Spécification et validation des systèmes hétérogènes embarqués

G. Nicolescu

▶ **To cite this version:**

G. Nicolescu. Spécification et validation des systèmes hétérogènes embarqués. Autre [cs.OH]. Institut National Polytechnique de Grenoble - INPG, 2002. Français. NNT : . tel-00002938

HAL Id: tel-00002938
https://theses.hal.science/tel-00002938

Submitted on 3 Jun 2003

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

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

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

THESE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : Microélectronique

préparée au laboratoire **TIMA** dans le cadre de l'Ecole Doctorale
« Electronique, Electrotechnique, Automatique, Télécommunications, Signal »

présentée et soutenue publiquement

par

Eugenia Gabriela NUTA NICOLESCU

le 27 novembre 2002

Titre :

Spécification et validation des systèmes hétérogènes embarqués

Directeur de thèse
Ahmed Amine Jerraya

Jury :

M. Bernard Courtois,	**Président**
M. Guy Bois,	**Rapporteur**
M. Jean-Paul Calvez,	**Rapporteur**
M. Ahmed Amine Jerraya,	**Directeur de thèse**
M. Songjoo Yoo,	**Examinateur**

Le temps de la réflexion est une économie de temps

A ma mère,
A Bogdan

Remerciements

*Je remercie **M. Ahmed Amine Jerraya**, directeur de recherche au CNRS et responsable du groupe SLS du laboratoire TIMA d'avoir encadré ce travail de thèse, avec beaucoup de compétence, d'enthousiasme et de disponibilité. Je lui remercie pour ses conseils et pour la confiance qu'il m'a accordée au cours de ces années. Merci de m'avoir fait découvrir le monde de langages et d'abstractions et de m'avoir guidé tout au long de ce voyage captivant dans les endroits pas encore explorés de ce monde. Puisse ce travail être à l'hauteur de son encadrement.*

*Je remercie **M. Bernard Courtois**, directeur de recherche au CNRS et Directeur du laboratoire TIMA, de m'avoir accueilli dans son laboratoire et m'avoir fait l'honneur de présider le jury de ma thèse.*

*Je tiens à remercier **M. Guy Bois** et **M. Jean-Paul Calvez** d'avoir accepté de rapporter sur mon travail, pour l'intérêt qu'il ont porté à ce projet ainsi que pour leurs commentaires et suggestions qui m'ont permis d'améliorer la version finale de ce mémoire.*

*Je remercie également **M. Sungjoo Yoo** pour sa participation au jury de cette thèse, pour m'avoir offert la chance de travailler avec lui, pour ses conseils pertinents, pour sa constante disponibilité et sa patience.*

*J'adresse mes remerciements à **M. Kjetil Svarstrad** et **M. Steven Levitan** auprès desquelles j'ai beaucoup appris. L'opportunité de travailler avec eux m'a permis de découvrir des nouveaux domaines et d'éteindre les domaines d'application de ce travail.*

*Je remercie tous ceux qui m'ont aidé à découvrir l'enseignement. Je remercie particulièrement **M. Regis Leveugle, M. Gerd Finke, M. Paul Amblard, et M. M. Burlet.***

*Un grand merci à touts les membres de l'équipe SLS pour leur collaboration et pour leur sympathie. Je suis très reconnaissante à **Philippe, Pascal, Fabiano, Thierry, Nacer et Ludovic** qui m'ont beaucoup aidé au commencement de cette thèse par leur conseils et leurs encouragements. Je remercie **Wander, Lovic, Amer, Damien Yanick et Sami** pour leur professionnalisme et leur collaboration dans le cadre du projet Roses. Je remercie **Aimen, Lobna, Wassim, Iuliana, Benajmin et Dhia** pour leur enthousiasme, leur disponibilité et leur sérieux pendant leurs stages qui ont beaucoup contribué dans cette thèse. Je remercie **Sungjoo, Ferid, Arif et Arnaud,** qui ont partagé avec moi le bureau 122 et qui m'ont beaucoup aidé et appris pendant cette thèse. Je remercie les nouveaux thésards pour leur optimisme et leur soutien pendant les moments difficiles avant ma soutenance : **Wassim, Aimen, Lobna, Frederic Arif, Adriano et Anouar**. Je leur souhaite bonne chance pour la suite. Je remercie **Sonia, Frédéric, Nacer et Paul** pour leur gentillesse et leurs conseils précieux.*

Ma profonde reconnaissance à tous ceux qui ont pris le temps pour la lecture et la révision du français de ce mémoire : ***Antoine, Isabelle Dorothée et Ludovic****.*

Un grand merci à mes amis ***Dorothée et Patrice Gerin*** *pour avoir été à mes côtés dans le moments difficiles et pour m'avoir compris chaque fois que je ne pouvais pas les voir parce que je restais travailler. Enfin, pour touts les beaux moments que nous avons partagé ensemble en Esterel, Vercors, Belledonne ou bien à Vienne ou à Chambéry avec leurs parents et Neige. Qu'il soient assurés de mon amitié pour la vie.*

J'exprime aussi ma gratitude à mes amis et collègues au laboratoire TIMA : ***Adriana, Amel, Antoine, Cosmin, Dan, Faiza, Florin, Iuliana, Lobna, Lorena, Nacer*** *(en ordre alphabétique).*

Je remercie ***Béatrice et Bernard Schwartzmann*** *pour leur gentillesse, leurs conseils et tous les beaux moments que nous avons passé ensemble.*

Une pensée toute particulière va à mon mari ***Bogdan****. Nos thèses ont commencé le même jour et nous avons parcouru ensemble tout le chemin jusqu'à la soutenance. On dit généralement que c'est un chemin très difficile. Mais son amour, sa patience et sa tendresse ont rendu ce chemin très beau et agréable.*

Je finirai par remercier ***mon père*** *et* ***ma sœur*** *pour leur soutien constant et la confiance accordée. Je remercie* ***ma mère*** *pour son amour et tout le bonheur qu'elle m'a offert pendant le peu de temps que nous avons pu passer ensemble. Je dédie ce manuscrit à sa mémoire et je regrette infiniment qu'elle n'est plus parmi nous pour pouvoir l'embrasser très fort.*

Sommaire

Liste des Figures

Liste des Tableaux

Introduction

Sommaire

1. Conception des systèmes embarqués

Ce travail s'inscrit dans le contexte de la conception des systèmes embarqués monopuces.

Les prévisions stratégiques du rapport ITRS [Itr 01] annoncent que 70% des ASIC comporteront au moins un CPU embarqué à partir de l'année 2005. Ainsi la plupart des ASIC seront des systèmes monopuces (SoC pour "System On Chip" en anglais). Cette tendance semble non seulement se confirmer mais se renforcer : les systèmes monopuces contiendront des réseaux formés de plusieurs processeurs dans le cas d'applications telles que les terminaux mobiles, les processeurs de jeux et les processeurs de réseau. De plus, ces puces contiendront des éléments non digitaux (par ex. analogique ou RF) et des mécanismes de communication très sophistiqués. Etant donné que tous ces systèmes correspondent à des marchés de masse, ils ont tous été (ou seront) intégrés sur une seule puce afin de réduire les coûts de production. Il est prévu que ces systèmes soient les principaux vecteurs d'orientation de toute l'industrie des semi-conducteurs. Il est donc crucial de maîtriser la conception de tels systèmes tout en respectant les contraintes de mise sur le marché et les objectifs de qualité.

La Figure 1 montre l'architecture d'un système monopuce. Cette architecture est structurée en couches en vue de maîtriser la complexité. La partie matérielle est composée de deux couches :

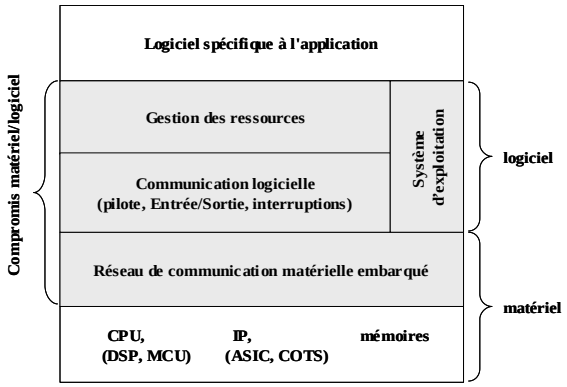


Figure 1. Représentation d'un système embarqué sous forme de couches

- La couche basse contient les principaux composants utilisés par le système. Il s'agit de composants standard tels que des processeurs (DSP, MCU, IP).
- La couche de communication matérielle embarquée sur la puce : il s'agit des dispositifs nécessaires à l'interaction entre les composants. Cette couche peut contenir un réseau de communication complexe allant du simple pont (bridge) entre deux processeurs au réseau de communication de paquets. Bien que le réseau de communication lui-même soit composé d'éléments standard, il est souvent nécessaire d'ajouter des couches d'adaptation entre le réseau de communication et les composants de la première couche.

Le logiciel embarqué est aussi découpé en couches :

- La couche basse contient les pilotes d'entrées/sorties et d'autres contrôleurs de bas niveau permettant de contrôler le matériel. Le code correspondant à cette couche est intimement lié au matériel. Cette couche permet d'isoler le matériel du reste du logiciel.
- La couche de gestion de ressources permet d'adapter l'application à l'architecture. Cette couche fournit les fonctions utilitaires indispensables à l'application qu'il est nécessaire de personnaliser pour l'architecture et ce pour des raisons d'efficacité. Cette couche permet d'isoler l'application de l'architecture. Bien que la plupart des systèmes d'exploitation fournissent une telle couche, il sera souvent utile d'en réaliser une spécifique pour l'application et ce pour des raisons de taille et/ou de performance. Les applications embarquées utilisent souvent des structures de données particulières avec des accès non standard (manipulation de champs de bits ou parcours rapides de tableaux) qui ne sont généralement pas fournis par les OS standard. D'autre part, les couches de gestion de ressources fournies par les OS standard sont généralement trop volumineuses pour être embarquées
- Le code de l'application.

Le processus de conception des systèmes sur puce peut être vu sous deux aspects : la conception des composants et l'intégration des différents composants dans le même système. Notre travail concerne l'aspect de l'intégration des composants.

La difficulté du processus d'intégration vient du fait que les différents composants sont hétérogènes. Ainsi, dans un système embarqué plusieurs processeurs, DSP, bus, etc. utilisant de protocoles de communication différents doivent communiquer. Dans ce cas nous parlons d'une hétérogénéité des composants en terme de protocoles de communication. Chaque composant peut être décrit dans un langage de spécification approprié ; ce qui implique l'hétérogénéité des langages de spécification. Il se peut aussi que les différents composants se trouvent dans des stades différents de raffinement, dans ce cas les composants sont hétérogènes en termes de niveaux d'abstraction. Les difficultés rencontrées pendant l'intégration des composants dans un système embarqué hétérogène sont les suivantes :

1. La première difficulté rencontrée pendant l'intégration des composantes hétérogènes est la **spécification** du système. Les différents composants d'un système embarqué étant hétérogènes, il est difficile de décrire les composants et les interconnexions dans un langage unifié. Afin de faciliter l'étape de spécification globale du système, des concepts qui permettent d'abstraire les interconnexions entre les composants hétérogènes sont nécessaires.
2. Pour l'implémentation de niveau RTL d'un système embarqué, des interfaces logiciel-matérielles peuvent être nécessaires entre les différents composants. Le développement de ces interfaces est un point clé pour la conception des systèmes électroniques, leur implémentation peut influencer les performances du système entier. La complexité de ces interfaces et leur diversité nécessite un temps trop long de conception vu les contraintes de mise sur le marché imposées. Dans ce contexte, très récemment les **modèles de simulation** pour la validation rapide des interfaces logicielles-matérielles sont devenus un besoin très important dans le processus de conception des systèmes embarqués.
3. L'hétérogénéité des composants des systèmes embarqués rend la **validation globale d'un système complet** très difficile. Cela nécessite un modèle de simulation hétérogène (qui puisse accommoder des différents protocoles de communication, niveaux d'abstraction ou langages de spécification) qui implique le développement des interfaces de simulation. Ces interfaces sont en charge des différentes adaptations entre les niveaux d'abstraction, les protocoles de communication ou langages de spécifications. Le développement d'un modèle de simulation est un travail long et fastidieux. En conséquence, très souvent, faute d'un modèle de simulation hétérogène, la validation globale n'est effectuée qu'au niveau du prototype, ce qui occasionne des coûts et des délais de développement très importants.

2. Objectif

L'objectif général encadrant cette thèse est de proposer un flot de conception pour les systèmes embarqués. Ce flot se propose :

- De décrire la spécification abstraite d'un système embarqué hétérogène
- D'automatiser le plus possible les passages aux niveaux d'abstraction inférieurs, jusqu'à la réalisation finale
- De pouvoir simuler l'ensemble du système à toutes les étapes de sa conception.

Ce flot est illustré Figure 2. Il part d'une spécification abstraite d'un système où l'interconnexion de chaque composant avec le reste du système est abstraite et génère une architecture de niveau RTL du système embarqué.

Cette thèse s'articulera autour de la spécification et la simulation des systèmes embarqués contenant des composants hétérogènes.

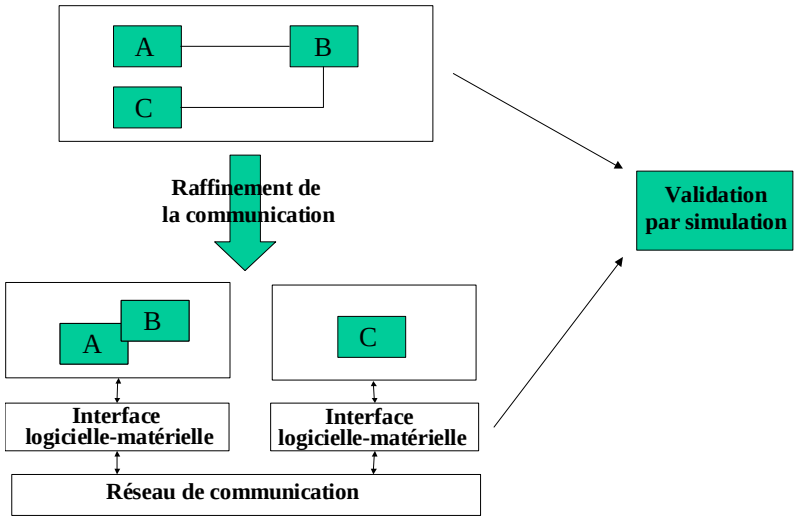


Figure 2. Flot de conception pour les systèmes embarqués hétérogènes

3. Contributions

Ce travail présente trois contributions : (1) une étude de la spécification des systèmes hétérogènes embarqués, étude qui a aidé à la définition d'un modèle de représentation pour la spécification en vue de la conception des systèmes hétérogènes embarqués, (2) la proposition d'une méthodologie de validation par simulation des systèmes hétérogènes embarqués et (3) la proposition d'un modèle de simulation pour la validation rapide des interfaces logicielles (les systèmes d'exploitation) dans les systèmes embarqués.

Chaque contribution sera présentée plus en détail par la suite

3.1. Etude sur la spécification et définition d'un format intermédiaire pour la conception des systèmes hétérogènes embarqués

Cette étude présente une approche pragmatique pour la spécification des systèmes. Les concepts de base utilisés pour la spécification des systèmes et les niveaux d'abstraction utilisés sont définis. Les langages existants pour la spécification des systèmes électroniques et leurs points forts et faibles sont présentés. Les solutions proposées actuellement pour exploiter les points forts et améliorer les points faibles des langages de spécification sont analysées. Finalement, cette étude amène à la définition d'un format intermédiaire pour la spécification en vue de la conception des systèmes hétérogènes embarqués. Ce format permet la spécification indépendamment des langages de spécification et couvre tous les niveaux d'abstraction, tout en respectant la séparation entre la communication et le comportement.

3.2. Méthodologie de validation par cosimulation des systèmes hétérogènes embarqués

Pour la validation par simulation, l'hétérogénéité des systèmes embarqués impose l'exécution conjointe des composants décrits en différents langages de spécification ou aux différents niveaux d'abstraction. L'adaptation des différents niveaux d'abstraction ou simulateurs participant à la simulation devient donc un point clé. Nous proposons une méthodologie pour la génération automatique des modèles exécutables des systèmes hétérogènes. Cette méthodologie est basée sur la définition d'un modèle exécutable générique où chaque composant est connecté avec le reste du système par des interfaces d'adaptation.

3.3. Modèle de simulation des systèmes d'exploitation pour la validation rapide du logiciel embarqué

La validation précise du logiciel à l'aide d'un simulateur du processeur s'avère trop lente pour les contraintes de temps de mise sur le marché imposées aujourd'hui pour la conception des systèmes embarqués. Pour surpasser cette difficulté nous proposons l'utilisation d'un modèle de simulation des systèmes d'exploitation qui permet la simulation du logiciel sur la machine hôte, ce qui apporte des gains importants en termes de vitesse de simulation, tout en respectant au maximum la réalisation finale du système d'exploitation.

4. Plan du document

Ce document est composé de cinq chapitres. Le Chapitre 1 présente l'étude de la spécification des systèmes hétérogènes embarqués. A partir de cette étude, dans le Chapitre 2 nous proposons un modèle de représentation pour la conception des systèmes hétérogènes embarqués. Le Chapitre 3 présente une méthodologie pour la validation par cosimulation des systèmes

hétérogènes embarqués. Le Chapitre 4 présente un modèle de simulation du système d'exploitation embarqué pour la validation native (sur la machine hôte) du logiciel. Pour finir, le dernier chapitre illustre l'utilisation des concepts proposés pour trois applications complexes : deux systèmes embarqués multiprocesseur (le modem VDSL et le système de téléphonie mobile IS-95 CDMA) et un micro-système optique (le commutateur optique).

Chapitre 1. Etude sur la spécification pour la conception des systèmes hétérogènes embarqués[1]

Sommaire

1.1. Introduction

La compréhension des concepts de base des langages de description des systèmes électroniques représente la clef de voûte de toute tentative d'automatisation du processus de conception. Malgré toutes les études effectuées jusqu'à présent pour définir un langage idéal de description, ce dernier n'existe toujours pas. Cela est dû principalement à 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 pouvoir modéliser le système à différents niveaux d'abstraction. Souvent, il faut même composer des modules décrits à des niveaux d'abstraction différents, dans le même système. Ces niveaux

[1] Ce travail a été effectué en collaboration avec Pascal Coste

d'abstraction peuvent manipuler des modèles de temps ou des concepts de communication qui sont différents et difficiles à composer.

- Les systèmes à concevoir peuvent être de natures très différentes (ex. discret/continu, numérique/analogique, logiciel/matériel) et même comporter des composants non électroniques (ex. mécanique, hydraulique, optique, 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, dans ce cas, aucun modèle exécutable n'est nécessaire.
  - La validation par simulation, nécessitant un modèle exécutable.
  - La vérification des systèmes, nécessitant un modèle formel du langage.
  - Le raffinement et la synthèse, une sémantique de réalisation est nécessaire.

