielle, un modem VDSL. Lors de cette expérience, il a été démontré les éléments suivants :

- La faisabilité de la cosimulation multilangage par une cosimulation entre les simulateurs COSSAP et VSS pour VHDL.
- La faisabilité de la cosimulation multiniveau par une expérience de cosimulation entre des modules abstraits COSSAP et un module plus concret VHDL – RTL.
- La faisabilité de la cosimulation répartie au travers d'un réseau de stations de travail. L'intérêt de la répartition géographique apparaît lors de la simulation de modules conséquents tel que le module VHDL – RTL.
- La dégradation limitée des performances de simulation lors de l'utilisation de la cosimulation. L'expérience portant sur deux simulateurs COSSAP a démontré l'intérêt positif en termes de performances de la cosimulation.

Cette expérience a permis de développer des techniques et un savoir faire dans le domaine de la cosimulation multilangage et géographiquement répartie et nous permet de proposer un outil doté des carractéristiques suivantes :

- **Cosimulation multiniveau**, il est possible de simuler un système composé de modules décrits à des niveaux de détail différents, par exemple un module COSSAP et un module VHDL.
- **Synchronisation RTL**, l'environnement assure la synchronisation entre des modules COSSAP ou VHDL (RTL ou non) et des modules VHDL-RTL.
- **Performances**, l'environnement autorise la cosimulation de modules decrits dans des langages différents pour un surcout nul ou négatif dans le cas où le parrallélisme entre les modules peut être exploité et réparti entre plusieurs stations de travail.

Ce travail a présenté une méthode de conception pour les systèmes électroniques hétérogènes basée sur cinq niveaux d'abstraction. Un environnement de cosimulation a été conçu afin de valider un système au cours de son développement. La synchronisation des simulateurs est traitée par des protocoles de communication adaptés et autorisant une cosimulation multiniveau. La synthèse de la communication apporte des solutions d'interconnexion entre des modules existants ou de natures différentes aussi bien pour la synthèse que pour la simulation.

Les perspectives à long terme sont l'intégration des méthodes présentées, de synthèse de la communication et de cosimulation, autour d'un langage de coordination à définir. En ce qui concerne la méthode de conception présentée, les perspectives sont la définition d'un langage de coordination et le développement d'outils de manipulation de ce dernier. Les perspectives de développement de l'environnement de cosimulation sont axées sur l'étude des performances et l'adaptation des techniques de

communications au domaine d'utilisation. L'environnement de synthèse de la communication doit évoluer de façon à encadrer le développement des interfaces de couplage entre les parties matérielles et logicielles d'un système.

Annexe

RÉFÉRENCES

[1] R. Allen, D. Garlan, *A Formal Basis for Architectural Connection*, ACM Transactions on Software Engineering and Methodology, 6(3), July, 1997.

[2] G.R. Andrews, *Concurrent Programming, Principles and Practice*, Benjamin/Cummings (eds), Redwood City, Calif., pg. 484-494, 1991.

[3] Arexsys Homepage, Arexsys Inc., Available on-line at http://www.arexsys.com, 2000.

[4] F. Balarin et al., *Hardware-Software Codesign of Embedded Systems, The POLIS Approach*, Kluwer Academic Publishers, 1997.

[5] Baganne, J.L Philippe, E. Martin, *A Formal Technique for Hardware Interface design*, IEEE Transaction on Circuits and Systepms-II: Analog anf Digital Signal Processing, Vol. 45, No. 5, May 1998.

[6] D. Becker, R. Sigh, S. Tell, *An Engineering Environment for Hardware-Software Cosimulation*, Proc. of DAC, pg. 129–134, 1992.

[7] F. Belina, D. Hogrefe, A. Sarma. 1991. *SDL with APPLICATIONS from PROTOCOL SPECIFICATION*. Prentice Hall International (UK) Ltd..

[8] G. Berry et L. Cosserat, *The Esterel Synchronous Programming Language and its Mathematical Semantics*. Language for Synthesis, Ecole Nationale Supérieure de Mines de Paris, 1984.

[9] G. Berry, *Hardware implementation of pure esterel*, Proc. of the ACM workshop on Formal Methods in VLSI Design, 1991.

[10] A.D. Birrell, B.J. Nelson, *Implementing Remote Procedures Call*, ACM Transactions on Computer Systems, Vol. 2, No. 1, pg. 39-59, February, 1984.

[11] G. Booch, *Object-Oriented Design With Applications*, Benjamin/Cummings, Menlo Park, California, 1991.

[12] J. Buck et al., *Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems*, International Journal on Computer Simulation, January, 1994.

[13] J-P. Calvez, *A system specification model and method*, CIEM current issues in electronic modeling, v. 4, High-level system modeling : specification and design methodologies (J. Berge, O. Levia, J. Rouillard editors), Kluwer Academic Publishers, 1996.

[14] J-P. Calvez, D. Heller, O. Pasquier, *Uninterpreted cosimulation for performance evaluation of hw/sw systems*, Proc. of CODES, pg. 132-139, 1996.

[15] P. Chou et al., *IPChinook: An Integrated IP-based Design Framework for Distributed Embedded Systems*, Proc. of Design Automation Conference, 1999.

[16] Cierto Virtual Component Codesign, Cadence Inc., Available on-line at http://www.cadence.com/technology/hwsw/ciertovcc, 1999.

[17] P. Coste et al., *Multilanguage design of heterogeneuos systems*, CODES'99, pg.54-58, 1999.

[18] S. Coumeri, D. Thomas, *A Simulation Environment for Hardware-Software Codesign*, Proc. of ICCAD, 1995.

[19] Coware N2C Homepage, Coware Inc., Available on-line at http://www.coware.com, 2000.

[20] J.M. Daveau, T.Ben Ismail, A.A. Jerraya, *Synthesis of System-level Communication by an allocation-based Approach*, Proc. of the ISSS, pg. 150-155, 1995.

[21] J.M. Daveau, *Spécification Systèmes et Synthèse de la Communication pour le Co-Design Logiciel/Matériel*, Thèse de Doctorat INPG, TIMA Laboratory, Décembre 1997.

[22] J.M. Daveau et al., *Protocol Selection and Interface Generation for Hw-Sw Codesign*, IEEE Trans. on VLSI Systems, vol. 5, no. 1, pg. 136-144, 1997.

[23] R. Domer, Jianwen Zhu, D. Gajski. 1998 (Mar.). *The SpecC Language Reference Manual*. Technical report. University of California. Irvine.

[24] W. Ecker, M.Glesner, A. Vombach, *Protocol merging : A VHDL based method for clock cycle minimising and protocol preserving scheduling of I/O operations*, Proc. of the EDAC with euro-VHDL, pg. 624-629, 1994.

[25] M. Eisenring, J. Teich, *Domain-specific interface generation from dataflow specifications*, Proc. of CODES, pg. 43-47, 1998.

[26] R. Ernst, J. Henkel, T. Benner, *Hardware-Software Cosynthesis for Microcontrollers*, IEEE Design Test of Computers, vol. 10, n° 4, pg. 64–75, 1993.

[27] R. Ernst et al., *The Cosyma environment for hardware/software cosynthesis*, Journal of Microprocessors and Microsystems, Butterworth-Heinemann, 1995.

[28] O. Faergemand, A. Olsen. *Introduction to SDL-92*. Computer Networks and ISDN Systems, 26, pg. 1143-1167. 1994.

[29] D. Filo et al., *Interface Optimisation for Concurrent Systems Under Timing Constraints*, IEEE Transactions on VLSI, Vol. 1, pg. 268-281, September 1993.

[30] Frontier design's Homepage, Frontier design Inc., Available on-line at http://www.frontierd.com, 2000.

[31] D. Gajski et al., *Specification and Design of Embedded Systems*, Prentice-Hall, 1994.

[32] D. Gajski, F. Vahid, S. Narayan, *System-design methodology: executable-specification refinement*, Proc. EDAC-ETC-EuroASIC, pg. 458-463, IEEE CS Press, 1994.

[33] D. Gajski, R. Domer, J. Zhu. *IP-Centric Methodology and Design with the SpecC Language*, Contribution to NATO-ASI Workshop on System Level Synthesis. Il Ciocco, Barga, Italy. 1998 (Aug.).

[34] D. Gajski et al., *SpecC Specification Language and Methodology*, Kluwer Academic Publishers, Boston, MA, ISBN 0-7923-7822-9, March 2000

[35] A. Girault, B. Lee, E.A. Lee. *Hierarchical Finite State Machines with Multiple Concurrency Models*. Technical report. UC Berkeley, Electronic Research Laboratory. Berkeley, California. 1997 (aug.).