Les quatre utilisations citées ci-dessus ne sont pas toujours compatibles. C'est ce qui explique la définition de sous-ensembles particuliers des langages pour des utilisations spécifiques. Tous les facteurs cités ci-dessus rendent l'analyse de 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 de pouvoir analyser les points forts et les points faibles des langages existants, mais aussi d'analyser les solutions actuelles proposées pour exploiter ces points forts ou pour améliorer ces points faibles.

Plusieurs études sur les langages existent dans la littérature. Gajski [Gaj 00] 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. Par contre, elle ne permet pas de comprendre les niveaux d'abstraction supportés par ces langages. Ed. Lee [Lee 97] 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 la comparaison des langages. Toutefois, cette étude ne concerne que les niveaux d'abstraction proches de la réalisation. Notre travail étend l'étude des langages aux plus hauts niveaux d'abstraction, et prend en compte les concepts spécifiques à la communication. Pour l'étude des langages, Hermani et Jantsch [Jan 00] présentent un cadre conceptuel qui analyse les concepts de base à travers les différents niveaux d'abstraction définis selon quatre axes (temps, communication, données et calculs). Dans leurs travaux, seul le cadre conceptuel est défini. Le positionnement dans ce cadre des différents langages et outils existants n'est pas présenté. En comparaison avec leurs travaux,

notre contribution est de définir un cadre conceptuel où l'axe principal est la communication. Cet axe est utilisé pour définir les notions de temps, données et calcul, mais aussi pour effectuer une étude plaçant les langages et les outils existants dans notre cadre conceptuel.

Une première contribution du travail présenté dans ce chapitre est l'étude des éléments fondamentaux utilisés par les concepteurs lors de la conception d'un système, à travers toutes les étapes du flot de conception en partant d'un niveau d'abstraction élevé. Une deuxième contribution est l'étude des différentes approches proposées actuellement pour la spécification des systèmes électroniques.

Ce chapitre est organisé comme suit. La section suivante introduit les concepts de base et les niveaux d'abstraction ; il définit aussi les différents concepts selon les niveaux d'abstraction. La troisième section présente brièvement les langages de spécification les plus utilisés et analyse les défaillances de ces langages pour la spécification des concepts de base à travers les niveaux d'abstraction. En fin, la dernière section analyse les approches proposées actuellement comme solutions possibles pour la spécification des systèmes hétérogènes.

1.2. Concepts de base et niveaux d'abstraction pour la spécification des systèmes

Ce paragraphe présente les concepts de base et les niveaux d'abstraction pour la spécification des systèmes hétérogènes. Cette étude est motivée par l'analyse des différents concepts pouvant couvrir toutes les étapes de conception d'un système allant jusqu'à la réalisation physique. Elle sera donc limitée aux modèles de spécification basés sur les concepts d'architectures. Sont donc exclus certains modèles purement fonctionnels de très haut niveau tels que B [Abr 97] et Z [Spi 89]. Ces modèles sont basés sur des formalismes algébriques et ils sont donc complètement indépendants de la notion de réalisation. Bien que ces langages soient très performants pour effectuer certaines tâches telles que l'analyse et la vérification formelle ils ne sont pas adaptés pour représenter des concepts spécifiques à une architecture de bas niveau.

1.2.1. Concepts de base pour la spécification des systèmes

Indépendamment du niveau d'abstraction, un système est modélisé comme un ensemble de modules hiérarchiques représentant une architecture abstraite. Les différents modules communiquent par des canaux de communication, via leurs interfaces.

Un module peut soit être hiérarchique, contenant un ou plusieurs modules, soit encore, il peut représenter une feuille dans la hiérarchie, présentant un comportement élémentaire. Le comportement peut être une tâche élémentaire ou un processus complexe pouvant cacher des flots d'exécution parallèles et hiérarchiques. 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.

Chaque module est connecté aux canaux de communication par une interface. L'interface de chaque module contient un ensemble de 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).

Un canal de communication prend en charge l'échange des données entre les modules. Il englobe tous les détails de communication et réalise l'échange des données entre les modules qu'il interconnecte. Les concepts qui définissent un canal de communication sont :

- Le média qui véhicule l'information (par exemple des fils physiques ou abstraits),
- Le comportement (la description des détails de communication),
- Le type de données transmises (par exemple des données génériques ou données avec représentation fixe).

Pour illustration, les concepts de base sont présentés dans la Figure 3 sur un exemple simple de système. Le système complet est représenté 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 est décrit par son comportement et le module hiérarchique B contient deux modules (B1 et B2). Les différents modules du système communiquent à travers les ports de leurs interfaces, par des canaux de communication (dans la figure, les modules A et B communiquent par le canal C1, via les ports P1 et P2). Le module A effectue un processus qui décrit un comportement composé d'opérations de calcul et de communication. Les opérations du comportement peuvent s'exécuter en parallèle ou en séquence. La communication permet l'échange de données avec l'extérieur ou entre les opérations parallèles du comportement.

Les concepts de module, interface, port et canal peuvent être utilisés à plusieurs niveaux d'abstraction. Ainsi, la structure de la Figure 3 peut illustrer aussi bien une représentation physique qu'un modèle de très haut niveau.

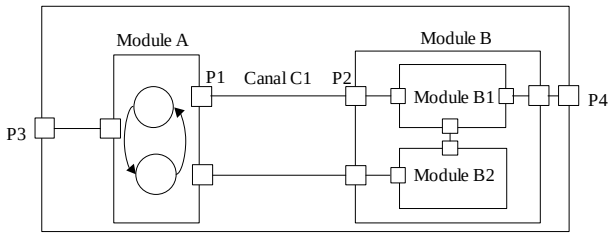


Figure 3. Exemple de modèle de système électronique

1.2.2. Niveaux d'abstraction de la communication

Cette section traite les modèles de communication en général sans faire de distinction entre les modèles hétérogènes et homogènes. L'accent est mis sur le type du réseau de communication sous-entendu pour la spécification.

Les différents niveaux d'abstraction de la communication considérés sont :

- Au **niveau service,** la communication est représentée comme une combinaison de requêtes et 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 sera une requête de service, comme par exemple « imprimer (fichier) ». CORBA (Common Object Request Broker Architecture) [Omg 97] est un exemple typique de ce niveau d'abstraction.
- Au **niveau transaction**, 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 aussi disposer d'un comportement complexe, comme 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 des protocoles de communication n'est faite. Des exemples de langages modélisant les concepts spécifiques à ce niveau sont SDL (System Description Language) [Sdl 87], UML (Unified Modeling Language) [Uml 02] et ObjecTime [Obj 00]. Certaines implémentations de StateCharts [Har 87] peuvent être placées à ce niveau.
- Le **niveau macro-architecture ou message** est le niveau spécifique à la communication par des *fils abstraits*, englobant des protocoles de niveau pilote des entrées/sorties (registre, file). Un modèle de ce niveau implique par conséquent le choix d'un protocole de communication et la topologie des interconnexions. Des exemples typiques des primitives de communication sont : l'envoi d'un message, l'écriture d'une valeur ou l'attente d'un événement ou message. Les langages caractérisant le mieux un système à ce niveau d'abstraction sont : CSP [Hoa 85], SystemC 1.1 [Sys 00], Cossap [Syn 02] et StateCharts [Har 87].
- Au **niveau RTL ou micro-architecture** la communication est réalisée par des *fils et des bus physiques*. La granularité de l'unité de temps devient le cycle d'horloge et les primitives de communication sont set/reset sur des ports et l'attente d'un nouveau cycle d'horloge. Les langages les plus utilisés pour la modélisation des systèmes à ce niveau sont SystemC 0.9-1.0, Verilog [Moo 98] et VHDL [Iee 93].

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

Niveau d'abstraction	Média de comm.	Primitive de comm. typique	Modèles typiques
Niveau Service	Réseaux abstraits	Demande (Imprimer, dispositif, fichier)	CORBA
Niveau Transaction	Canaux actifs	Send (fichier, disque)	SDL, UML
Niveau Macro-Architecture	Fils Abstraits (canaux abstraits)	Write(donnée, port) Wait until x=y	Cossap, StateCharts, CSP SystemC
Niveau Micro-Architecture	Fils Physiques	Set (Valeur, port) Wait (clock)	VHDL, Verilog, SystemC

Tableau 1. Niveaux d'abstraction pour la communication

1.2.3. Particularisation des concepts de base au travers des différents niveaux d'abstraction de la communication

Comme il a été dit plus haut, les systèmes peuvent être représentés comme un ensemble de modules hiérarchiques interconnectés, indépendamment du niveau d'abstraction. Cependant, les concepts de base (module, interface, contenu et canal de communication) ont des significations différentes suivant le niveau d'abstraction considéré. Ce paragraphe présente les concepts de base à travers les niveaux d'abstraction de la section 2.2.

Ainsi, au niveau service, les interfaces des modules sont composées de *ports d'accès à des réseaux abstraits* qui ont été présentés dans la section précédente. Ces ports fournissent des services d'un certain type et les opérations sur les ports sont des *requêtes et des services*. Les tâches élémentaires sont des processus qui interagissent avec l'environnement via des requêtes et des services. Le temps de communication est non nul et peut ne pas être prévisible. Le temps global est abstrait à un ordre partiel entre les requêtes de services. A ce niveau, les processus peuvent contenir des flots d'exécution (threads) hiérarchiques similaires à ce qui est présenté dans StateCharts [Har 87].

Au niveau transaction, 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 (par exemple « send », « receive », « put » ou « get »). Par contre, chaque canal peut encapsuler un comportement complexe. Le comportement des tâches élémentaires sont des processus qui communiquent avec l'extérieur par des transactions. Le temps de communication est aussi non nul et peut ne pas être prévisible. L'abstraction du temps est réduite à un ordre partiel des envois de messages. A ce niveau, les processus peuvent aussi contenir des threads hiérarchiques à la StateCharts [Har 87].

Au niveau macro-architecture, les ports composant les interfaces de ce niveau d'abstraction sont des *ports logiques*, qui assurent l'interconnexion des différents modules par des fils abstraits. Les opérations effectuées sur ces ports sont des *envoies de messages de type fixé* (ex. entier, réel). Ces

opérations peuvent cacher le décodage des adresses et la gestion des interruptions. A ce niveau, les opérations de communication peuvent durer un temps non nul mais prévisible. Le comportement des modules de base est décrit par des processus qui réalisent un pas de calcul et des opérations de communication. Ainsi, on peut avoir la notion de relation d'ordre global entre les événements du système (horloge globale). A ce niveau, les processus correspondent à des machines d'états finies étendues (EFSM) où chaque transition peut cacher un calcul complexe pouvant prendre plusieurs cycles d'horloge au niveau micro-architecture.

Au niveau RTL ou micro-architecture, les interfaces des modules sont composées de ***ports physiques***, qui permettent de connecter les différents modules par des fils physiques. Sur ces ports peuvent s'effectuer des opérations du type « ***set/reset*** » sur des signaux. La communication étant concrétisée par des fils physiques, les données sont transmises instantanément en représentation binaire fixe. A ce niveau, la gestion des interruptions et le décodage des adresses sont explicites. Dans ce cas, l'unité temporelle devient le cycle d'horloge et les processus correspondent à des machines d'états finies où chaque transition est réalisée en un cycle d'horloge.

Pour conclure, 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).

Comme nous le verrons dans la section 1.3, il est difficile de décrire tous les concepts à travers tous les niveaux d'abstraction à l'aide d'un seul langage.

Concept \ **Niveaux d'abstraction**				**Niveau service**	**Niveau transaction**	**Niveau macro-architecture**	**Niveau RTL (micro-architecture)**
Module	**Interface**	**Port**		Port d'accès à un réseau	Accès à un canal actif	Port logique	Port physique
		Opération (primitives de comm.)		Demande d'un service	Send/Receive vers un processus identifié	Assignation de donnée sur un port	Set/reset de bits
	Contenu	**Processus/tâche**		Objets parallèles communicants au niveau service	Processus parallèles communicants au niveau transaction	Processus parallèles communicants au niveau message	Processus parallèles communicants au niveau RTL
		Instance		Instance d'un module	Instance d'un module	Instance d'un module	Instance d'un module
		Canal de comm.	- media	- réseau abstrait	- canal actif	- fils abstraits	- fils physiques
			- comportement	- routage	- conversion de protocole	- protocoles niveau message	- transmission
			- donnée	- demandes de services	- transmission de données génériques	- type fixé de données	- données en représentation fixe

Tableau 2. Concepts de base à travers les niveaux d'abstraction

1.3. Les langages de spécification des systèmes électroniques

Le langage de description idéal doit couvrir tous les niveaux d'abstraction, tout en permettant les différent types d'utilisation, documentation, vérification, exécution et raffinement. Malheureusement, ce langage n'existe pas encore. Cette section analyse les limites des langages existants. Les différentes solutions qui existent pour remédier à ce problème seront discutées dans la section 1.4.

1.3.1. Méthodes actuelles de spécification des systèmes

Pour la spécification de 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[2], structures de données distribuées ou variables partagées). Les plus importantes méthodes de spécification sont :

ADL (Architecture Description Languages). Ces langages permettent la définition formelle de l'architecture de haut niveau d'un système complexe. Pour la spécification d'un système, les ADL utilisent trois concepts de base [Med97] : le module, le connecteur et la configuration (une combinaison particulière des connecteurs et modules). Ces concepts sont interprétés différemment par les différents ADL. Cela signifie que le même système peut être décrit différemment suivant le langage utilisé.

Rapide [Luk 95], Wright [All 97] et UniCon [Med 96] sont des exemples représentatifs d'ADL. [Sha 95] présente une étude comparative des différents ADL.

Méthodes orientées objet pour l'analyse et la conception des systèmes (OOA/OOD). Ces méthodes sont destinées à la description des systèmes complexes à un très haut niveau d'abstraction. L'idée principale est la décomposition de systèmes complexes en sous-systèmes plus faciles à maîtriser. Elles utilisent les avantages de l'approche orientée-objet pour la description, la visualisation et la documentation des systèmes.

UML (Unified Modeling Language) est actuellement la méthode de ce type la plus populaire. Cette méthode représente en fait l'unification des meilleurs concepts présents dans les trois plus importantes méthodes OOA/OOD : Booch [Boo 91], OMT [Mar 96] et OOSE [Jac 94]. Le principal inconvénient de cette méthode est l'absence d'un modèle exécutable pour la synthèse et la vérification. StateCharts [Har 87], ObjectTime [Obj 00] et SDL [Sdl 87] sont présentés comme des

[2] RPC (venant de l'anglais Remote Procedure Call) est un mécanisme de communication utilisé par un programme pour déclencher l'exécution d'une procédure dans un autre programme.

modèles exécutables de UML. Il s'agit de langages existants auxquels on associe des méthodes de décomposition de spécification conformément à UML.

Plusieurs langages ont aussi été développés pour la **modélisation de systèmes divers** à un niveau d'abstraction qui reste assez haut. Ces langages fournissent de larges bibliothèques pour la description des systèmes appartenant aux différents domaines d'application (mécanique, hydraulique ou traitement du signal). Les langages de ce type les plus connus sont Matlab [Mat 00] et Matrixx [Max 00]. Certains langages sont dédiés à la modélisation des systèmes orientés traitement signal et les plus utilisés dans le monde de la conception des systèmes électroniques sont Cossap [Syn 97] et SPW [Del 95].

Méthodes pour la modélisation des systèmes temps réel. Les systèmes temps réel sont des systèmes dont le comportement doit respecter des contraintes temporelles : entre la variation des entrées et la réaction correspondante, un temps maximum est imposé. L'importance prise par ce type de systèmes a entraîné le développement des langages synchrones tels que Lustre [Hal 92], Esterel [Ber 84] et des langages asynchrones tels que SDL et ObjecTime [Obj 00].

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

Langages de programmation. Certaines approches ont étendu les langages utilisés en informatique pour la programmation (C, C++) en y ajoutant les concepts spécifiques aux matériels, présentés ci-dessus. Le choix de ces langages se base sur trois raisons principales : ils fournissent le contrôle et le type de données nécessaires, la plupart des systèmes contiennent des parties matérielles et logicielles et donc, l'un de ces langages représente un choix naturel pour la partie logicielle. De plus, les concepteurs sont familiers avec ces langages et les outils conçus autour. Les langages développés consistent en la définition de fonctions ou de classes permettant de modéliser la concurrence, le temps et le comportement réactif. Les langages de ce type les plus utilisés sont N2C [Ver 96], SystemC [Sys 00], SpecC [Gaj 00] et Cynlib [For 01]. Syslib [Fil 02], [Fil 02a] est un langage complémentaire au langage Cynlib proposant des nouveau concepts qui permettent la spécification des systèmes logiciels/matériels au niveau système.

1.3.2. Capacités des langages pour la modélisation des concepts de base

Cette section analysera les langages de spécification selon leur capacité à modéliser les concepts de base à travers les niveaux d'abstraction présentés dans la section 1.2.2.

Les langages **HDL (VHDL, Verilog , etc.)** modélisent un système comme un ensemble d'entités physiques interfacées par des ports physiques. Les opérations sur les ports sont set/reset bits ou des 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 au niveau RTL, mais il manque toutefois plusieurs concepts qui sont nécessaires dans un flot de conception complet. Ainsi, le concept de port abstrait sur lequel peuvent s'effectuer des opérations indépendantes de protocoles ne peut pas être modélisé. 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, SystemC, Cossap et SPW** fournissent en plus des concepts des langages HDL, de nouveaux concepts **:** les systèmes sont découpés en modules abstraits qui ont des interfaces composées d'accès aux canaux de transmission. Les opérations effectuées par les accès sont des RPC ou du « token flow » (pilote de jeton). Les canaux transmettent ou temporisent des données de types fixés. Néanmoins des concepts sont manquants : le concept de réseau abstrait capable de réaliser le routage des services et le concept de port d'accès à un réseau abstrait.

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 les contraintes de temps imposées.

Ces langages supposent que la communication est réalisée de façon instantanée. Donc, ils ne peuvent pas modéliser une communication hiérarchique et distribuée.

Les langages asynchrones modélisant les systèmes temps réel apportent aussi leurs nouveaux concepts. Les modules abstraits composant le système communiquent au travers des interfaces composées d'accès aux canaux actifs. 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. Par ailleurs, ces langages n'offrent pas la possibilité de décrire la communication de niveau service.

Le langage UML et les ADL 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, accessibles 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 inconvénient 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 illustrés dans le Tableau 3.

Nous pouvons remarquer que chacune de ces méthodes présente des points forts pour la modélisation de certains concepts, mais aussi des points faibles pour la modélisation d'autres concepts. Cela signifie qu'aucun langage existant ne présente la totalité des capacités nécessaires à la modélisation des systèmes électroniques à tous les niveaux d'abstraction. Ils offrent donc seulement des solutions partielles pour la spécification des systèmes électroniques actuels. C'est pourquoi, depuis quelques années, l'un des enjeux du monde industriel et de la recherche est de proposer de nouvelles solutions pour une spécification complète des systèmes électroniques.