[36] W. Glunz. *Methodology for using SDL for Hardware Design at abstract Levels*. Technical report. Siemens AG. 1993 (sept.).

[37] R. Gupta, G. De Micheli, *System-level synthesis using re-programmable components*, Proc. European conference design automation, pg. 2-7, IEEE CS Press, 1992.

[38] R. Gupta, C. Coelho, G. de Micheli, *Synthesis and Simulation of Digital Systems Containing Interactive Hardware and Software Components*, Proc. of DAC, pg. 225–234, 1992.

[39] K. ten Hagen, H. Meyr, *Timed and Untimed hardware/software cosimulation: application and efficient implementation*, Proc. of CODES, 1993.

[40] N. Halbwatchs, F. Lagnier, C. Ratel. *Programming and verifying real-time systems by means of the synchronous data-flow lustre*, IEEE Transactions on Software Engineering, 18(9), September 1992.

[41] D. Harel, *Statecharts: a Visual Formalism for Complex Systems*, in Science for Computer Programming 8, pg. 231-274, North-Holland, 1987.

[42] D. HermanN, J. Henkel, R. Ernst. *An Approach to the Adaptation of Estimated Cost Parameters in the Cosyma System*, Third Int'l Wshp on Hardware/Software Codesign Codes/CASHE. IEEE CS Press, Grenoble. 1994 (Sept.).

[43] F. Hessel et al., *MCI – Multilanguage Distributed Co-Simulation Tool*, Chapter in Distributed and Parallel Embedded Systems, DIPES'98 organized by IFIP WG10.3/WG10.5, edited by F. Rammig, ISBN: 0-7923-8614-0, Kluwer, 1999.

[44] K. Hines, G. Borriello, *Dynamic communication models in embedded system co-simulation*, Design Automation Conference, pg. 395-402, 1994.

[45] C. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985 ;

[46] I. Jacobson, *Object-Oriented Software Engineering : A Use Case Driven Approach* (Addison-Wesley Object Technology Series), Hardcover, Mars,1994,

[47] A. Jantsch, S. Kumar, A. Hemani, *The Rugby Model: A Conceptual Frame for the Study of Modelling, Analysis and Synthesis Concepts of Electronic Systems*, in Proc. of Design Automation and Test in Europe Conference, 1999, pg. 256-263.

[48] A.A. Jerraya et K. O'Brien, *SOLAR: An Intermediate Format for System-Level Modeling and Synthesis*, Computer Aided Software/Hardware Engineering, IEEE Press 1994.

[49] A.A. Jerraya et al. Hardware and Software Co-design : Principles and Practice KLUWER. *Chap. Languages for system-level specification and design*, pg. 235-262. 1997.

[50] A.A. Jerraya et al., *Multilanguage Specification for System Design*, chapter in System-Level Synthesis (A.A. Jerraya and J. Mermet editors), NATO Series, pg. 103-136, Kluwer, 1999.

[51] A. Kalavade, E. Lee, *A Hardware-Software Codesign Methodology for DSP Applications*, IEEE Design and Test of Computers, September, 1993.

[52] R. Klein, *MIAMI: a hardware software co-simulation environment*, Proc. RSP, pg. 173-177, 1996.

[53] B. Kleinjohann. *Multilanguage Formalism*. MEDEA/ESPRIT conference HW/SW Codesign, pg. X.1.1-X.1.22. 1998 (sept.).

[54] C.D. Kloos et al., *From Lotos to VHDL*, Current issues in electronic modelling, 3, 1995.

[55] P.V. Knudsen, J. Madsen. *Communication Estimation for Hardware/Software Codesign*. CODES98, pg. 55-59. 1998.

[56] G. Koch, U. Kebshull, W. Rosenstiel, *A prototyping environment for hardware/software codesign in the COBRA project*, Proc of CODES, pg. 10-16, Grenoble, France, 1994.

[57] D. Ku, G. De Micheli, *HardwareC, a language for hardware design*, Tech. Rep. CSL-TR-88-362, Computer Systems aboratory, Stanford University, Aug., 1988.

[58] A. Lee, A. Sangiovanni-Vicentelli, *A Denotational Framework for Comparing Models of Computation.*

[59] B. Lee, E. A. Lee. *Hierarchical Concurrent Finite State Machines in Ptolemy*, Proc. Of International Conference on Application of Concurrency to System Design. Aizu-Wakamatsu City, Japan. Pg. 34-40. 1998 (march).