Concept \ Langage				HDL	CoWare, SystemC, Cossap, SPW	Temps réel Langages synchrones	Temps réel Langages asynch. UML, ADLs
Module	Interface		Port	Ports abstraits	Ports d'accès au réseau abstrait	Ports d'accès aux canaux actifs	Ports physiques
			Opération	Protocoles indépendants des opérations	Demande d'un service	Opération sans délai	Opération sur les données en repr. fixe, protocoles détaillés
	Contenu		Processus/tâche	Multitâche synchronisée	Objets parallèles communicants au niveau service	Indépendance d'un cycle fixe d'exécution	Modèle au niveau cycle-près Opérations spécifiques à une implémentation
		Canal de communication	Media	Canaux abstraits	Réseau abstrait	Comm. hiérarchique et distribuée	Signaux physiques
			Comportement	Comportement autre que transmission	Routage	Autres que 'broadcasting'	Flot de données uniforme fixé
			Données	Type générique de données	Demandes de services	Type générique de données	Représentation fixée d'une donnée

Tableau 3. Concepts manquants des langages de spécification

1.4. Solutions pour la spécification des systèmes hétérogènes

Afin de spécifier les systèmes hétérogènes, plusieurs solutions sont actuellement proposées. Ces solutions sont appliquées dans l'une des deux approches ci-dessous :

- Approche basée sur l'utilisation d'un langage de spécification : un seul langage est utilisé pour la modélisation d'un système électronique. Ce langage doit fournir tous les concepts nécessaires à la modélisation des systèmes complexes à tous les niveaux d'abstraction en couvrant les différents types d'applications [Lee 00], [Ern 99].
- Approche multi-langage (indépendante des langages de spécification) : des langages spécifiques sont utilisés pour la modélisation des différents modules d'un système électronique. Il faut donc disposer d'un modèle de coordination qui décrit les interconnexions entre les différents sous-systèmes [Jer 97].

1.4.1. Approche basée sur l'utilisation d'un seul langage de spécification

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

La première solution implique **la définition d'un nouveau langage**. Ce langage peut avoir une nouvelle syntaxe (comme dans le cas des langages Rosetta créé par le comité SLDL (System Level Design Language) [Sdl 87]) ou avoir une extension d'une syntaxe déjà existante. Cette dernière alternative a été adoptée pour la définition du langage SpecC [Gaj 00] et du langage SuperLog [Sup02] proposé par Co-Design Automation.

La deuxième solution propose **l'ajout d'une extension exécutable à un langage déjà existant**. Selon les desiderata du nouveau langage, cette solution implique seulement l'addition des librairies ou des classes d'exécution. Des exemples de langages obtenus par l'ajout d'extensions exécutables au langage C++ sont SystemC [Sys 00] ou C-level [Cle 00].

1.4.2. Approche multi-langage

Cette approche implique la modélisation de chaque module du système dans un langage spécifique et approprié, permettant d'exploiter au mieux les performances des langages existants.

Le problème qui se pose dans ce cas est la spécification des interfaces et des interconnexions entre les différents modules du système. Cela implique l'existence d'un modèle de représentation ou de coordination. Ce modèle peut être perçu au niveau de l'utilisateur comme sous la forme d'un langage de coordination, ce langage étant traduit ensuite dans un format intermédiaire en vue du

raffinement. Indépendamment des langages employés, le format intermédiaire est généralement utilisé comme une représentation interne pour le raffinement inter-niveaux de la spécification initiale. Les différents types de formats intermédiaires et leurs utilisations sont présentés dans [Jer 97].

1.4.3. Analyse des solutions existantes pour la spécification des systèmes électroniques

Ce paragraphe présente une évaluation des solutions existantes pour la spécification de systèmes électroniques, afin de trouver la solution la plus appropriée, qui respecte les exigences d'un flot de conception : la spécification des systèmes à tous les niveaux d'abstraction, la validation et le raffinement inter-niveaux.

Compte tenu de cet objectif, les critères de notre analyse seront [Jer 97]:

- La puissance d'expression – ce critère fixe la difficulté ou la simplicité de spécifier un système. Dans notre cas, nous considérons le modèle d'exécution et la communication comme les principales composantes de la puissance d'expression d'un langage.
- 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 plus intéressants dans notre cas sont la standardisation, la courbe d'apprentissage ainsi que les bibliothèques et les outils mis à la disposition par un langage.

Le Tableau 4 présente l'analyse des solutions pour la spécification des systèmes hétérogènes en fonction de ces critères.

Critère d'analyse	Solution pour la spécification	Approche basé sur l'utilisation d'un langage		Approche multi-langage
		Nouveau langage	Ajout d'une extension exécutable à un langage déjà existant	Utilisation de plusieurs langages de spécification et un modèle de représentation/coordination
Puissance d'expression	Modèle d'exécution	Restrictif	Illimité	Illimité
	Communication	Restrictif	Illimité	Illimité
Puissance d'analyse	Analyse Formelle	Facilitée	Non pratique	Facilitée
	Construction de nouveaux outils	Facile	Non pratique	Facile
Puissance commerciale	Standardisation	Difficile	Implicite	Solutions pratiques
	Courbe d'apprentissage	Longue	Implicite pour la syntaxe Longue pour la sémantique	Longue pour le langage de coordination
	Outils existants et bibliothèques	__	Oui pour l'exécution Non pour la synthèse/analyse	Oui pour l'exécution partiale Non pour la communication

Tableau 4. Analyse des solutions pour la spécification des systèmes électroniques

Un nouveau langage ou une extension syntaxique d'un langage déjà existant peut être ciblé pour faciliter l'analyse formelle et la construction de nouveaux outils. Cette solution est donc avantageuse du point de vue de la puissance d'analyse. Par contre, un tel choix entraîne généralement une puissance d'expression restrictive pour la partie calcul ainsi que pour la partie communication. Par exemple, dans le langage Rosetta [Sld 02], les règles de spécification utilisées permettent seulement l'utilisation des modèles de communication restrictifs. En plus ces restrictions sont perçues par l'usager. En analysant ce langage du point de vue de la puissance commerciale, plusieurs problèmes se posent : la standardisation est difficile à réaliser, la courbe d'apprentissage est longue, et évidemment un nouveau langage ne dispose pas des bibliothèques et des outils existants.

Une extension exécutable d'un langage existant est intéressante du point de vue de la puissance d'expression, du modèle d'exécution déjà existant, de la possibilité d'être simulé sans effort supplémentaire de la standardisation. Par contre, un tel langage est très difficile à synthétiser et à analyser. De plus, il nécessite une longue phase d'apprentissage pour la sémantique.

La solution apportée par l'approche multi-langage basée sur l'utilisation de différents langages de spécification et un modèle de représentation ou coordination présente aussi une puissance d'expression illimitée. Du point de vue de la puissance de commercialisation, les problèmes sont identiques à ceux des nouveaux langages. Ils sont difficiles à imposer comme standard. Mais encore, en vue de la spécification initiale du système, l'apprentissage du langage de coordination est nécessaire. L'utilisation d'un modèle de coordination est préférable pour le raffinement, l'analyse formelle étant accessible et permettant facilement la construction de nouveaux outils. Un autre avantage est de pouvoir faire évoluer la sémantique du modèle de coordination au fur et à mesure du raffinement du système.

1.5. Conclusion

La conception des systèmes électroniques nécessite la modélisation des concepts spécifiques aux différents niveaux d'abstraction, la validation et le raffinement. Actuellement un langage respectant toutes ces exigences n'existe pas.

Ce chapitre a présenté une étude des concepts fondamentaux utilisés tout au long du flot de conception des systèmes électroniques afin de pouvoir analyser les points forts et les points faibles des langages existants.

Une analyse des solutions proposées actuellement pour la spécification des systèmes hétérogènes a été effectuée. Cette analyse aboutit à la conclusion qu'une approche indépendante des langages de spécification est une solution attractive. Ainsi, un modèle de représentation liant

les différents composants du système peut être utilisé. Son avantage est de changer sa sémantique au fur est à mesure du raffinement.

Ce travail a guidé la définition d'un modèle de représentation que nous avons appelé Colif. Ce modèle sera présenté dans le chapitre suivant de ce mémoire.

Chapitre 2. Colif - Modèle de représentation pour la conception des systèmes hétérogènes embarqués

Sommaire

2.1. Introduction

L'analyse sur la spécification des systèmes effectuée dans le chapitre précédent a montré que le langage universel qui couvre tous les concepts nécessaires pour la spécification tout au long d'un flot complet de conception n'existe encore pas. Nous avons montré aussi que le recours à l'approche basée sur l'utilisation de plusieurs langages de spécification un modèle de représentation s'avère être la solution la plus attractive pour la conception des systèmes électroniques par raffinement à travers plusieurs niveaux d'abstraction. L'avantage le plus important d'un modèle de représentation est de pouvoir faire évoluer sa sémantique au fur et à mesure du raffinement.

La définition du modèle de représentation est une étape clé dans le processus de conception. La flexibilité, la modularité et l'extensibilité d'un flot de conception sont influencées par les propriétés du modèle de représentation utilisé. Plusieurs modèles existent aujourd'hui : ils sont bien adaptés pour la modélisation de différents modèles de calcul ou bien pour la modélisation de certains niveaux d'abstraction. Il serait désirable de disposer d'un modèle de représentation qui puisse fournir les concepts de tous les niveaux d'abstraction et de tous les modèles de communication.

Dans ce chapitre nous présenterons Colif, un modèle de représentation basé sur la séparation entre la communication et le comportement. Ce modèle de représentation couvre les différents niveaux d'abstraction et permet la spécification des systèmes complexes par une approche « diviser pour mieux régner ». La deuxième section de ce chapitre présentera la tendance actuelle dans la conception des systèmes hétérogènes embarqués et les modèles de représentation existants. La section suivante introduira Colif et les concepts de base utilisés pour sa définition. La quatrième section illustrera l'application de Colif dans un flot de conception pour les systèmes hétérogènes multiprocesseur embarqués; il s'agit du flot de conception défini dans le group SLS. La dernière section de ce chapitre donnera les conclusions.

2.2. La conception des systèmes hétérogènes embarqués et les modèles de représentation

Cette section présentera les besoins pour la conception des systèmes hétérogènes embarqués et fixera donc les pré-requis d'un modèle de représentation. Ensuite nous effectuerons un état de l'art sur les modèles de représentation proposés actuellement dans le monde de la conception. Pour clore, nous proposerons Colif comme une nouvelle solution pour la représentation des systèmes, solution qui satisfait les besoins d'un flot efficace pour la conception des systèmes hétérogènes embarqués.

2.2.1. Conception des systèmes hétérogènes embarqués

La Figure 4 illustre un flot générique de conception des systèmes électroniques. Ce flot traverse les niveaux d'abstraction présentés dans le Chapitre 1 de ce mémoire : le niveau service, le niveau transaction, le niveau macro-architecture et le niveau micro-architecture.

Le système à concevoir est initialement spécifié au niveau service, comme un ensemble de modules hiérarchiques communiquant par des réseaux abstraits. Après la validation de cette spécification de haut niveau, la première étape du flot de conception consiste à expliciter le réseau de communication, à fixer le routage des données et à adapter les interfaces des modules obtenant ainsi un nouveau modèle de niveau transaction. La validation de ce nouveau modèle est nécessaire.

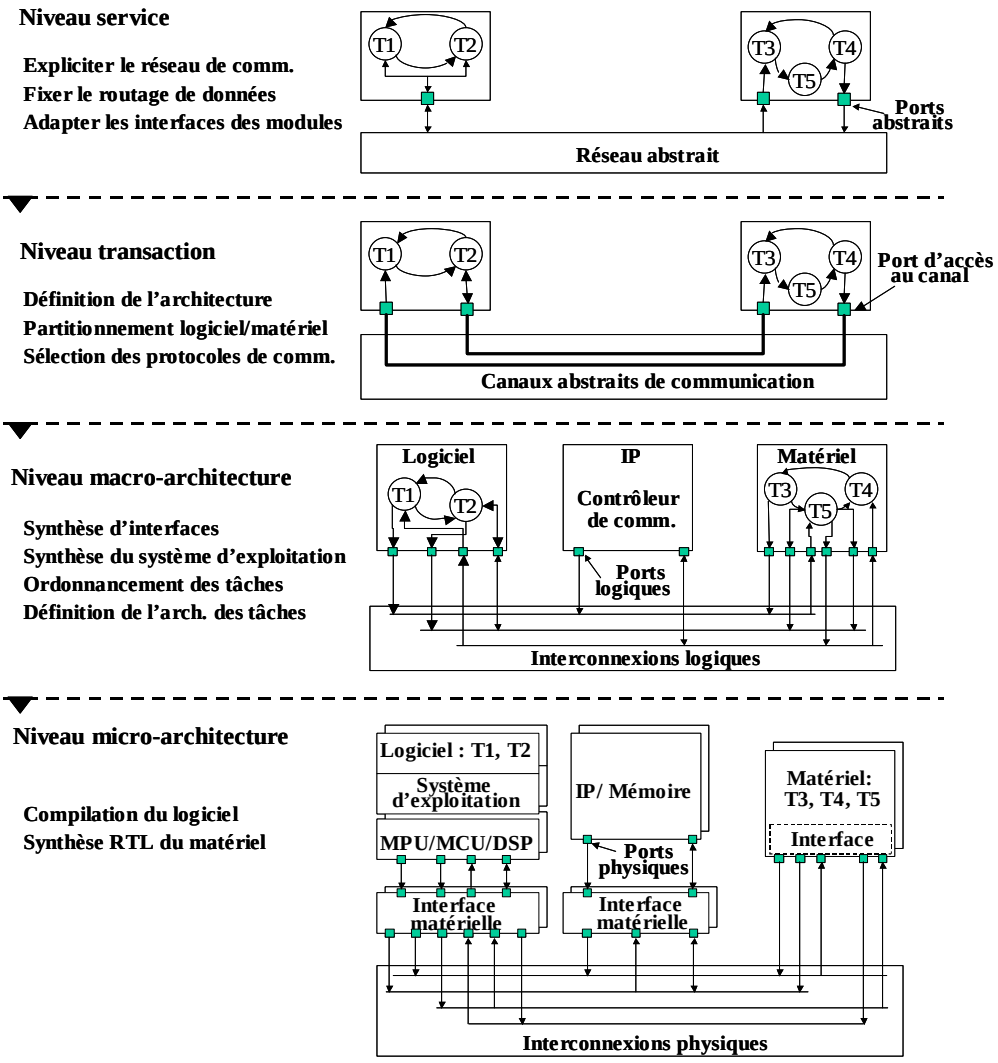


Figure 4. Flot générique pour la conception des systèmes hétérogènes embarqués

La seconde étape consiste à définir l'architecture, à réaliser le partitionnement logiciel/matériel [Cal 97] et à résoudre les protocoles des canaux actifs. Une étape de synthèse de protocole peut être nécessaire [Dav 97]. Cette étape peut introduire des nouveaux modules qui implémentent les comportements des canaux (contrôleurs de communication). Le modèle obtenu est une architecture abstraite au macro-architecture. Au niveau macro-architecture chaque module représente un processeur de l'architecture finale. Un tel processeur peut être réalisé par un processeur logiciel (ex. DSP ou microcontrôleur), un processeur matériel (ex. coprocesseur matériel) ou un module déjà existant (appelé IP venant de l'anglais 'Intellectual Property'). Les modules de cette architecture sont interconnectés via des canaux qui encapsulent des protocoles fixes. La validation du modèle de niveau macro-architecture est nécessaire.

Le raffinement d'une macro-architecture en une micro-architecture implique plusieurs étapes : la définition de l'architecture mémoire, le ciblage des modules logiciels sur des processeurs spécifiques ; cela nécessite la synthèse des interfaces matérielles qui permettent d'adapter la communication du processeur avec le reste de l'architecture. Si la partie logicielle contient des tâches parallèles et/ou des entrées/sorties parallèles, la synthèse d'un système d'exploitation et la définition d'une police d'ordonnancement des tâches est aussi nécessaire. Les modules matériels sont raffinés au niveau du cycle d'horloge. Dans l'architecture finale, les blocs déjà existants sont encapsulés par une interface pour l'adaptation du protocole final.

Dans ce flot générique de conception, nous avons considéré que tous les modules d'un système sont représentés au même niveau d'abstraction. En réalité, il faut souvent traiter des cas où les différentes parties du système peuvent être décrites à des niveaux d'abstraction différents ; un flot de conception efficace doit permettre le raffinement indépendant de la communication et du comportement. Ce type de raffinement est généralement connu sous le nom de raffinement incrémental. Dans le cas d'un tel raffinement plusieurs étapes intermédiaires où seulement une partie du système est raffinée sont ajoutées aux étapes illustrées dans la Figure 4.

La mise en place d'un tel flot générique de conception nécessite un modèle qui représente les systèmes à tous les niveaux d'abstraction et qui rend possibles toutes les étapes de raffinement illustrées dans la Figure 4 et les éventuelles étapes intermédiaires pour un raffinement incrémental. Le point clé est dans ce cas la séparation entre la communication et le comportement.

2.2.2. Etat de l'art sur les modèles de représentation utilisés pour la conception des systèmes

Très récemment l'approche de la spécification indépendante des langages de spécification à l'aide des modèles de représentation est de plus en plus utilisée. Open Verilog International propose un manuel référentiel pour la sémantique (Semantic Reference Manual) nommé OVI-SRM et qui décrit d'une façon indépendante des langages de spécification le mécanisme d'ordonnancement de différentes tâches dans un système embarqué multiprocesseur. Les différents processeurs communiquent par des mémoires partagées ; ce modèle est restreint à un seul niveau d'abstraction. Il est prévu que les efforts de l'OVI soient focalisés sur la représentation du matériel [Goe 99].

VSIA (The Virtual Socket Interface Alliance) propose un standard pour l'intégration des IP : des interfaces de niveau système pour le modèle de communication, un standard de modèle de performances pour les contraintes d'un système et un standard pour les types de données pour l'intégration de nouveaux composants dans un système. Bien que très flexible, ce modèle ne couvre pas tous les niveaux d'abstraction [Len 00].

L'approche MCSE [Cal 97] propose la représentation de niveau système à l'aide d'un modèle fonctionnel basé sur le modèle des processus communicants et d'un modèle non-fonctionnel exprimé sous forme des contraintes (temps, performances, technologies). Ce modèle de représentation est utilisé pour le partitionnement logiciel/matériel des systèmes.

GSRC (Gigascale Silicon Center) [Keu 99] propose une « syntaxe abstraite » qui est à la fois indépendante du langage et de la sémantique [Fer 99]. La syntaxe abstraite permet la spécification des netlists, des diagrammes de transition d'états, des diagrammes de blocs et de modèles d'objets. MoML [Lee 00] est un langage dérivé de XML[3] [XML00] qui utilise la syntaxe GSRC. MoML est utilisé pour représenter différents modèles de calcul [Lee 99] qui sont assemblés pour la simulation globale d'un système. Plusieurs autres approches s'articulent autour des modèles de calcul : les machines abstraites d'états finis pour la co-conception (Abstract Codesign Finit State Machine) [Sgr 00] modélisent la communication en utilisant des queues de taille infinie. El Greco [Buc 00] utilise le flot de données cyclo-statique (Cyclo-static Dataflow) [Par 95] et des FIFO de taille fixe pour la communication. SPIC (System Properties Intervals) est un modèle abstrait de réseau de processus avec des annotations de performances du système sous forme d'intervalles, et utilise des FIFO pour la communication [Ern 99].

Dans ce travail, nous présentons un modèle de représentation, Colif, qui supporte plusieurs modèles de communication spécifiques aux différents niveaux d'abstraction. La spécification des systèmes contenant des composants décrits aux différents niveaux d'abstraction ou en utilisant des langages différents de spécification est aussi supportée. Le concept central de Colif est la séparation entre la communication et le comportement.

2.3. Colif : modèle de représentation pour la spécification des systèmes électroniques

Cette section introduit les concepts de base de Colif. A la définition de ces concepts, l'objectif a été de pouvoir représenter les systèmes tout au long du flot de conception présenté dans la section 2.2.1. Cela implique la modélisation des concepts de base à tous les niveaux d'abstraction ainsi que la nécessité de disposer de nouveaux concepts permettant l'abstraction des interconnexion entre des composants hétérogènes tout en respectant la séparation entre la communication et le comportement. Pour ce faire nous avons introduit le concept d'architecture virtuelle qui représente les systèmes comme un ensemble de composants virtuels communicant par des canaux virtuels via des ports virtuels. Par la suite nous illustrerons premièrement l'utilisation de Colif à tous les niveaux d'abstraction et ensuite nous présenterons les nouveaux concepts permettant l'abstraction des interconnections entre des composants hétérogènes.

[3] Plus de détails sur XML peuvent être trouvés dans l'annexe A

2.3.1. Réutilisation des concepts conventionnels – module, interface, canal de communication

Dans notre modèle de spécification nous réutilisons les concepts de base conventionnels pour la spécification des systèmes (cf. section 1.2 du Chapitre 1 de ce mémoire) : Colif représente les systèmes comme un ensemble de modules hiérarchiques interconnectés. Comme la Figure 5 le montre, chaque module est défini par son interface (ensemble de ports) et son contenu. Le contenu d'un module est un ensemble d'autres modules ou un comportement élémentaire.

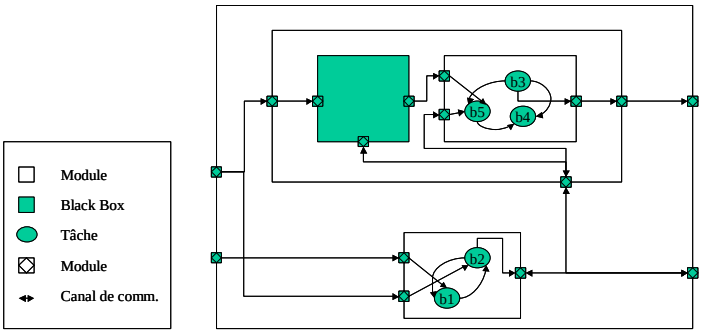


Figure 5. Réutilisation des concepts conventionnels en Colif

Nous allons illustrer à l'aide d'un exemple simple l'utilisation de Colif pour la modélisation des systèmes dans un flot de conception. L'application consiste en une tâche « compteur » et en quatre tâches, chacune échangeant des jetons (une structure de données) avec les autres tâches. Deux jetons circulant en directions opposées sont échangés entre les quatre tâches qui forment un anneau. Le comportement de chaque tâche peut être décrit en différents langages de spécification (par exemple SDL, VHDL, C/C++).

Nous considérons que le système est initialement spécifié au plus haut niveau d'abstraction, le niveau service et il est raffiné au niveau RTL à travers plusieurs niveaux d'abstraction.

Modèle de système au niveau service

La spécification de niveau service de l'application utilisée comme exemple est illustrée dans la Figure 6. La communication est réalisée par un réseau abstrait qui réalise dynamiquement le routage des services. Le réseau est accédé par des ports sur lesquels les tâches demandent des services et il connecte ces ports aux ports qui offrent les services correspondants (appelés ports de service). Le réseau détient des informations sur les services offerts par chaque tâche, le type des paramètres requis et le pas de routage.

Dans la Figure 6, le port de service P5 indique au réseau que la tâche T5 offre un service appelé **`Count`** et le port P2 indique que la tâche T2 offre le service **`GetToken2`** qui prend

comme paramètre l'objet **`tokendir`**. Les tâches demandent un service par son nom, en utilisant leurs ports. Le pseudo-code **`T2_token_task()`** illustre les demandes de services sur le port P2. Le service consiste en l'envoi du jeton à la tâche T3 quand la tâche T2 détient le jeton qui circule dans le sens des aiguilles d'une montre. Le pseudo-code pour la tâche **`T5.service(Count)`** montre qu'un compteur interne est incrémenté chaque fois que le service **`Count`** est appelé. Le pseudo-code pour **`T2.Service(GetToken2, tokendir)`** montre que le drapeau interne **`dtoken`** prend la valeur « vrai » chaque fois que le service **`GetToken2`** est appelé avec un paramètre **`direct`**. Chaque tâche exécute une boucle infinie mais elle laisse le contrôle à chaque appel de la fonction **`wait()`** et quand elle effectue des appels sur les ports d'entrée/sortie. Plusieurs stratégies pour la résolution des différents services peuvent être implémentées et stockées dans une bibliothèque de comportements pour le réseau abstrait.

Le raffinement de la communication de niveau service implique le remplacement du réseau abstrait par un réseau explicite composé de canaux actifs. Le concepteur choisit le comportement de chaque canal actif : FIFO infinie, boîte à messages, etc.

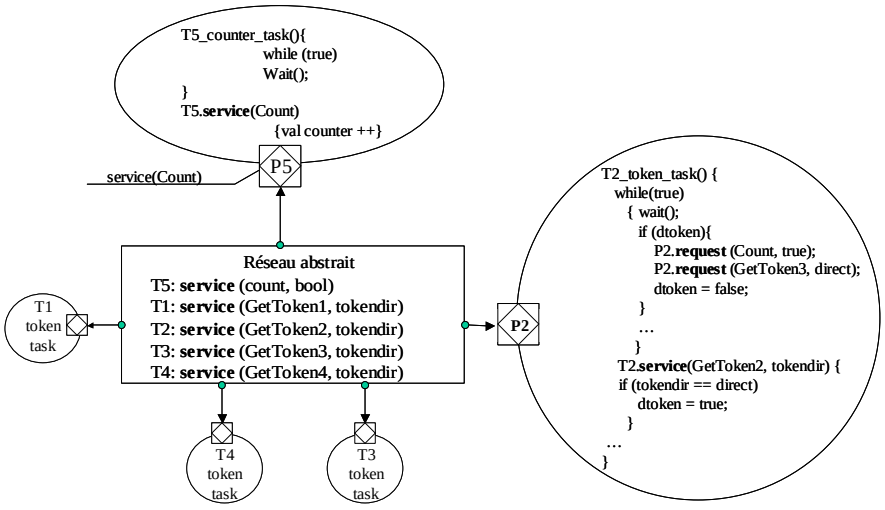


Figure 6. Modèle de niveau service en Colif

Modèle de système au niveau message

La Figure 7 montre notre application au niveau message où la communication est réalisée par des canaux actifs. Les tâches T1 et T2 sont groupées dans un même module. De même pour les tâches T3 et T4. Les différents ports sont les intermédiaires des appels des primitives de communication de haut niveau, **`send/receive`**. Les canaux actifs peuvent effectuer la conversion des protocoles manipulant des données de types génériques. Dans la Figure 7 **`T5_counter_task()`** appelle la primitive **`receive`** par l'intermédiaire du port P1 pour recevoir une valeur de type *bool*. Si la valeur de cette donnée est « vrai », le compteur interne est

incrémenté. Une fois que la tâche **`T2_token_task()`** a reçu le jeton de la tâche T1 ou T3 par l'intermédiaire du port P1, elle appelle la primitive **`send`** par l'intermédiaire du port P2 pour envoyer la valeur « vrai » à la tâche T5. Le comportement des canaux actifs est décrit dans une bibliothèque externe.

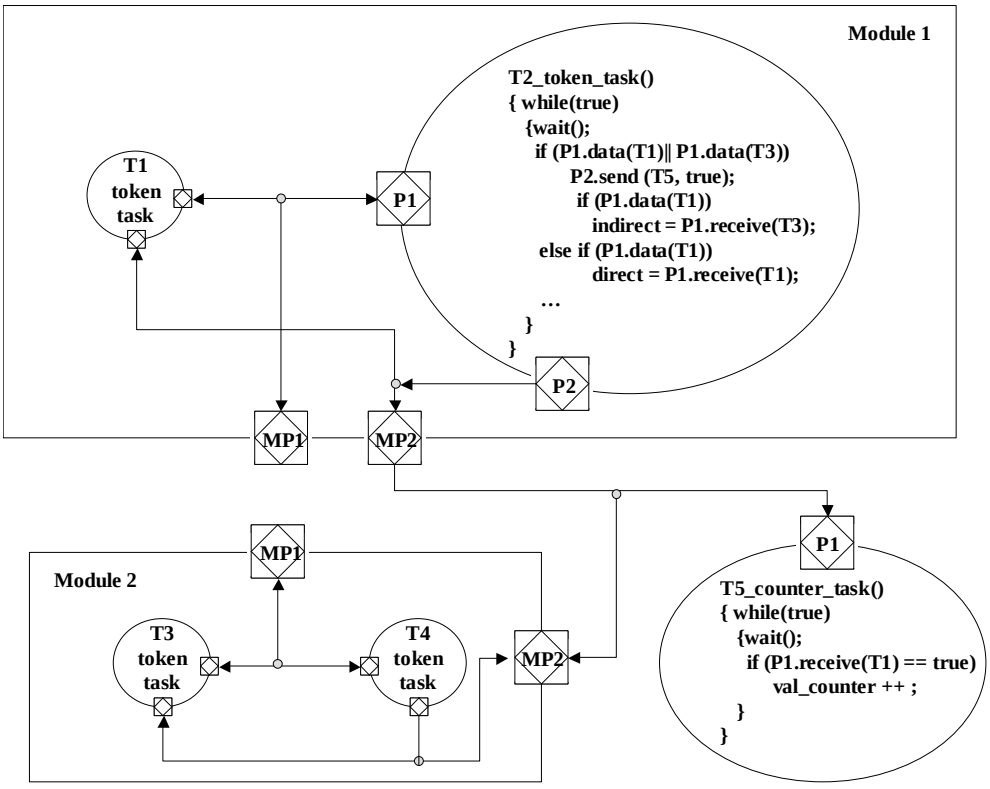


Figure 7. Modèle de niveau message en Colif

Le raffinement du modèle de niveau message en un modèle de niveau macro-architecture implique la définition de l'architecture et le partitionnement logiciel/matériel. Concernant la communication, la sélection des protocoles de communication et la définition des paramètres (par exemple la taille de la FIFO, écriture bloquante/non-bloquante, etc.) doivent être effectuées.

Modèle de système au niveau macro-architecture

Au niveau architecture, chaque module/tâche située au plus haut niveau de la hiérarchie correspond à un processeur ou un bloc matériel dans l'architecture finale du système. Par exemple, dans la Figure 8 le module 1 est un module logiciel, donc les tâches T1 et T2 s'exécuteront sur le même processeur. De même pour les tâches T3 et T4. Les canaux de communication sont des fils abstraits englobant un protocole de communication (par exemple rendez-vous). Ces fils abstraits cachent des détails tels que la gestion d'interruption et le décodage d'adresses. Les ports facilitent

l'accès aux fils abstraits. Par exemple, la tâche **`T1_token_task()`** de la Figure 8 obtient un jeton envoyé par la tâche T4 en utilisant le protocole FIFO et l'envoi à la tâche **`T2_token_task()`** en utilisant le protocole rendez-vous. Le comportement des canaux de communication connectant la tâche **`T5_counter_task()`** avec le reste du système est une simple transmission.

Le raffinement du modèle de niveau macro-architecture en un modèle de niveau micro-architecture implique la synthèse des interfaces et du système d'exploitation et la définition de la police d'ordonnancement des tâches.

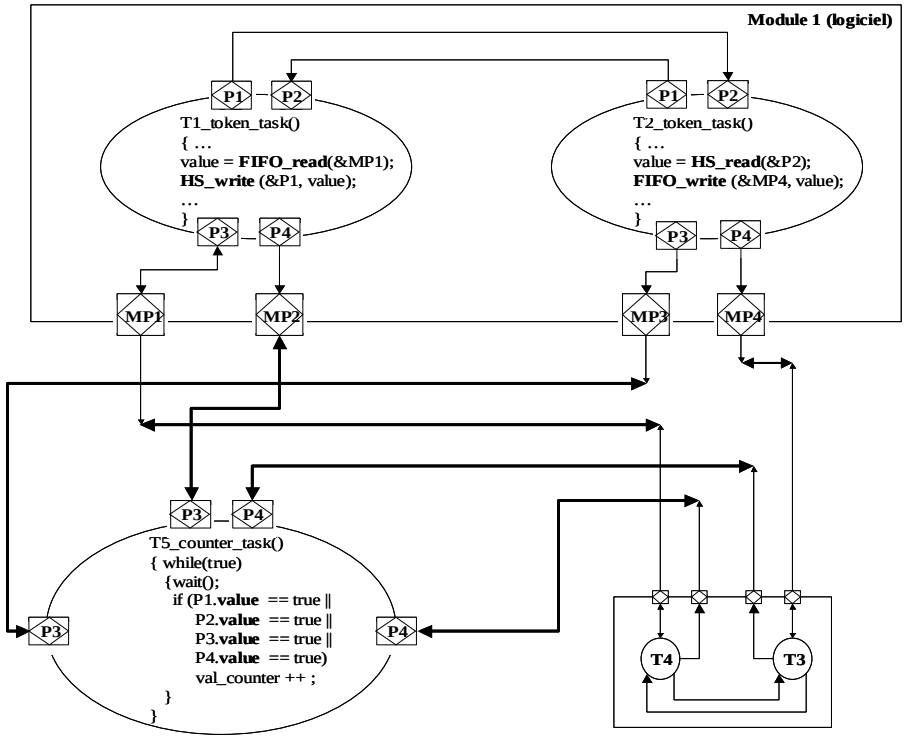


Figure 8. Modèle de niveau macro-architecture en Colif

Modèle de système au niveau RTL/micro-architecture

La Figure 9 illustre l'architecture de niveau RTL dans le cas de l'application étudiée.

Le module 1 a été ciblé sur un processeur (par exemple MC68000) qui exécute les tâches et le système d'exploitation. Le module 2 a été aussi ciblé sur un processeur (par exemple ARM7) et la tâche T5 est implémentée par un IP. Les interfaces matérielles ont été ajoutées pour connecter les différents processeurs au réseau de communication. Ces interfaces peuvent englober une logique assez simple comme dans le cas du bloc IP ou peuvent représenter des modules complexes qui incluent la gestion des interruptions, le décodage d'adresses, les contrôleurs de communication, etc. Les tâches logicielles exécutent des opérations d'entrée/sortie par des appels de primitives de haut

niveau (API) fournies par le système d'exploitation. Dans la Figure 9, la tâche **`T1_token_task()`** appelle la primitive **`P3_IN_FIFO_16`** pour obtenir le jeton de la tâche T4, en utilisant le protocole FIFO.

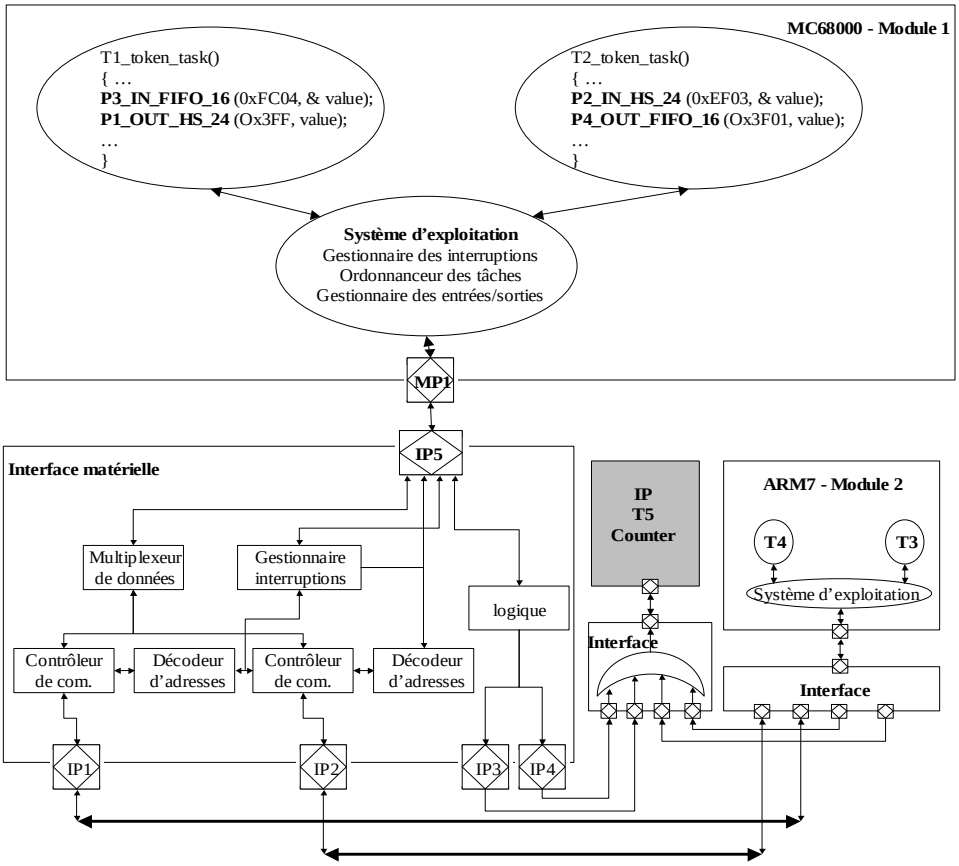


Figure 9. Modèle de niveau RTL en Colif

Les canaux de communication représentent des bus physiques transmettant des vecteurs de bits de taille fixée. Dans notre exemple, dans le nom des primitives le dernier numéro indique la taille du vecteur de bits ; le premier paramètre indique l'adresse associée au port de la tâche dans le processeur hôte.

2.3.2. La séparation de la communication et du comportement

Dans le cas d'une **spécification hétérogène** les différents modules et/ou canaux de communication peuvent être spécifiés à des niveaux d'abstraction différents ou à l'aide de langages de spécification différents. Afin de permettre la spécification des systèmes hétérogènes nous utilisons des nouveaux concepts en plus des concepts conventionnels (module, port, canal de communication).