[60] B. Lin, S. Vercauteren, *Synthesis of Concurrent System Interface Modules with automatic Protocol Conversion Generation*, Proc. of ICCAD, pg. 395-399, 1994.

[61] LOTOS, ISO, IS 8807. *LOTOS a Formal Description Technique Based on the Temporal Ordering of Observational Behavior*. 1989 (Feb.).

[62] D. Luckman et al., *Specification and Analysis of system architecture using RAPIDE*, IEEE on software engineering – special issue on software architecture, 21(4), pg. 336-355, April, 1995.

[63] D. Luckman, J. Vera, *An event-based architecture definition language*, IEEE transactions on software engineering, 21(9), pg. 717-734, September, 1995.

[64] J. Madsen, B. Hald, *An Approach to Interface Synthesis*, Proc. of the ISSS, pg. 16-21, 1995.

[65] P. Mariatos et al., *Integrated System Design with an Object-Oriented Methodology*, vol.7 . Object-Oriented Modelling, J.M. Berge, Oz Levia, J. Rourillard Editors, CIEM : Courrent Issues in Electronic Modelling, Kluwer Academic Publishers, September, 1996.

[66] P. LeMarrec et al., *Hardware, Software and Mechanical Cosimulation for Automotive Applications*. RSP'98, 9th IEEE International Workshop on Rapid System Prototyping. 1998 (june).

[67] P. LeMarrec, *Cosimulation multiniveaux dans un flot de conception multilangage*, Thèse de Doctorat INPG, TIMA Laboratory, Juin, 2000.

[68] MathWorks. Matlab. 1998. Available on-line at http://www.mathworks.com/

[69] MATRIXx, Integrated Systems, Inc , 1998. Available on-line at http://www.isi.com/Products/MATRIXx

[70] E. Martin, A. Baganne, J.L Philippe, *Towards System Level Design Methodology For Dedicated Telecommunication Systems*, Invited Paper for the 10th International Conference on Microelectronics, Monastir, Tunisia, December 1998.

[71] N. Medvidovic, D. S. Rosenblum. *Domains of Concern in Software Architectures and Architecture Description Languages*. USENIX Conference on Domain-Specific Languages, Santa Barbara, CA, October 1997.

[72] MCSE Homepage, MCSE research group, IRESTE, Nantes University, Available on-line at http://www.ireste.fr/mcse, 2000.

[73] Mentor Graphics, *A cosimulation tool from Mentor Graphics for st20*, rapport technique, Mentor Graphics, 1996.

[74] F. Naçabal, *Outils pour l'exploration d'architectures programmables embarquées dans le cadre d'applications industrielles*, Thèse de Doctorat INPG, TIMA Laboratory, Février, 1998.

[75] S. Narayan, D. Gajski, *Synthesis of System-Level Bus Interfaces*, Proc. of EDAC with Euro-VHDL, pg. 395-399, February 1994.

[76] S. Narayan, D. Gajski, *Protocol Generation for Communication Channels*, Design Automation Conference, pg. 547-548, 1994.

[77] S. Narayan, D. Gajski, *Interfacing Incompatible Protocols Using Interface Process Generation*, Proc. of the IEEE Design Automation Conference, pg. 157-164, 1995.

[78] ObjecTime, Available on-line at http://www.objectime.on.ca/

[79] Object Management Group, CORBA Services; Common Object Services Specification, Technical Rapport, OMG, July, 1997.

[80] R. Ortega, L. Lavagno, G. Borriello, *Models and Methods for Hw/Sw Intellectual Property Interfacing*, NATO-ANSI Workshop on System Level Synthesis, 1998.

[81] R. Ortega, G. Borriello, *Communication Synthesis for Distributed Embedded Systems*, Proc of ICCAD, pg. 437-444, 1998.

[82] A. Österling et al., *The Cosyma system*, chapter in Hardware/Software Co-design: Principles and Practice (J. Staunstrup, W. Wolf editors), pg. 263-305, Kluwer, 1997.

[83] C. Passerone et al., *Fast hardware/software co-simulation for virtual prototyping and trade-off analysis*, Proc. of 34th DAC, pg. 389-394, 1997.