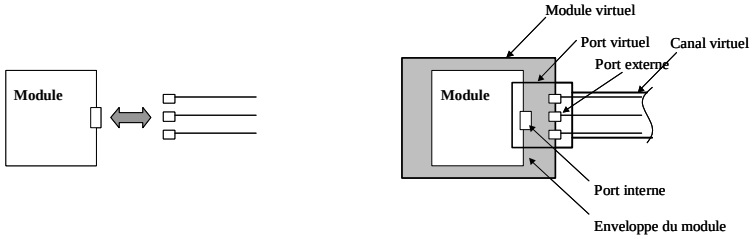
Un premier concept de base introduit pour séparer le comportement de la communication

dans la spécification des systèmes électroniques est **l'enveloppe d'un module**. Ce concept représente l'abstraction de l'interconnexion entre deux composants hétérogènes et permet la séparation entre le comportement du module et la communication. L'enveloppe est un ensemble de deux types de ports :

- Les ports internes – les ports du module,
- Les ports externes – les ports qui permettent la connexion avec les canaux de communication.

Les différents ports internes et externes d'une enveloppe sont mis en correspondance en les groupant dans des **ports virtuels**. Dans un port virtuel nous pouvons avoir *m* ports internes et *n* ports externes. (n, m sont des entiers naturels). Les ports internes et externes d'un port virtuel peuvent être spécifiques aux niveaux différents d'abstraction ou aux différents langages de spécification.

Un module et son enveloppe forment un **composant virtuel**. Les différents canaux de communication connectés à un port virtuel peuvent être groupés dans des **canaux virtuels**. La Figure 10 illustre un exemple d'utilisation des concepts de base présentés. La Figure 10.a montre un module qui doit être interconnecté avec des canaux de communication de niveau d'abstraction différent (par exemple un module de niveau macro-architecture qui doit être connecté à des signaux physiques). La Figure 10.b montre comment cette interconnexion entre ces composants hétérogènes a été abstraite et spécifiée à l'aide de concepts de port, canal et composant virtuel.



(a) Exemple d'interconnexion des composants hétérogènes

(b) Spécification de l'interconnexion hétérogène

Figure 10. Concepts de base pour la spécification hétérogène

En utilisant les concepts présentés, un système hétérogène peut être modélisé par une **architecture virtuelle** comme un ensemble de composants virtuels interconnectés. Les ports internes et externes des enveloppes des modules peuvent être de natures différentes (en termes de langages de spécification, niveaux d'abstraction et protocoles de communication). Les enveloppes représentent donc une abstraction d'un comportement qui est nécessaire en vue du raffinement ou de l'exécution de l'architecture virtuelle. Pour cela, l'architecture virtuelle doit permettre l'annotation avec des paramètres de configuration. La Figure 11 illustre un exemple d'architecture virtuelle.

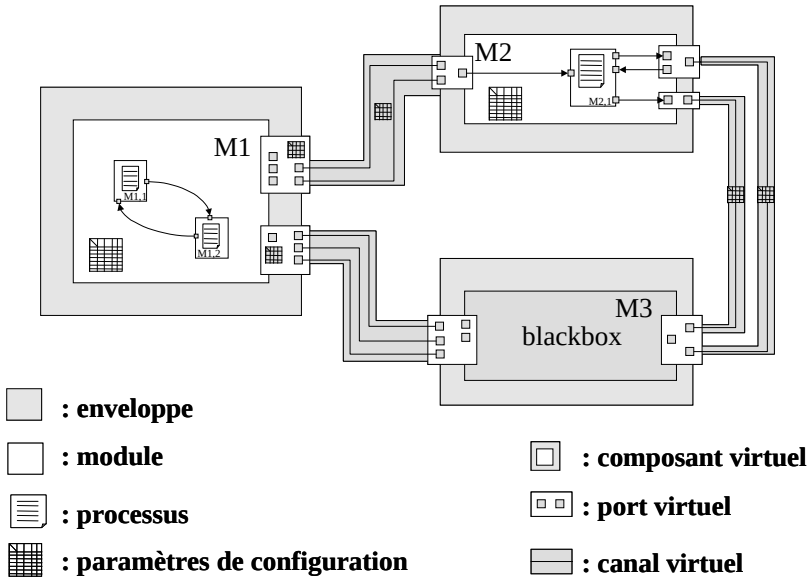


Figure 11. Modèle d'architecture virtuelle

2.4. L'environnement génie logiciel Colif

L'environnement génie logiciel Colif est le résultat d'un travail de groupe qui dépasse le cadre de cette thèse. Juste un aperçu sera donné dans cette section. Pour plus de détail, le lecteur peut consulter [Ces 01] et [Gaut 01].

2.4.1. Le modèle d'objets Colif

Colif utilise une syntaxe uniforme pour représenter les systèmes hétérogènes impliquant plusieurs niveaux d'abstraction ou langages de spécification. Chaque module est défini par son interface (qui consiste en un ensemble de ports) et son contenu. Le contenu d'un module est une liste des instances de modules ou un comportement. Les modules sont représentés comme des boîtes blanches s'ils contiennent d'autres instances ou comme des boîtes noires si leur contenu est non relevant ou inconnu.

La Figure 12 illustre une représentation simplifiée du diagramme des classes Colif, en utilisant des notations UML. Dans ce diagramme, chaque rectangle correspond à une classe et chaque flèche avec losange correspond à l'agrégation[4]. Le nom de chaque classe est inscrit dans la partie haute de chaque rectangle correspondant et les attributs sont inscrits au-dessous du nom de la classe (pour la simplicité de l'explication nous avons omis spécialement les méthodes et d'autres détails). Par exemple, le champ *ports* de la classe MODULE_ENTITY est un agrégat qui peut contenir un nombre quelconque d'instances (0..*) de la classe PORT_DECL. Par contre, un

[4] Relation entre classes signifiant que les instances d'une classe sont les composants d'une autre classe. L'agrégation permet de composer des objets pour en créer de plus complexes.

PORT_DECL est contenu par 0 ou 1 (0..1) instances de MODULE_ENTITY. Le diagramme des classes Colif illustre l'inclusion par les losanges grisés ; les losanges blancs indiquent une forme plus faible d'agrégation (par référence).

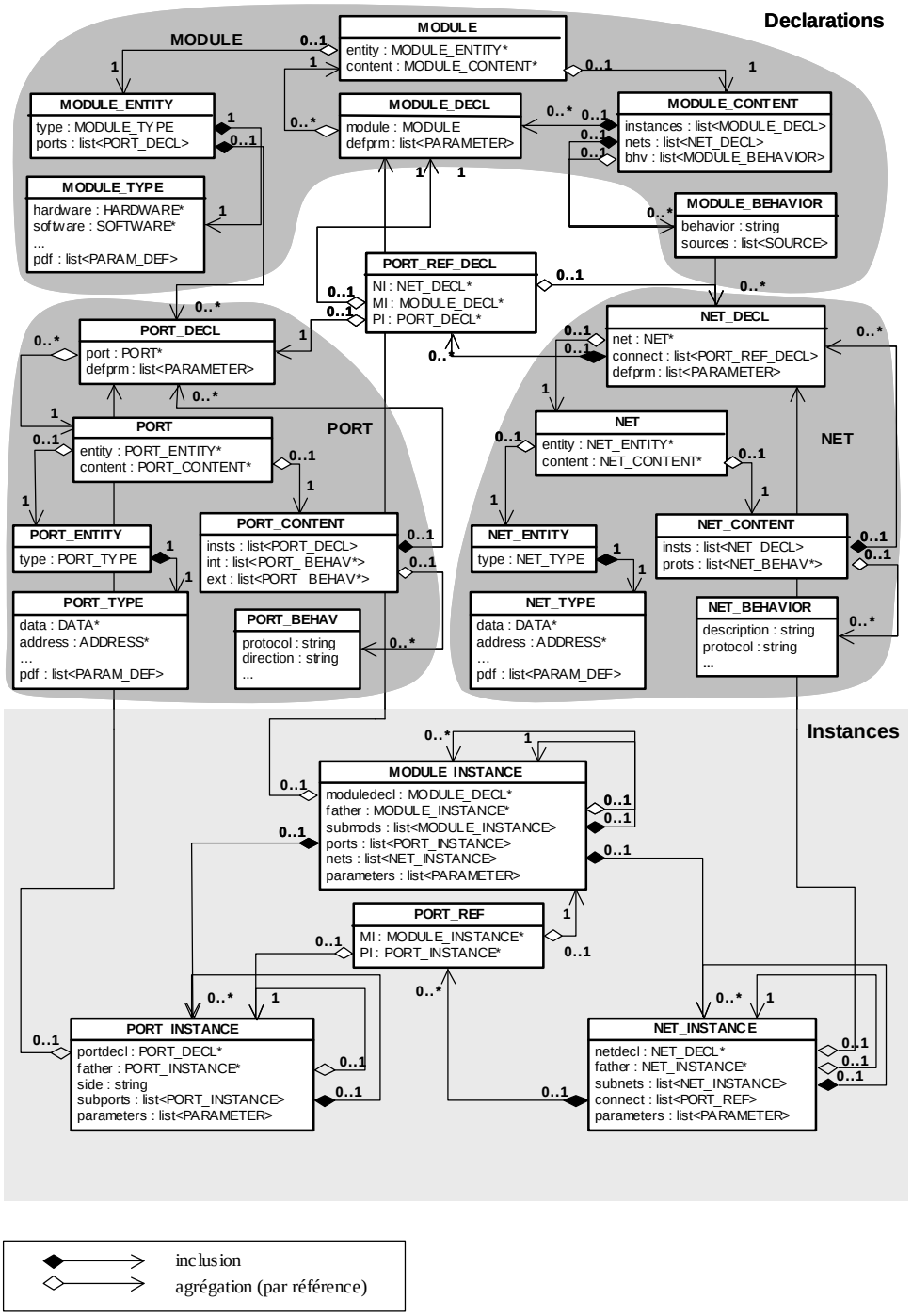


Figure 12. Diagramme de classes Colif

Comme la Figure 12 le montre, le modèle d'objets Colif est divisé en deux parties principales: une partie déclarative et une partie d'instanciation. La partie déclarative représente des objets définissant un modèle (template) réutilisable avec des définitions des propriétés génériques (PARAM_DEFS) et des valeurs par défaut pour certaines propriétés (PARAMETER). La partie d'instanciation représente les objets utilisés pour la construction d'un arbre d'instances.

Comme principe général, chaque concept de base de Colif – MODULE, PORT, NET est divisé en deux parties : une interface (ENTITY) et le contenu (CONTENT). L'entité a un type (TYPE) qui encapsule des propriétés définissables par l'utilisateur. Le contenu CONTENT contient une référence vers un comportement (interne ou externe) défini et/ou une liste des déclarations DECL d'objets. Dans ce dernier cas, l'objet CONTENT est hiérarchique (il contient des objets situés à un niveau plus bas de la hiérarchie). Une INSTANCE est une spécialisation de la définition d'un objet ; elle hérite toutes les propriétés de l'objet mais chaque instance peut avoir un ensemble de propriétés spécifiques. Une instance a une référence vers l'objet réutilisé et un nom qui peut être choisi par l'utilisateur. Les classes Colif sont polymorphes : leur sémantique change avec le niveau d'abstraction considéré.

La classe MODULE est un modèle (template) pour une description structurelle ou comportementale. La classe PORT définit l'interface entre le comportement du module et les canaux de communication internes. Comme nous l'avons expliqué dans le chapitre 2 de ce mémoire, les ports Colif peuvent être de deux types : les ports internes (spécifiques au module) et les ports externes (spécifiques aux canaux de communication). En conséquence, une instance de port, PORT_INSTANCE hiérarchique est composée des ports internes et externes et agit comme un adaptateur entre le contenu du module et le monde extérieur.

La classe NET représente le média de communication entre différents ports ; son comportement peut être décrit aux différents niveaux d'abstraction. Le concept de canal de communication joue un rôle très important en Colif. Selon le niveau d'abstraction, un canal de communication peut englober un comportement très complexe.

La flexibilité de Colif réside principalement dans la structure hiérarchique des modules, ports et canaux de communication. Cette flexibilité est très importante dans le processus de conception, et elle a été notre objectif prioritaire. Pour cette flexibilité nous avons payé le prix d'un manque d'analyse formelle, qui requièrt des sémantiques d'exécution bien définies. Notre choix est justifié par la nature du problème que nous voulons résoudre – la synthèse et la conception des systèmes hétérogènes embarqués. Généralement, les concepteurs divisent la spécification en plusieurs parties et conçoivent les systèmes d'une manière modulaire. Dans ce cas, ils peuvent appliquer des

sémantiques d'exécution bien définies et l'analyse formelle pour chaque sous-système et utiliser Colif pour la modélisation du système global.

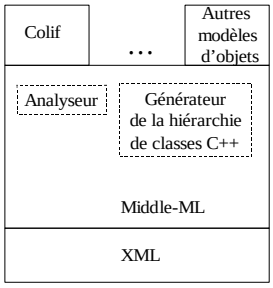
2.4.2. Détails d'implémentation

Pour la réalisation de Colif nous avons utilisé le langage XML (Extensible Markup Language) **[XML00]**. XML, métalangage standardisé par W3C, connaît aujourd'hui un grand succès dans le monde de l'informatique grâce à sa souplesse, sa clarté et sa facilité à être analysé. Il facilite l'intégration des outils et l'échange d'informations entre les différents environnements de synthèse.

XML est basé sur le concept de balise : toute information est délimitée par une balise ouvrante et une balise fermante. Les balises XML peuvent avoir des attributs et peuvent être hiérarchiques. Pour obtenir un langage dérivé de XML il faut définir des balises particulières et préciser les contraintes qui leur sont associées. Ces contraintes peuvent concerner leurs attributs ou à propos de la hiérarchie. Ces informations sont à donner dans un fichier à part nommé *dtd*. Ce langage possède deux caractéristiques intéressantes pour le but fixé : c'est un méta-langage qui grâce au système de *dtd* n'a besoin que d'un seul analyseur quel que soit le langage dérivé et c'est un standard recommandé pour la représentation de données. XML est cependant plus une notation qu'un langage, en particulier il ne dispose pas de sémantique.

Nous aurions pu utiliser XML et les structures de données associées, mais pour des raisons d'efficacité nous avons décidé d'écrire un analyseur spécialisé et de développer des structures de données en mémoire afin de pouvoir manipuler des systèmes très complexes. Ainsi, nous utilisons Middle [Gaut 01], un langage dérivé de la notation XML ; ce langage permet de définir des structures de données complexes avec une sémantique associée. Middle est lui-même un méta-langage à partir du quel a été dérivé Colif. La Figure 13.b illustre les couches de langages utilisés pour l'implémentation de Colif.

La Figure 13.b illustre un extrait de fichier Colif. La ligne 2 montre que nous utilisons la grammaire Middle (voire Figure 13.a); à l'aide de Middle les concepteurs peuvent décrire des modèles complexes des systèmes. La ligne 5 commence la description d'un nouveau type appelé MODULE et énonce sa structuration comme un ensemble de champs nommés. La ligne 9 déclare un champ appelé *entity* et la ligne 11 montre que le type de la structure référencée est MODULE_ENTITY.




```
1.  <? Xml version = "11.00 " encoding = "UTF-8"? >
2.  <!DOCTYPE middle SYSTEM "middle.dtd " >
3.  <middle>
4.  …
5.   <typedef name = "MODULE" structure = "set" >
6.     <typefield name = "description" >
7.       <typeref base = "str"/ >
8.     </typefield>
9.     <typefield name = "entity" >
10.      <typedef structure = "ref" >
11.        <typeref base  = "MODULE_ENTITY" />
12.      </typedef>
13.    </typefield>
14.    <typefield name = "content" >
15.      <typedef structure = "ref" >
16.        <typeref base  = "MODULE_CONTENT" />
17.      </typedef>
18.    </typefield>
19.  </typedef>
20.  …
21.  <middle>
```

(a) **(b)**

Figure 13. Détails pour la réalisation de Colif

Extrait d'un fichier (a) ; Couches de langages utilisés pour la réalisation (b)

2.5. Application de Colif dans un flot de conception des systèmes hétérogènes multiprocesseur basé sur l'assemblage des composants logiciels/matériels

Colif a été adopté dans le flot de conception des systèmes hétérogènes multiprocesseurs développé dans le groupe SLS. Ce flot permet la conception des systèmes hétérogènes multiprocesseurs par l'assemblage des composants logiciels/matériels. L'objectif et de générer automatiquement la réalisation de niveau RTL (ou la micro-architecture) du système à partir d'une architecture abstraite qu'on appelle macro-architecture.

Une première partie de cette section introduira les niveaux d'abstraction définies dans le flot de conception. La deuxième partie présentera la micro-architecture générique des systèmes hétérogènes embarqués proposée par le groupe SLS. La troisième partie donnera la vue conceptuelle du flot de conception et pour finir la quatrième partie expliquera la stratégie utilisée pour la génération automatique de l'architecture de niveau RTL pour les systèmes hétérogènes multiprocesseurs.

2.5.1. Niveaux d'abstraction dans le flot de conception des systèmes hétérogènes multiprocesseur

Par rapport au flot de conception idéal que nous avons présenté dans la section 2.2.1, le flot de conception du groupe SLS part d'une spécification de niveau macro-architecture du système le résultat étant une spécification de niveau RTL (ou une micro-architecture).

Dans cette section nous présenterons le flot de conception du groupe SLS en terme des niveaux d'abstraction. Nous définirons non seulement les niveaux d'abstraction pour la

communication mais encore les niveaux d'abstraction des composants logiciels et matériels pendant le raffinement d'une macro-architecture dans une réalisation de niveau RTL.

Ces niveaux sont illustrés dans la Figure 14 et ils seront présentés par la suite.

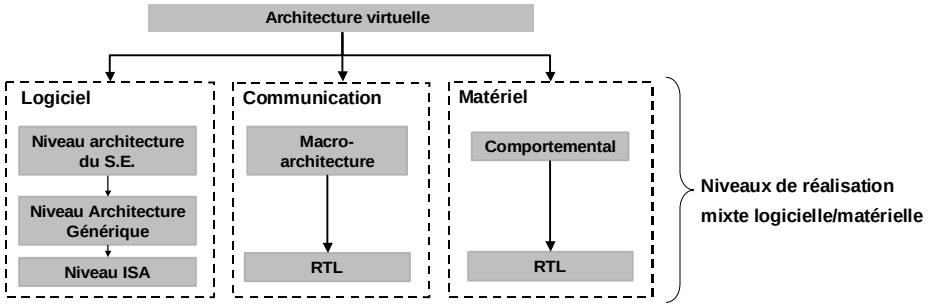


Figure 14. Niveaux d'abstraction pour la spécification des systèmes hétérogènes embarqués

Niveaux d'abstraction pour la réalisation des composants matériels

Pour les composants matériels on peut considérer plusieurs classes couvrant les éléments analogiques, numériques, optiques, mécaniques, etc. Plusieurs études ont déjà été effectuées sur les niveaux d'abstraction des composants matériels [Gaj 00].

Les niveaux d'abstraction dans le cadre de notre flot de conception sont le niveau comportemental et le niveau RTL :

- Le **niveau comportemental** la fonctionnalité du composant matériel est fixée mais aucune hypothèse n'est faite sur l'implémentation physique ; la communication avec le reste du système est réalisée en utilisant de primitives de communication de haut niveau.
- Au **niveau RTL** tous les détails de l'implémentation physique y compris les détails d'implémentation des protocoles de communications sont pris en compte ; les interfaces sont décrites par des ports physiques.

Niveaux d'abstraction pour la réalisation des composants logiciels

Pendant le processus de conception, les composants logiciels peuvent aussi être spécifiés à plusieurs niveaux d'abstraction :

- Au niveau **architecture du système d'exploitation** seuls les appels système correspondant aux services fournis par le système d'exploitation sont fixés, mais la réalisation de ces services reste encore abstraite. Dans le code de l'application des appels système sont utilisés pour la communication.
- Au **niveau architecture générique**, l'implémentation des services du système d'exploitation est fixée mais les détails de l'architecture matérielle sont encore abstraits, ainsi le matériel (processeur, mémoires, etc.) peut varier. Le code de l'application est

donc étendu par les couches du système d'exploitation réalisant les services tels que la gestion de tâches, la gestion d'interruptions et la gestion d'entrées/sorties génériques.

- Au **niveau jeu d'instructions** (**Instruction Set Architecture** ISA), les paramètres matériels sont fixés. Le logiciel est réalisé en code assembleur. Les détails internes du processeur ne sont pas pris en compte.

Niveaux d'abstraction pour la communication

Concernant la communication, deux niveaux d'abstraction sont utilisés : le niveau macro-architecture et le niveau micro-architecture. Ces niveaux ont été présentés dans le Chapitre 1 de ce mémoire. Elles seront présentés brièvement pour la clarté :

- Le **niveau macro-architecture** est le niveau spécifique à la communication par des canaux de communication permettant d'abstraire les protocoles de communication.
- Au **niveau RTL ou micro-architecture** la communication est réalisée par des fils et des bus physiques.

2.5.2. Modèle d'architecture cible pour les systèmes hétérogènes multiprocesseurs embarqués

La Figure 15 illustre l'architecture cible typique d'un système hétérogène multiprocesseur où les différents processeurs sont connectés au réseau de communication.

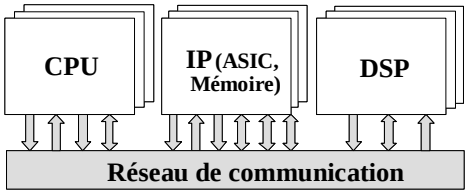


Figure 15. Architecture typique d'un système hétérogène multiprocesseur

Le logiciel est généralement organisé sous forme de couches, comme une abstraction du matériel. Cette organisation est illustrée dans la Figure 16 :

- La couche basse, la couche d'abstraction du matériel (HAL[5]), fournit les pilotes et les contrôleurs pour la gestion de la communication (par exemple pour la gestion d'interruptions).
- La couche des services du système d'exploitation offre les fonctionnalités de base pour faire tourner les applications utilisateur ou système, ainsi que pour gérer d'une manière efficace les ressources matérielles sous-jacentes.

[5] venant de l'anglais Hardware Abstraction Layer

§ La couche API (venant de l'anglais Application Programming Interface) représente les services (appels système) de haut niveau invoqués par l'application.

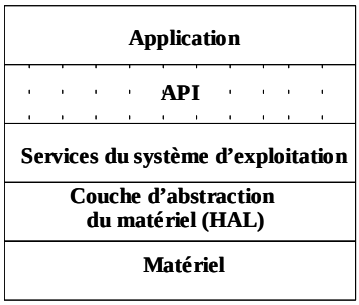


Figure 16. Organisation en couches du logiciel

Le groupe SLS utilise une architecture cible générique pour les systèmes hétérogènes multiprocesseurs [Bag 01]. Ce modèle peut être adapté aux besoins spécifiques d'une application ; la communication ainsi que le calcul peuvent être adaptés. Pour l'adaptation du calcul, le nombre de composants de l'architecture et leurs types peuvent être changés. Pour l'adaptation de la communication, une topologie spécifique à l'application peut être sélectionnée. Ce modèle d'architecture se propose de couvrir un domaine vaste d'applications.

L'architecture générique est illustrée dans la Figure 17. Les différents processeurs sont connectés au réseau de communication via des interfaces. En effet, les processeurs sont séparés du réseau de communication par des interfaces qui agissent comme des adaptateurs pour la communication. Une telle séparation est nécessaire pour décharger les processeurs de la gestion de la communication et pour permettre l'exécution parallèle des différentes tâches et des différents protocoles de communication.

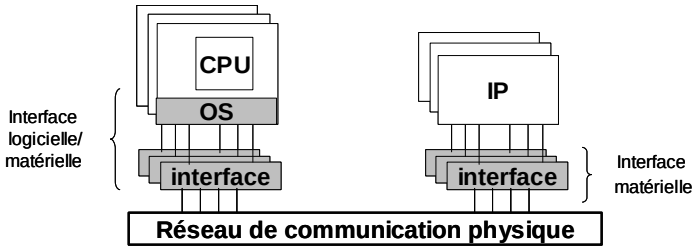


Figure 17. Modèle d'architecture générique pour les systèmes hétérogènes multiprocesseur embarqués

La Figure 18 montre l'architecture interne explicite des interfaces logicielles et matérielles.

L'interface matérielle est composée de deux parties : une partie spécifique au processeur se dénommant adaptateur de processeur (PA) et une partie spécifique aux communications, qui se compose de sous-éléments, les adaptateurs de canal[6] (CA). L'interconnexion de ces deux parties

[6] Le nombre d'adaptateurs de canal est donnée par le nombre des canaux virtuels connectés au processeur

nécessite un nouvel élément, le bus de communication interne. L'adaptateur du processeur traduit le protocole natif du bus du processeur en un protocole unifié pour le protocole du bus interne. Chaque adaptateur de canal est le responsable du contrôle du protocole de communication externe.

L'interface logicielle est structurée sous forme de couches de services : la couche de services de haut niveau (les API) appelés dans le code de l'application, la couche de pilotes pour le contrôle du matériel et finalement si nécessaire, l'interface contient aussi une couche de services plus sophistiqués tels que l'ordonnancement des tâches et la gestion d'interruptions et des entrées/sorites parallèles.

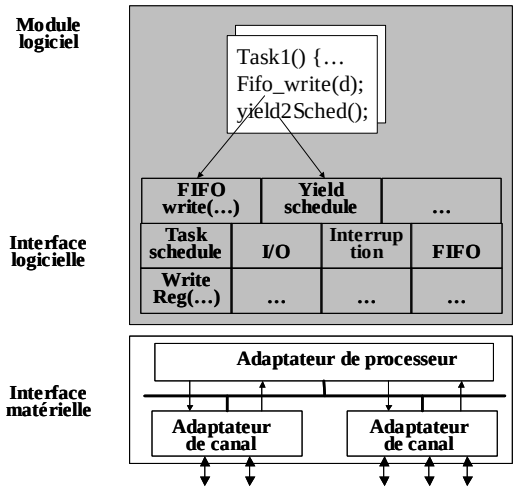


Figure 18. Architecture interne d'une interface logicielle/matérielle

2.5.3. Le flot de conception des systèmes hétérogènes multiprocesseurs embarqués

Vue conceptuelle

La vue conceptuelle du flot de conception des systèmes hétérogènes multiprocesseur embarqués est illustrée à la Figure 19. L'entrée du flot de conception, la macro-architecture, peut être écrite manuellement ou elle peut être générée automatiquement en utilisant des outils d'analyse des systèmes (par exemple VCC [Cad 02], MCSE [Cal 97]).

Comme la Figure 19 le montre, nous représentons la macro-architecture du système à l'aide des concepts Colif par une architecture virtuelle comme un ensemble des modules virtuels communicant par des canaux virtuels via des ports virtuels. A partir de cette architecture, la représentation de niveau RTL est obtenue par la génération des interfaces logicielles/matérielles qui constitue en fait l'implantation des enveloppes des modules virtuels. Les canaux de communication virtuels sont transposés sur un réseau de communication spécifique.

Les concepts Colif de module virtuel, port virtuel et canal virtuel permet dans le cas de notre flot l'abstraction des interfaces logicielles/matérielles et du réseau de communication de niveau RTL. Afin de permettre la génération de la représentation de niveau RTL, les ports et les canaux virtuels composant l'architecture virtuelle sont annotés par des attributs modélisant les décisions de raffinement. Ces annotations permettent de spécifier les ressources de calcul et de communication à mettre en œuvre au sein de la réalisation de niveau RTL. Il est important de mentionner que ce flot de conception permet le raffinement indépendant des différents composants. Dans ce cas, les concepts de module, port et canal virtuel nous permettent l'abstraction des interconnexions entre des composants hétérogènes, situé à des niveaux d'abstraction différents. Dans ce cas, des annotations pour la simulation sont nécessaires. Ces annotations permettent de spécifier les adaptations en vue de la simulation d'un système hétérogène.

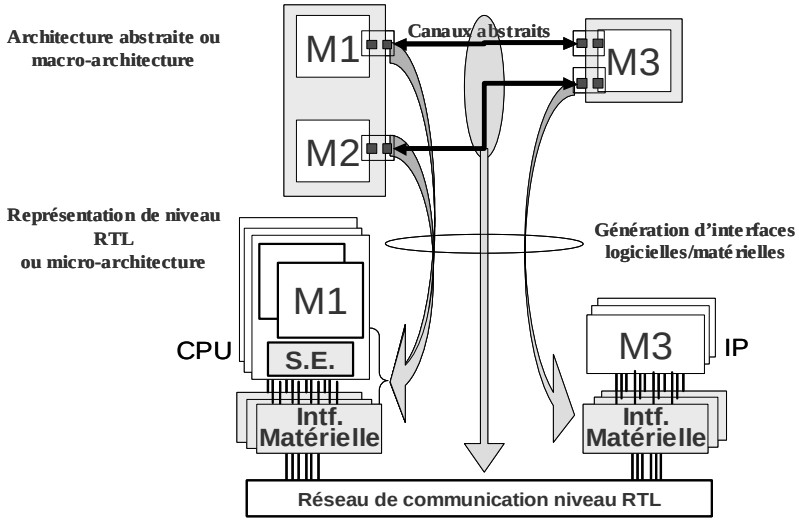


Figure 19. Vue conceptuelle du flot de conception des systèmes hétérogènes multiprocesseur embarqués

Génération automatique de la réalisation de niveau RTL d'une macro-architecture

Pour la génération automatique de la réalisation de niveau RTL d'une macro-architecture, un environnement de conception a été développé dans le groupe SLS. Cette environnement est illustré dans la Figure 20. Il comporte trois outils de génération automatique : un outil pour la génération des modèles de cosimulation, un outil pour la génération des interfaces logicielles et un outil pour la génération des interfaces matérielles. Tous ces outils repose sur Colif comme modèle interne pour la représentation des systèmes.

Une première représentation interne en Colif est obtenue par la translation de la spécification de l'architecture virtuelle annotée avec des paramètres de configuration. Cette spécification est

réalisée à l'aide du langage SystemC [Sys 00], langage que nous avons étendu pour ajouter les concepts de module, canal et port virtuel.

Dans un premier temps, l'**outil de génération de modèles de simulation** est utilisé pour la validation de l'architecture virtuelle. Comme il a été dit plus haut, les enveloppes des différents modules peuvent abstraire les interconnexions entre des composants hétérogènes. Afin de permettre la simulation de l'architecture virtuelle, l'outil de génération des modèles de simulation génère le modèle de simulation des telles enveloppes. La structure des modèles de simulation ainsi que la méthodologie de génération automatique des modèles de simulation font l'objet du Chapitre 3 de ce mémoire.

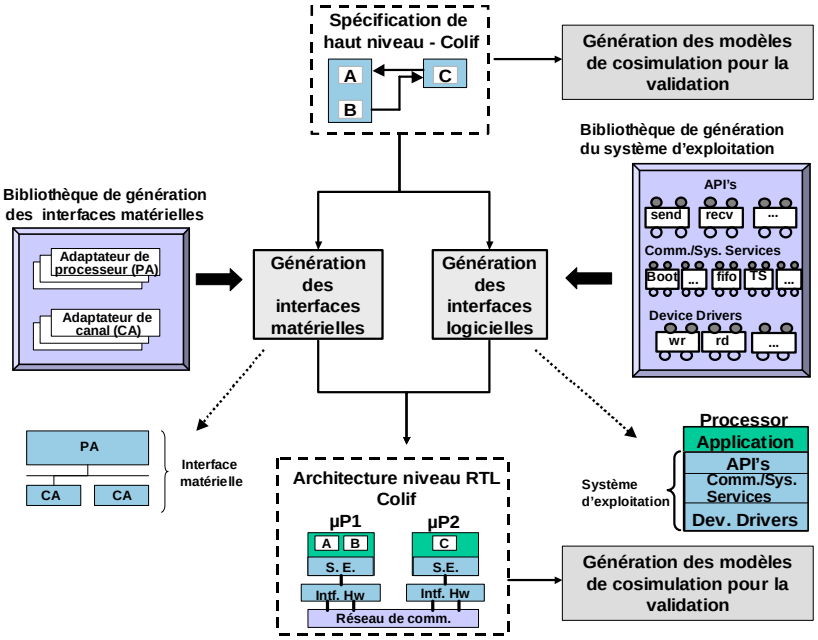


Figure 20. Environnement de conception utilisant Colif comme format intermédiaire

L'outil de génération automatique des interfaces logicielles génère des systèmes d'exploitation spécifiques et pré-configurés pour chaque module logiciel s'exécutant sur un processeur. Pour ce faire, cet outil extrait tous les annotations concernant l'implantation des interfaces logicielles qui sont stockés dans la représentation Colif de la macro-architecture. Il utilise une bibliothèque structurée en trois parties : les primitives de haut niveau (les API), les services de communication/système, et les paquetages pour la communication avec le matériel. Chaque partie de la bibliothèque contient des éléments qui seront utilisés dans une couche du système d'exploitation généré. Ce mécanisme est utilisé pour optimiser la taille du système d'exploitation, en évitant la présence des éléments inutilisables dans le système d'exploitation. Des détails sur cet outil peuvent être trouvés dans [Gau 01]. Cet outil sera cité aussi dans le Chapitre 4 de ce mémoire

qui illustrera son utilisation pour la génération des modèles de simulation pour les systèmes d'exploitation.

L'outil de génération des interfaces matérielles reçoit en entrée la représentation en Colif de la macro-architecture ; il est en charge de la génération des interfaces matérielles ainsi que de la représentation Colif de l'architecture de niveau RTL. Il utilise une bibliothèque composée de deux types d'éléments : des adaptateurs de processeurs (PA) et des adaptateurs de canal (CA). Cette organisation de la bibliothèque est motivée par la structure interne des interfaces matérielles à générer (cf. section 2.5.2). Ainsi, les interfaces matérielles sont générées automatiquement par assemblage des éléments de la bibliothèque. Tous les informations sur la topologie du système, les paramètres de raffinement, les protocoles de communication et les types de processeurs sont extrait de la représentation Colif de l'architecture virtuelle. Des détails sur cet outil peuvent être trouvés dans [Lyo 01].

Après la génération des interfaces logicielles et matérielles, une représentation interne en Colif de l'architecture de niveau RTL est obtenue. L'outil de génération des modèles de simulation est utilisé à nouveau pour la validation de cette représentation. Cela permettra la simulation globale où le logiciel s'exécute sur un simulateur du jeu d'instructions d'un processeur (ISS) ou en natif sur la machine hôte et le matériel s'exécute sur un simulateur de modèles matériels (par exemple SystemC ou un simulateur du langage VHDL).

2.6. Conclusions

Nous avons présenté un modèle de représentation pour la conception de systèmes hétérogènes. Ce modèle couvre les différents niveaux d'abstraction pour la communication et permet l'abstraction des interconnexions entre des composants hétérogènes.

Le concept central est la séparation entre la communication et le comportement. Chaque composant hétérogène est enveloppé dans une enveloppe qui sépare la communication et le comportement et qui représente l'abstraction de l'interconnexion d'un composant avec le reste du système. Un composant et son enveloppe forment un composant virtuel. Ainsi, les systèmes hétérogènes sont représentés comme un ensemble de composants virtuels interconnectés, dans une architecture virtuelle.

Nous avons illustré l'application de ce modèle de représentation dans un flot de conception des systèmes hétérogènes multiprocesseurs embarqués.

Chapitre 3. Méthodologie pour la validation des systèmes hétérogènes embarqués

Sommaire

3.1. Introduction

Actuellement, l'étape de validation représente un goulot d'étranglement dans le flot de conception des systèmes embarqués. En parallèle, il est observé que plus la validation intervient tôt dans la conception, plus une erreur peut être corrigée rapidement [Cal 95]. Cette étape nécessite la plus grande partie du temps de conception (50%-80%) [Kea 99]. Cela est dû principalement à l'hétérogénéité des systèmes embarqués en termes de niveaux d'abstraction, protocoles de communication ou langages de spécification, ce qui rend de plus en plus difficile l'élaboration des modèles exécutables en vue de la simulation globale des systèmes. Ces modèles sont très sophistiqués : ils engobent l'exécution des différents composants, l'interprétation des interconnexions entre ces composants et aussi l'adaptation entre les différents niveaux d'abstraction,

protocoles de communication ou simulateurs spécifiques aux différents composants du système à valider. L'élaboration de ces modèles est donc un travail qui nécessite beaucoup de temps et qui peut constituer une source d'erreurs dans le flot de conception. Dans ce contexte, la génération automatique des modèles d'exécution en vue de la validation par simulation est devenue indispensable pour la conception des systèmes hétérogènes embarqués.

Plusieurs travaux portent aujourd'hui sur la génération automatique des modèles exécutables pour la validation des systèmes hétérogènes embarqués. Ces solutions présentent un domaine d'application restreint : elles sont bien adaptées pour certaines étapes d'un flot de conception ou elles maîtrisent seulement l'adaptation de certains niveaux d'abstraction et/ou protocoles de communication. Ces solutions ainsi que leurs avantages et inconvénients seront présentées dans la section 3.2.3 de ce mémoire.

L'objectif de ce travail est de définir une méthodologie pour la validation par cosimulation des systèmes hétérogènes embarqués. Cette méthodologie pourra être utilisée tout au long du flot de conception des systèmes hétérogènes embarqués. Elle recevra en entrée la représentation d'un système hétérogène et à l'aide d'une bibliothèque elle générera automatiquement le modèle exécutable correspondant. La clé de voûte dans la définition d'une méthodologie de génération des modèles exécutables est la définition du modèle à générer.