[84] P. Paulin et al., *High Level Synthesis and Codesign Methods: An Application to a Videophone Codec*, Proc. of EuroDAC/EuroVHDL. Brighton. 1995 (Sept.).

[85] Polis Homepage, Berkeley University, Hardware/software design group, Available on-line at http://www.eecs.berkeley.edu/~polis, 1999.

[86] Ptolemy Project Homepage, UC Berkeley, EECS, Available on-line at http://ptolemy.eecs.berkeley.edu, 2000.

[87] RAPIDE Project Homepage, Stanford University, Available on-line at http://pavg.stanford.edu/rapide, 2000.

[88] S.P. Reis, *Working in the garden environment from conceptual programming*, IEEE Software, 4, pg. 16-27, 1987.

[89] N.L. Rethman, P.A. Wilsey. *RAPIDE : A Tool For Hardware/ Software Tradeoff Analysis*, Proc. CHDL'93. Elsevier Science, Otawa,Canada. 1993 (Apr.).

[90] M. Romdhani et al., *Evaluation and Composition of Specification Languages, an Industrial Point of View*, Proc. IFIP Conf. Hardware Description Languages (CHDL). Pg. 519-523. 1995 (Sept.).

[91] M. Romdhani, *Ingénierie des systèmes complexes avec la méthode de conception concurrente co-design matériel/logiciel: application aux calculateurs embarqués*, Thèse de Doctorat INPG, TIMA Laboratory, Décembre, 1996.

[92] K.Van Rompaey et al., *Coware - a Design Environment for Heterogeneous Hardware/Software Systems*, The European Design Automation Conference. Geneve. 1996 (Sept.).

[93] J. Rowson, A. Sangiovanni-Vincetelli, *Interface-based design*, Proc. of Design Automation Conference, pg. 178-183, 1997.

[94] P. Runstadler, R. Crevier, *Virtual prototyping for system design and verification*, Synopsys documentation, 1995.

[95] SDL, Computer Networks and ISDN Systems. CCITT SDL, 1987

[96] SLDL Homepage, SLDL Comitte, Available on-line at http://www.inmet.com/SLDL, 2000.

[97] Mary Shaw et al. *Abstractions for Software Architecture and Tools that Support Them*, IEEE Transactions on Software Engineering 21(4), 1995.

[98] SYNOPSYS Homepage, SYNOPSYS Inc., Available on-line at http://www.synopsys.com/.

[99] Synopsys. COSSAP (Reference Manuals). Synopsys. 1997 (Jan.).

[100] System-level Synthesis Group Homepage, TIMA Laboratory, Available on-line at http://tima-cmp.imag.fr/Homepages/cosmos/sls.html, 2000.

[101] Synopsys Inc., SystemC: reference manual, Synopsys Inc., Available on-line at http://www.systemc.org, 1999.

[102] L. Thiele et al., *FunState – An internal design representation for codesign*, Proc. of ICCAD, 1999.

[103] D. Thomas, J. Adams, H. Schmit, *A Model and Methodology for Hardware-Software Codesign*, IEEE Design Test of Computers, pg. 16-28, 1993.

[104] UML, Available on-line at http://www.rational.com/uml/

[105] F. Vahid, L. Tauro, *An Object-Oriented Communication Library for Hardware/Software Codesign*, Proc. of CODES, pg. 81-86, 1997.

[106] C. Valderrama et al., *A Unified Model for Cosimulation and Cosynthesis of Mixed Hardware/Software Systems*, Proc. of European Design and Test Conference. Paris, France. 1995 (Mar.).

[107] C. Valderrama, F. Naçabal, P. Paulin, A. Jerraya, *Automatic Generation of Interfaces for Distributed C-VHDL Cosimulation of Embedded Systems: an Industrial Experience*, Proc. of RSP, pg. 72-77, 1996.

[108] C. Valderrama et al., *COSMOS: a transformational codesign tool for multiprocessor architectures in hardware/software codesign*, chapter in Hardware/Software Co-design: Principles and Practice (J. Staunstrup, W. Wolf editors), Kluwer, 1997.

[109] C. Valderrama, *Prototype Virtuel pour la génération des architectures mixtes logicielles/matérielles*, Thèse de Doctorat INPG, TIMA Laboratory, Octobre 1998.

[110] Verilog, *The Verilog Hardware Description Language* by Philip R. Moorby, Donald E. Thomas. Hardcover, May 1998

[111] D. Verkest et al., *CoWare – A Design environment for heterogeneous hardware/software systems*, Design Automation for Embedded Systems, vol. 1, no. 4, pg. 357-386, 1996.

[112] VHDL, Institute of Electrical and Electronically Engineers, IEEE Standard VHDL Language Reference Manual, STD 1076-1993. IEEE, 1993.

[113] D.S. Wiles, *Integrating syntaxes and their associated semantics*, Technical Report RR-92-297, Univ. Southern California, 1992.

[114] T. Yen, W. Wolf, *Communication Synthesis for Distributed Embedded Systems*, Proc. of ICCAD, pg. 288-294, 1995.

[115] J.S. Young et al., *Design and specification of embedded systems in Java using successive, formal refinement*, Proc. of 35th DAC, pg. 70-75, 1998.

[116] P. Zave, M. Jackson, *Conjunction as composition*, ACM Trans. on Software engineering and methodology, 8(2), pg. 379-411, 1993.

[117] J. Zhu, R. Döemer, D. Gajski, *Syntax and Semantics of SpecC language*, Proc. of Workshop on Synthesis and System Integration of Mixed Technologies, 1997.

[118] J. Zhu, D. Gajski, *OpenJ: an extensible system level design language*, Proc. DATE, 1999.

[119] D. Ziegenbein et al., *Combining multiple models of computation for scheduling and allocation*, Proc. of CODES, pg. 42-46, 1998.

Annexe

INDEX BIBLIOGRAPHIQUE

[1],49
[10],62
[100],20, 72
[101],19, 45, 50
[102],18
[103],18, 91
[104],19, 45, 49
[105],19, 60, 64
[106],88, 92
[107],91, 92
[108],72
[109],89, 92, 95
[11],49
[110],19, 46
[111],50, 60, 93
[112],19, 46
[113],18
[114],61
[115],18, 50
[116],18
[117],18
[118],18, 50
[119],18
[12],91
[13],92
[14],92
[15],61
[16],18
[17],72, 79
[18],91
[19],93
[2],57, 62
[20],60
[21],26, 57, 59, 60, 79
[22],47, 61
[23],18
[24],61
[25],60
[26],24, 91, 98
[27],91
[28],24, 26, 98
[29],64

[3],96
[30],93
[31],61
[32],61
[33],50
[34],43, 50, 52
[35],27, 37
[36],24, 98
[37],91
[38],91
[39],89, 90
[4],18, 61, 93
[40],49
[41],19, 45, 46, 49
[42],91
[43],21, 56, 82, 88, 95, 100
[44],60
[45],19, 45
[46],49
[47],43
[48],61, 63, 77
[49],18
[5],61
[50],57, 82
[51],91
[52],18
[53],18
[54],49
[55],19
[56],91
[57],91
[58],43
[59],18, 27
[6],91
[60],61
[61],19
[62],49
[63],49
[64],60
[65],49
[66],95
[67],56, 82, 88, 95
[68],24, 49, 98
[69],49
[7],24, 26, 98
[70],61
[71],49
[72],92
[73],92
[74],89
[75],61, 64
[76],60
[77],61, 79
[78],19, 45, 49
[79],19, 45
[8],49
[80],78
[81],19, 61
[82],91
[83],91
[84],102
[85],18
[86],18, 91
[87],18
[88],18
[89],18
[9],49
[90],18
[91],18
[92],18

[93],61
[94],93
[95],19, 45
[96],52
[97],49
[98],98
[99],19, 24, 45

Annexe

PUBLICATIONS

G. NICOLESCU, P. COSTE, F. HESSEL, Ph. LE MARREC, A. A. JERRAYA**, "Multilanguage Design of a Robot Arm Controller: Case Study"**, IEEE Computer Society Workshop on VLSI 2000, Orlando, USA, April 27-28, 2000.

S. MIR, B. CHARLOT, G. NICOLESCU, P. COSTE, F. PARRAIN, N-E. ZERGAINOH, B. COURTOIS, A.A. JERRAYA, M. RENCZ**, "Towards Design And Validation Of Mixed-Technology SOCs"**, 10th ACM Great Lakes Symposium on VLSI, Chicago, USA, pp. 29-33, March 2000.

P. COSTE, F. HESSEL, A.A. JERRAYA, **"Multilanguage Codesign Using SDL and Matlab"**, SASIMI 2000, Kyoto, Japan, April 2000.