Une première contribution de ce travail consiste en la définition d'une structure interne générique pour les modèles d'exécution des systèmes hétérogènes embarqués. Cette structure permet la génération automatique des modèles d'exécution par assemblage des composants de base existants dans une bibliothèque. L'intégration des nouveaux simulateurs, niveaux d'abstraction ou protocoles de communication revient à enrichir la bibliothèque de génération. Une deuxième contribution de ce travail est la proposition d'une méthodologie pour la génération des modèles exécutables des systèmes hétérogènes embarqués et l'illustration de son application dans le cas du flot de conception du groupe SLS.

Ce chapitre est organisé en six grandes parties. La deuxième partie sera consacrée à la problématique générale de la validation des systèmes hétérogènes par la cosimulation. Dans la troisième partie nous proposerons un modèle de simulation pour la validation des systèmes hétérogènes ; ce modèle a comme principale caractéristique une structure interne générique qui permet sa génération automatique. Dans la quatrième partie de ce chapitre, nous présenterons une méthodologie de génération automatique des modèles de simulation des systèmes hétérogènes. La cinquième partie de ce chapitre illustre l'application de la méthodologie de génération de modèles de simulation dans le cadre du flot de conception du groupe SLS. Enfin, la dernière partie sera consacrée aux conclusions.

3.2. La validation par cosimulation des systèmes hétérogènes embarqués

Dans cette section nous présenterons la problématique de la validation par cosimulation des systèmes hétérogènes embarqués. Dans un première paragraphe nous donnerons la définition de la technique de cosimulation. Ensuite, dans un deuxième paragraphe nous fixerons les qualités requises pour une méthodologie efficace de cosimulation. Enfin, les travaux existants sur la validation par cosimulation des systèmes hétérogènes embarqués seront présentés.

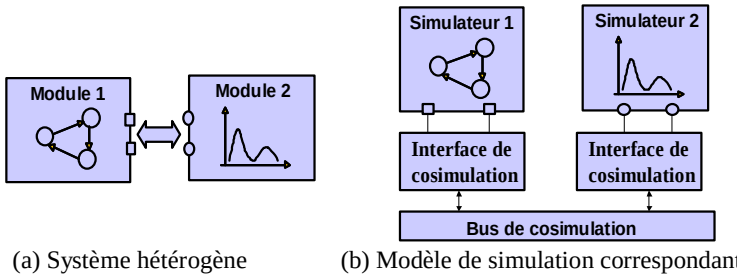
3.2.1. La cosimulation – principe de base

La cosimulation est une technique qui permet l'exécution de systèmes hétérogènes. Généralement, deux approches sont utilisées pour la cosimulation [Cos 99]:

- L'approche compositionelle consiste en la traduction de la spécification multi-langage ou graphique d'un système en un langage unique et en l'exécution de ce modèle résultant [Cal 96], [Pas 99].
- L'approche distribuée ou multi-langage qui consiste en l'exécution conjointe des différents simulateurs. Dans ce cas la spécification des différents composants du système est gardée intacte. Par la suite de ce mémoire, nous sous-entendrons par le terme de cosimulation, la cosimulation distribuée.

Dans le Chapitre 1 de ce mémoire nous avons défini un système hétérogène comme un ensemble de modules hiérarchiques interconnectés. Les différents modules et interconnexions peuvent être caractérisés par différents niveaux d'abstraction, et/ou protocoles de communication et peuvent être spécifiés en utilisant différents langages de spécification. L'exécution d'un tel système implique donc l'exécution parallèle des différents modules ainsi que l'exécution des interconnexions entre les modules.

Par la suite, nous appellerons *instance de cosimulation* l'exécution d'un système hétérogène, en utilisant la technique de cosimulation. Toute instance de cosimulation requiert un *modèle de simulation* du système hétérogène à simuler. Un système hétérogène et son modèle de simulation pour une instance de cosimulation sont illustrés dans la Figure 21.



(a) Système hétérogène (b) Modèle de simulation correspondant

Figure 21. Principe de la cosimulation

Les éléments de base d'un modèle de simulation sont (voir Figure 21.b) :

1. Les différents ***simulateurs*** exécutant les composants du système hétérogène à valider.
2. Le ***bus de cosimulation*** qui est en charge de l'interprétation des interconnexions entre les différents composants d'un système hétérogène. Il est donc responsable du routage des informations entre les différentes composantes. Le bus de cosimulation doit être capable d'interpréter tous les types d'interconnexions d'un système hétérogène (cf. Chapitre 1 de ce mémoire) : des interconnexions de niveau service, des interconnexions de niveau transaction, des interconnexions de niveau message et des interconnexions de niveau RTL.
3. Les ***interfaces de cosimulation*** qui facilitent la communication des différents composants du système via le bus de cosimulation. Elles sont donc en charge d'adapter chaque composant au bus de cosimulation. Le rôle des interfaces de cosimulation revient donc à :
   - L'adaptation des différents simulateurs au bus de cosimulation – pour garantir la transmission des informations entre les simulateurs sur lesquels les composants s'exécutent et le bus de cosimulation,
   - L'adaptation des différents protocoles de communication – pour permettre l'exécution des systèmes englobant des modules caractérisés par des protocoles de communication différents,
   - L'adaptation des niveaux d'abstraction – pour permettre l'exécution des systèmes englobant des modules situés à des niveaux d'abstraction différents.

L'ensemble composé du bus de cosimulation et des interfaces de cosimulation représente l'épine dorsale du modèle de simulation ; il garanti la communication et la synchronisation des composants du système.

Un ***environnement de cosimulation*** est un instrument pour la création des instances de cosimulation. Il reçoit en entrée une spécification hétérogène et génère automatiquement les interfaces de cosimulation et le bus de cosimulation nécessaires pour l'exécution de la spécification d'entrée. Un tel environnement repose sur un ***cadre conceptuel*** définissant les éléments de base et l'algorithmique mise en œuvre. Deux étapes importantes dans la définition du cadre conceptuel d'un outil de cosimulation sont :

- La définition du bus de cosimulation et des interfaces de cosimulation
- La définition de la méthodologie de génération automatique des modèles de simulation ou des instances de cosimulation.

Ces deux étapes sont liées : la définition du bus de cosimulation et des interfaces de cosimulation peut faciliter la génération automatique des modèles d'exécution. Cette définition peut aussi influencer directement l'efficacité de l'outil de cosimulation.

Dans le paragraphe suivant nous présenterons les qualités requises pour un modèle de simulation des systèmes hétérogènes afin d'obtenir un environnement de cosimulation efficace.

3.2.2. Qualités requises pour le modèle de simulation des systèmes hétérogènes embarqués

Comme nous l'avons expliqué dans la section précédente, pour la cosimulation des systèmes hétérogènes embarqués un modèle de simulation est nécessaire. Les principales caractéristiques requises pour la cosimulation des systèmes hétérogènes embarqués sont la flexibilité, la modularité et la précision.

La flexibilité

Le modèle de simulation doit être flexible. Il doit donc pouvoir s'adapter aux changements qui interviennent pendant le flot de conception :

- Le changement de l'environnement (par exemple le changement de contraintes de fonctionnement),
- Le changement de l'utilisation (par exemple le changement des protocoles de communication),
- Le changement de la technologie (par exemple le changement de simulateur).

La modularité et l'extensibilité

Un modèle de simulation doit être aussi modulaire et extensible. Ainsi il doit permettre le traitement indépendant des différents composants du système et maîtriser l'évolution de la complexité du système à simuler.

Les principaux avantages d'utilisation d'un tel système dans un flot de conception sont :

- La possibilité de la validation des différentes solutions de réalisation d'un même système,
- La possibilité de la validation du système dans le cas d'intégration de nouvelles fonctionnalités ou de nouveaux composants,
- La validation des systèmes avec leur environnement.

La précision

Deux niveaux de précision peuvent être distingués pour la validation par simulation :

- La validation temporelle qui prend en compte la notion de temps et qui permet d'évaluer les performances d'un système et de vérifier les contraintes temporelles du système,
- La validation fonctionnelle qui a pour but de vérifier la fonctionnalité d'un système. Pour cela seules les données échangées entre les simulateurs sont considérées, sans prise en compte du temps de propagation.

Un modèle de simulation efficace doit permettre au concepteur le choix du niveau de précision de la simulation.

3.2.3. Etat de l'art sur la cosimulation

Plusieurs méthodologies et environnements de cosimulation ont été proposés aussi bien sur le marché que dans le monde universitaire. La plupart de ces méthodologies permettent la cosimulation pour la validation des systèmes au niveau RTL. Plusieurs travaux proposent des méthodologies pour l'intégration des différents langages et simulateurs pour la cosimulation ainsi que l'intégration des différents modèles de calcul afin d'étendre le domaine d'application de cette technique pour des applications multi-domaines. Récemment, des nouvelles méthodologies pour la cosimulation multi-niveau sont apparues.

Les principaux travaux existants dans chacune de ces directions seront présentés à la suite.

Cosimulation logicielle-matérielle au niveau RTL

La cosimulation est utilisée couramment dans l'industrie pour la validation des systèmes au niveau RTL. L'objectif principal de ces travaux est de permettre la validation de l'intégration du logiciel et du matériel dans un cadre virtuel, avant le prototypage.

La cosimulation logicielle-matérielle au niveau RTL permet l'exécution parallèle du logiciel et du matériel. La partie logicielle d'un système est simulée à l'aide d'un ou plusieurs simulateurs de processeur (ISS[7]) et la partie matérielle est simulée sur un simulateur de modèles matériels (par exemple un simulateur de VHDL, Verilog, plus récemment SystemC etc.).

Comme environnements commerciaux, Mentor Graphics Seamless CVE [Men 02], Synopsys Eagle*i* [Syn 02] et CoWareN2C [Cow 02] permettent la simulation parallèle de plusieurs ISS et d'un simulateur matériel.

Albreght *et al.* [Alb 93] présente une méthodologie d'estimation des performances de la cosimulation logicielle-matérielle dans l'hypothèse où plusieurs ISS et un simulateur du matériel sont utilisés. L'outil de cosimulation utilisé est Synopsys Eagle*i* [Syn 02].

Dans le même cadre de la cosimulation logicielle-matérielle de niveau RTL, A. Ghosh *et al.* [Gho 95] propose un environnement de cosimulation où plusieurs simulateurs peuvent s'exécuter de manière concurrentielle. Leur travail est focalisé sur la synchronisation entre les différents simulateurs logiciels/matériels.

Ce type de cosimulation logicielle-matérielle s'est imposé par sa précision, mais malheureusement, avec la croissance de la complexité des systèmes, il est devenu trop lent pour une exploration rapide de l'espace de solutions.

[7] de l'anglais Instruction Set Simulator

Cosimulation multi-modèles de calcul

La diversité des modèles de calcul des différents sous-systèmes d'un système hétérogène constitue le centre d'intérêt de plusieurs travaux.

Un travail représentatif de cette direction est le travail effectué dans le cadre du projet de recherche Ptolemy [Pto 02] par l'Université Berkeley. Ce projet porte sur la modélisation et la simulation des systèmes hétérogènes en termes de modèles de calcul. Le principal travail de fond de ce projet est la définition d'un cadre conceptuel pour la comparaison des différents modèles de calcul [Lee 97]. L'issue de cette étude est la définition d'un environnement construit en Java, appelé PtolemyII [Eke 01]. Cet environnement permet la cosimulation des systèmes hétérogènes intégrant plusieurs modèles de calcul dont les plus importants sont : le modèle à événements discrets (DE), le modèle à base de machines d'états finis (FSM), le modèle à base de processus communicants (CSP), etc. La réalisation d'un modèle de calcul est appelée domaine.

Plusieurs projets ont repris les travaux de Ptolemy pour construire des outils de cosimulation. Ainsi l'environnement de conception logicielle-matérielle Polis [Edw 97] utilise le domaine DE de Ptolemy comme support de réalisation des instances de cosimulation pour la validation au niveau RTL des systèmes logiciels-matériels. SPW [Spw 02] de Cadence utilise les concepts de Ptolemy pour la validation par cosimulation des systèmes orientés traitement de signal.

Cosimulation multi-langage

Plusieurs travaux de recherche orientés vers la cosimulation ont été focalisés sur l'aspect de la génération automatique des modèles exécutables pour la cosimulation multi-langage. Ces travaux exploitent les bibliothèques fournies par les différents simulateurs de langages en vue de la communication avec l'extérieur.

Valderrama *et al***[Val** 94] [Val 94] présente un environnement de cosimulation appelé VCI et qui permet la génération automatique des modèles pour la cosimulation C-VHDL. La communication et la synchronisation entre les différents simulateurs participants dans une instance de cosimulation sont réalisées en utilisant les IPCs (Inter Process Communication) [And 91].

[Lem 00] et [Cos 99] exploitent les travaux présentés antérieurement. La contribution apportée consiste en la génération automatique des interfaces de cosimulation qui permettent l'intégration de Matlab/Simulink [Mat 00], et SDL/ObjectGeode [Sdl 87]. Une autre contribution est la proposition d'un environnement qui permet la cosimulation multi-langage géographiquement distribuée. Dans ce cas, la communication et la synchronisation entre les différents simulateurs participant dans une instance de cosimulation sont réalisées en utilisant les sockets [And 91].

Les travaux présentés dans [Hen 01] proposent une approche pour la validation rapide des systèmes matériels, tout en permettant leur spécification multi-langage. Les différents modules du

système peuvent être décrits en VHDL et Verilog, ainsi que en des langages de spécification basés sur C/C++ (par exemple Cynlib [For 01]). Cette solution utilise l'interface FLI[8] du simulateur Modelsim [Mod 01]. Ainsi, les modules spécifiés dans des langages de spécification différents sont exécutés sur un seul simulateur (Modelsim), ce qui permet de résoudre le problème des approches classiques de la cosimulation multi-langage : le sur-coût de synchronisation entre les différents simulateurs. L'approche a été intégrée dans l'environnement Picasso pour la conception des systèmes logiciels-matériels [Hen 99], [Fil 02b].

Bien que très efficaces pour l'intégration des différents simulateurs dans une même instance de cosimulation, ces travaux ne traitent pas de manière systématique l'aspect multi-niveau dans la cosimulation. Ainsi, dans le cas où une adaptation entre différents niveaux d'abstraction est nécessaire cette adaptation doit être réalisée explicitement par l'utilisateur.

Cosimulation multi-niveau

Très récemment des travaux portant sur la cosimulation ont abordé l'approche de la cosimulation multi-niveau. Ce type de cosimulation implique l'exécution conjointe des composants situés à différents niveaux d'abstraction.

La majorité de ces travaux ont proposé des solutions pour l'exécution conjointe des modèles de simulation fonctionnelle et des modèles de simulation précise au niveau cycle.

Parmi ces solutions, le modèle fonctionnel de bus BFM[9] [Sem 00] constitue aujourd'hui la méthodologie conventionnelle pour l'interconnexion des modèles de simulation fonctionnelle et des modèles de simulation cycle-près, surtout pour la validation de l'interfaçage logiciel-matériel. Malheureusement, cette méthodologie est limitée seulement à la transformation des accès mémoire fonctionnels en des accès mémoire de cycle-près ; les accès à des canaux de communication de haut niveau (par exemple FIFO) ne sont pas pris en compte.

CoWareN2C (Synopsys) [Cow 02] est un environnement qui offre une solution de cosimulation multi-niveaux. Les niveaux d'abstraction proposés dans cet environnement sont : le niveau BCA (venant de l'anglais Bus Cycle Accurate) qui correspond au niveau RTL classique et le niveau UT (venant de l'anglais UnTimed) ou seulement la fonctionnalité d'un système est prise en compte, sans les aspects concernant la notion de temps. Pour la simulation conjointe des sous-systèmes décrits au niveau BCA et des sous-systèmes décrits au niveau UT, CoWareN2C présente le concept BCASH (Bus-Cycle-Accurate-Shell). Il s'agit en fait d'une enveloppe des sous-systèmes de niveau UT qui permet d'estimer le temps nécessaire pour l'exécution d'une procédure dans un

[8] venant de l'anglais Foreign Language Interface
[9] de l'anglais Bus Functional Model

tel modèle. Une telle enveloppe est générée automatiquement seulement dans le cas ou le sous-système UT est ciblé en logiciel et il s'exécutera en fait sur un simulateur de processeur (ISS).

SystemC [Sys 00] propose le concept d'interface de conversion entre différents niveaux d'abstraction. Une telle interface facilite l'interconnexion des modules communicant par des RPC (Remote Procedural Call) et des modules de niveau RTL. Ce concept est mis à la disposition des concepteurs qui doivent construire manuellement les interfaces, ce que implique généralement un travail fastidieux qui peut induire des erreurs et une perte de temps de conception.

Le consortium VSIA [Lem 00] travaille sur l'assemblage de composants hétérogènes. Il défini un standard de spécification et documentation (SLI[10]) ainsi qu'un standard des interfaces des bus (VCI[11]). Actuellement VSIA représente un guide très efficace pour la conception des systèmes hétérogènes, mais il ne se concentre pas encore sur la génération automatique des interfaces de simulation des composants hétérogènes. La plupart de travaux courants utilisent le standard VSIA en vue de la conception des interfaces logicielles/matérielles. Ainsi, les travaux présentés dans [Cyr 01] proposent la génération automatique de l'interface pour le processeur ARM, en utilisant les recommandations VSIA ; ce travail permet aussi une évaluation du standard VSIA. COSY [Bru 00] propose un modèle générique pour l'implémentation et un modèle de performances pour les communication de niveau système. Cela pour permettre l'interconnexion des différents IP via des interfaces des bus (VCI) proposées par VSIA.

Finalement, afin d'obtenir un bon compromis entre la précision et la vitesse de simulation d'un système complet, Chinook [Cho 95] propose une méthodologie qui permet de changer dynamiquement pendant la simulation les niveaux d'abstraction des différents modèles de simulation.

3.3. Modèle de simulation pour la validation des systèmes hétérogènes multi-langage, multi-niveau

Dans cette section nous présentons un modèle de simulation pour la validation des systèmes embarqués hétérogènes. Notre contribution consiste en la définition de la structure des interfaces de cosimulation, en vue de la génération automatique des modèles de simulation pour les systèmes hétérogènes représentés à l'aide du modèle de représentation présenté dans le Chapitre 2 de ce mémoire (voir Figure 22). Les interfaces de cosimulation constituent en fait le modèle d'exécution des enveloppes de différents composants.

[10] de l'anglais System Level Interface
[11] de l'anglais Virtual Component Interface

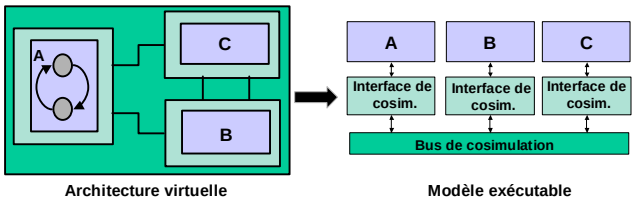


Figure 22. Spécification hétérogène et modèle exécutable correspondant

Le point de départ dans la définition de ce modèle est le modèle de simulation utilisé généralement pour la cosimulation : les différentes composants d'un système communiquent par un bus de cosimulation via des interfaces de cosimulation. Ce modèle a été présenté dans la section 3.2.1 de ce mémoire et il a été repris pour la clarté de l'exposé dans la Figure 23.a.