F. HESSEL, P. COSTE, G. NICOLESCU, Ph. LE MARREC, N-E. ZERGAINOH, A.A. JERRAYA, **"Multi-level Communication Synthesis of Heterogeneous Multilanguage Specifications"**, Accepted for publication in the Proc. of ICCD, Texas, USA, September 2000.

F. HESSEL, P. COSTE, Ph. LE MARREC, N-E. ZERGAINOH, G. NICOLESCU, J.M. DAVEAU, A.A. JERRAYA, **"Interlanguage Communication Synthesis for Heterogeneous Specifications"**, Design Automation for Embedded Systems Journal, Vol.5, No.3, pp. 223-237, Kluwer Academic Publishers, August 2000.

A.A. JERRAYA, M. ROMDHANI, PH. LE MARREC, F. HESSEL P. COSTE, C. VALDERRAMA, G.F. MARCHIORO, J.M.DAVEAU, N.-E. ZERGAINOH, **"Multilanguage Specification for System Design and Codesign"**, TIMA RR-02-98/12 ; chapter in "System-level Synthesis", NATO ASI 1998 edited by A. Jerraya and J. Mermet, Kluwer Academic Publishers, 1999.

P. COSTE, F. HESSEL, Ph. LE MARREC, Z. SUGAR, M. ROMDHANI, R. SUESCUN, N. ZERGAINOH, A.A. JERRAYA, **"Multilanguage Design of Heterogeneous Systems"**, CODES'99, Rome, Italy, May 1999.

F. HESSEL, P. COSTE, Ph. LE MARREC, N.E. ZERGAINOH, J.M. DAVEAU, A.A. JERRAYA, **"Communication Interface Synthesis for Heterogeneous Specifications"**, TIMA RR-04-99/03, RSP'99, Clearwater, USA, June 99.

RESUME en français

La conception d'un système électronique devient de plus en plus difficile pour des raisons de complexité et d'hétérogénéité sans cesse croissantes, ajouté à cela, les méthodes de travail et les compétences des concepteurs évoluent moins vite que les possibilités techniques d'intégration. Ces constatations amènent à la conclusion que les méthodes de conception actuelles avouent leurs limites et ont besoin d'évoluer.

Le sujet de cette thèse porte sur une méthode de conception permettant d'appréhender de façon globale un système électronique complexe. Cette méthode repose sur une approche modulaire d'un système où sont séparés le comportement d'un module et les communications entre les modules. Le raffinement et la simulation de chaque module sont confiés aux outils habituels et adaptés au domaine d'application. La coordination globale est décrite au travers d'un langage de coordination et la simulation globale fait appel à la technique de cosimulation géographiquement répartie. L'environnement de raffinement des communications permet de préciser le comportement des communications afin de suivre le raffinement du comportement des modules jusqu'à la synthèse.

La méthode présentée doit permettre de réduire significativement le temps de mise sur le marché d'un produit par son approche globale permettant de simuler très tôt un système. Cette méthode, et plus particulièrement l'environnement de cosimulation, a été utilisée avec succès lors d'une expérience menée en collaboration avec le CNET(France Télécom), ST–microelectronics et AREXSYS.

TITRE en anglais

Heterogeneous System Design

RESUME en anglais

The design of electronic systems becomes more and more difficult mainly because of 2 points:

- Increase in complexity and heterogeneity,
- Engineers have more and more difficulties to meet the requirements of new integration possibilities.

This explains the need of new design methodologies.

The main goal of this work is to develop a new design methodology based on a global approach and modularity, whereby the behaviors and the communications are divided into different specifications. Each module is refined and simulated by specific and adapted tools. A coordination language specifies the global coordination and the simulation uses the cosimulation approach. The communication synthesis environment is used to refine communications to match the needed abstraction.

This methodology can reduce drastically the "time to market" of a new product by the global approach and the early simulations. This methodology has been validated by a successful experiment on a VDSL modem carried out in collaboration with CNET (France-Telecom), ST–microelectronics, AREXSYS and TIMA on a VDSL modem experience.

DISCIPLINE

Informatique

MOTS-CLES

Conception multilangage, cosimulation, synthèse de la communication, langage de coordination

INTITULE ET ADRESSE DU LABORATOIRE

Laboratoire TIMA-CMP, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France.

ISBN 2-913329-56-X broché **ISBN 2-913329-57-8 format électronique**