Comme nous l'avons expliqué dans la section 3.2.1, l'interface de cosimulation a une fonctionnalité complexe qui doit assurer l'adaptation d'un composant au bus de cosimulation. Il s'agit principalement de l'adaptation des simulateurs et de l'adaptation de la communication, plus exactement de l'adaptation des niveaux d'abstraction ou des protocoles de communication.

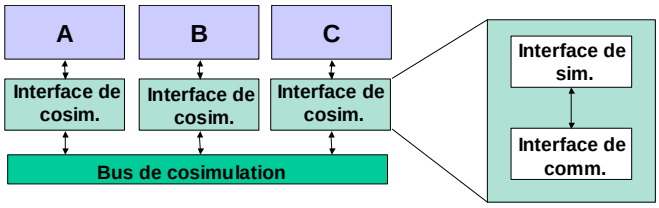


Figure 23. Séparer le problème d'adaptation des simulateurs et le problème de l'adaptation de la communication

Pour séparer le problème de l'adaptation des simulateurs et le problème de l'adaptation de la communication nous proposons une première structuration de l'interface de cosimulation en deux interfaces sous-jacentes :

- Une interface qui adapte le simulateur au bus de cosimulation du modèle d'exécution, interface que nous appellerons **interface du simulateur**,
- Une interface qui adapte les différents protocoles de communication ou niveaux d'abstraction, interface que nous appellerons **interface de communication**.

Le séparation du problème de l'adaptation du simulateur et de la communication rend plus extensible le modèle de simulation : ainsi, l'ajout d'un nouveau simulateur dans un environnement de simulation nécessitera seulement une nouvelle interface du simulateur. De même, l'ajout des nouveaux protocoles de communication ou des nouveaux niveaux d'abstraction impliquera seulement la construction des nouvelles interfaces de communication. Cette séparation rend aussi plus flexible le modèle de simulation : le remplacement d'un simulateur du même modèle (par

exemple le remplacement du simulateur de VHDL ModelSim par le simulateur VSS) ne demandera pas le changement de l'interface de cosimulation entière; seulement l'interface du simulateur sera adaptée.

3.3.1. Interface du simulateur

L'interface du simulateur est en charge d'adapter un environnement de simulation dans le modèle de simulation globale. Elle rend en fait transparente l'existence d'un simulateur étranger dans l'instance de cosimulation. La fonctionnalité d'une telle interface doit assurer la synchronisation et l'échange correct des données avec l'extérieur du simulateur. Dans notre travail, pour la définition de l'interface du simulateur nous utilisons les études déjà effectuées dans notre groupe dans [Lem 00] et [Cos 99].

Cette interface résout seulement en termes de langages de spécification le problème d'hétérogénéité des ports internes et externes de l'enveloppe d'un module. La Figure 24 illustre un module et son interface du simulateur. Ces deux derniers communiquent par des interconnexions point-à-point. Du côté communication avec le module, l'interface de simulation présente donc des ports duals aux ports du module. Pour la communication avec le reste du modèle de simulation l'interface du simulateur présente les même types de ports que le module qu'elle adapte. Les entrées/sorties de l'interface du simulateur sont donc données par les ports internes de l'enveloppe du module à adapter

L'adaptation de la communication entre les ports internes et externes d'une enveloppe est effectuée par l'interface de communication qui sera présentée par la suite.

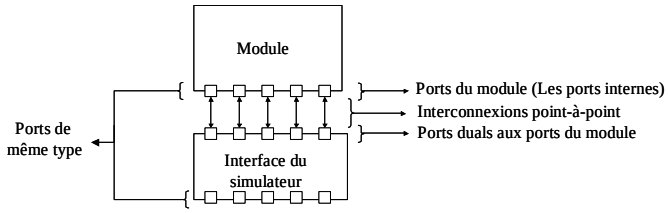


Figure 24. Module et son interface du simulateur

3.3.2. Interface de communication

L'interface de communication est nécessaire chaque fois qu'une adaptation en terme de niveaux d'abstraction ou protocoles de communication doit être effectuée pour la réalisation d'une instance de cosimulation.

La Figure 25 illustre un exemple de module qui nécessite une interface de communication. Ce module est situé au niveau d'abstraction N (par exemple RTL) et il est interconnecté avec des canaux de communication de niveau M ≠ N (par exemple le niveau macro–architecture présenté dans le Chapitre 1). Dans la figure, nous avons utilisé le modèle de représentation des systèmes

hétérogènes présenté dans le Chapitre 2. Plusieurs adaptations peuvent être nécessaires entre les ports internes et les portes externes de l'enveloppe :

1. Les ports internes et les ports externes de l'enveloppe peuvent être caractérisés par des primitives de communication différentes ; ainsi le comportement du module peut appeler via les ports internes d'autres primitives de communication que les primitives offertes par les canaux de communication
2. Les ports internes et les ports externes de l'enveloppe peuvent manipuler des types de données différents
3. Un arbitrage de la communication entre les ports internes et les ports externes de l'enveloppe peut aussi être nécessaire. Ainsi plusieurs informations peuvent être envoyées simultanément via des ports internes vers un même port externe. De même, plusieurs informations peuvent être envoyées simultanément via des ports externes vers un même port interne.

L'interface de communication est en charge de réaliser toutes ces adaptations. Afin de maîtriser la complexité d'une telle interface, nous proposons sa structuration en différents éléments de base ayant une fonctionnalité bien définie.

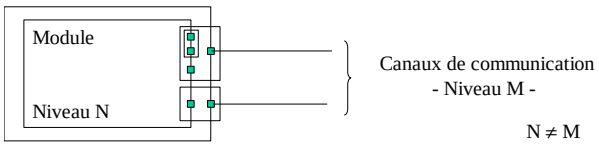


Figure 25. Exemple de module nécessitant une interface de communication pour la simulation

La structure de l'interface de communication est illustrée dans la Figure 26. Dans cette figure, pour la clarté de l'explication, l'interface de simulation n'est pas représentée. L'interface de communication a une structure interne modulaire composée de trois types d'éléments de base :

- L'adaptateur de module (MA) qui est spécifique au module qui doit être interconnecté,
- L'adaptateur de canal (CA) qui est nécessaire pour chaque canal de communication avec lequel le module doit être interconnecté,
- Le média de communication (ICM) qui assure la communication entre l'adaptateur de module et les adaptateurs de canal.

En terme d'entrées/sorties, l'interface de communication présente les ports internes et les ports externes constituant l'enveloppe du module à intégrer dans une instance de cosimulation. Ces différents ports peuvent être spécifiques aux différents niveaux d'abstraction ou protocoles de communication. Dans la Figure 26, les protocoles de communication (P) et les niveaux d'abstraction (L) sont illustrés par Pk/Li, Pi/Li, Pj/Lj, Pk/Lk et Pl/Ll.

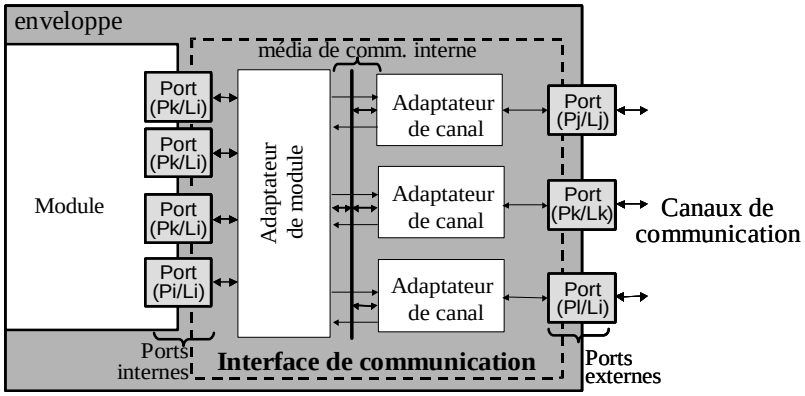


Figure 26. Structure générique de l'interface de communication

L'avantage de cette structure interne de l'interface est de permettre l'indépendance entre le traitement du comportement et celui de la communication. Ainsi, le raffinement (ou le remplacement) du module implique seulement le choix d'autres adaptateurs de modules. L'adaptateur de canal lui reste inchangé. Réciproquement, le raffinement de la communication au niveau du canal touche uniquement l'adaptateur de canal en question.

Cette structure permet non seulement de maîtriser la complexité de l'opération d'adaptation, mais aussi de réduire considérablement la dimension de l'espace des choix possibles des adaptateurs.

En effet, plaçons-nous dans le cas où un module pourrait être choisi parmi m types possibles (avec considération des différents niveaux d'abstraction) et où un canal peut être choisi parmi n possibles. Dans ce cas le nombre d'adaptateurs nécessaire est $m*n$ si on considère que l'adaptateur est constitué d'un seul bloc, alors qu'il n'est que de $m+n$ si on adopte le principe de la décomposition de l'interface.

Par la suite, nous allons présenter les différentes composants de l'interface de communication.

L'adaptateur de module (MA[12])

L'adaptateur de module est en charge de l'intégration d'un module dans une instance de cosimulation. Ces principales fonctions sont de fournir les primitives de communication appelées dans le comportement du module via les ports internes et d'assurer l'échange d'information « brute » dépourvue de tout protocole de ou vers le média de communication. Il est aussi en charge d'une éventuelle conversion de données. Par exemple, la conversion d'un type fixe de donnée (entier) en représentation fixe de données en binaire. La conversion de données peut représenter

[12] venant de l'anglais Module Adapter

aussi l'unification des différentes données de type fixe dans une structure de données ou bien le fractionnement d'une structure de données en plusieurs données séparées.

Les types de modules dépendent fortement de l'application. Ceci est d'autant plus vrai qu'au cours du processus de raffinement, le nombre et le type de modules utilisés peuvent varier considérablement. Cette variété et cette dépendance par rapport à l'application et la position dans le flot de conception entraîne systématiquement un manque de flexibilité au niveau de la génération des adaptateurs de module, manque qui vient s'ajouter à la taille toujours importante que risque d'avoir la bibliothèque de génération de ces adaptateurs.

Pour remédier à cet inconvénient nous proposons le découpage de la structure de l'adaptateur de module de celle de l'application ou du système. Nous proposons de structurer l'adaptateur de module en terme de groupement logique de ses ports au lieu de considérer le module comme un tout entier (par exemple dans le cas d'un processeur tous ces ports sont groupés logiquement, contrairement au cas d'une mémoire multi-ports où ses ports sont logiquement indépendants). En effet en procédant ainsi, la diversité des types de modules se trouve maîtrisée et la taille de la bibliothèque pour la génération des adaptateurs est considérablement réduite.

En terme d'entrées/sorties, les ports de l'adaptateur de module peuvent être groupés en deux principales catégories (voir la Figure 26) :

- Les ports internes de l'enveloppe du module à adapter
- Les ports spécifiques au média de communication interne

Comme nous l'avons expliqué dans le Chapitre 2 (section 2.3.2) de ce mémoire, les ports peuvent être groupés logiquement par un port hiérarchique. La structure de l'adaptateur de module sera donc en rapport direct avec le groupement des ports internes de l'enveloppe du module en ports hiérarchiques. Ainsi, l'adaptateur de module sera décomposé en blocs élémentaires, où chaque bloc correspondra à un port interne hiérarchique.

Nous appellerons les blocs élémentaires composant un adaptateur de module, ***adaptateurs de port (PA[13])***. L'adaptateur de port réalisera une partie de la fonctionnalité de l'adaptateur de module : la partie concernant le port qu'il adapte. Ainsi, il fournira les primitives de communication demandées par le comportement de module via le groupement logique de ports internes et assurera le transfert de données de et vers le média de communication interne. Il est aussi en charge d'une éventuelle conversion de données. Le principe de la décomposition de l'adaptateur de module en adaptateurs de ports est illustré dans la Figure 27. Les adaptateurs de canal ne sont pas représentés dans cette figure pour des raisons de clarté. A gauche nous avons représenté un module ayant plusieurs ports. Remarquons qu'une partie de ses ports sont groupés

[13] venant de l'anglais Port Adapter

pour former un ensemble logique ou encore un port hiérarchique. Un de ses ports est traité individuellement.

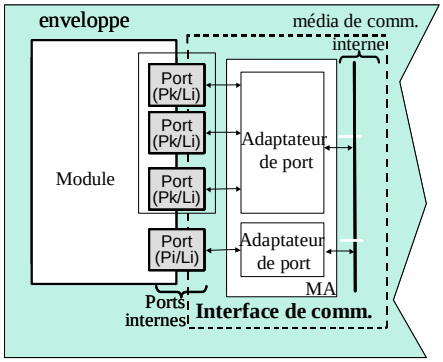


Figure 27. Décomposition de l'adaptateur de module en adaptateurs de ports

L'adaptateur du canal (CA[14])

Ce composant de l'interface de communication assure l'adaptation des interconnexions dans une instance de cosimulation d'une architecture virtuelle. Comme nous l'avons présenté dans le Chapitre 2 c de ce mémoire, dans une architecture virtuelle les interconnexions sont représentées par des canaux de communication virtuels. Un canal de communication virtuel peut encapsuler un canal de communication ou plusieurs canaux de communication groupés logiquement.

Dans le cas d'un canal virtuel encapsulant un seul canal de communication, le comportement de l'adaptateur de canal se réduit au transfert de l'information de ou vers le média de communication interne en effectuant des appels de primitives de communication fournies par le canal de communication à adapter et en utilisant les primitives de communication spécifiques au média de communication interne. Si nous considérons les niveaux d'abstraction pour la communication définis dans le Chapitre 1 de ce mémoire, l'adaptateur d'un canal de niveau service appelle un service, l'adaptateur d'un canal de niveau message appelle des primitives *send/receive()*, l'adaptateur d'un canal de niveau macro-architecture appelle des primitives spécifiques à un protocole de communication (ex. *fifo_write()*), l'adaptateur d'un canal de niveau RTL effectue des set/reset sur des signaux physiques. Les différents appels de primitives de communication selon le canal à adapter sont résumés dans le Tableau 5.

Dans l'hypothèse où le canal à adapter est un canal virtuel groupant plusieurs canaux de communication (par exemple un bus), la fonctionnalité d'un adaptateur de canal ne se restreint pas aux appels de primitives de communication offertes par le canal ; il a un comportement plus complexe qui respecte la sémantique du protocole groupant les canaux de communication dans un

[14] venant de l'anglais Chanel Adapter

canal virtuel. Des exemples de tels adaptateurs de canal seront donnés dans la section 3.5.1 de ce mémoire.

Type de canal de communication à accéder	Exemples de primitives de communication appelés par l'adaptateur de canal
Réseau abstrait de niveau service	Demande de service
Canal actif de niveau message	Send/Receive (fichier, disque)
Canal abstrait de niveau macro-architecture	Write/Read (donnée, port)
Fil physique de niveau RTL	Set (valeur, port)

Tableau 5. Particularisation des appels de primitives de communication dans un CA selon le niveau d'abstraction du canal de communication à accéder

Le média de communication interne (ICM[15])

Le média de communication interne permet l'échange de données entre les deux types d'adaptateurs : l'adaptateur de module composé éventuellement de plusieurs adaptateurs de port et l'adaptateur de canal. Le média de communication garantit en fait une interconnexion correcte entre les composants de base de l'interface de communication.

Devant assurer une communication multi-point, le média de communication interne est aussi en charge d'un éventuel arbitrage de la communication. Dans le Chapitre 2 (section 2.3.2) nous avons présenté les ports virtuels comme les ports groupant des ports internes et des ports externes qui transfèrent une même information. Le média de communication doit donc respecter cela et assurer, via les PA et les CA, le routage de l'information entre les ports internes et les ports externes groupés dans un port virtuel.

3.3.3. Le bus de cosimulation

Dans le cas de notre modèle de simulation d'une architecture virtuelle, le bus de cosimulation peut avoir une des deux fonctionnalités :

1. Il peut englober le modèle d'exécution des canaux de communication de l'architecture virtuelle. Ceci correspond au cas où le concepteur de l'architecture virtuelle ne fournit pas une implémentation de la communication.
2. Dans le cas où le comportement des canaux de communication est fourni par le concepteur dans l'architecture virtuelle, les canaux de communication sont traités comme des modules dans l'instance de cosimulation de l'architecture virtuelle. Le bus de cosimulation est donc un ensemble de signaux assurant un simple transfert des données ou des appels de primitives de communications entre les différents éléments de l'instance de cosimulation.

[15] venant de l'anglais Internal Communication Media

3.3.4. Architecture virtuelle versus modèle de simulation

Pour la clarté de l'explication, dans ce paragraphe, à l'aide des définitions de l'architecture virtuelle et du modèle de simulation de l'architecture virtuelle nous réalisons une mise en correspondance entre ces deux concepts.

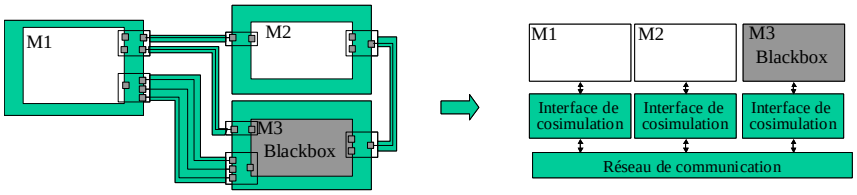
Une architecture virtuelle a été présentée comme un ensemble de modules virtuels interconnectés par des canaux de communication virtuels. Un module virtuel est constitué en fait d'un module composant le système hétérogène modélisé par l'architecture virtuelle et une enveloppe. L'enveloppe est constituée à son tour des ports internes spécifiques au module enveloppé et des ports externes spécifiques aux canaux de communication interconnectés à ce module. Les différents ports internes et externes participant à la transition d'une même information sont groupés en ports virtuels.

Le modèle de simulation d'une architecture virtuelle est un ensemble de modules s'exécutant sur des simulateurs différents interconnectés par un bus de cosimulation via des interfaces de cosimulation. Chaque interface de cosimulation peut être composée d'une interface du simulateur et d'une interface de communication. Les entrées/sorties de l'interface du simulateur sont données par les ports internes de l'enveloppe du module. L'interface de communication est composée d'un adaptateur de module et de plusieurs adaptateurs de canal. L'adaptateur de module est composé de plusieurs adaptateurs de port. La décomposition d'un adaptateur de module en adaptateurs de port est donnée par le groupement des ports internes en ports hiérarchiques. A chaque port externe, ou groupement de ports externes en ports hiérarchiques, correspond un adaptateur de canal. Les adaptateurs de canal et l'adaptateur de module sont interconnectés par le média de communication interne. La structure de média de communication est donnée par les ports virtuels de l'enveloppe d'un module. Quant au bus de cosimulation, il peut englober le modèle d'exécution des différents canaux de communication de l'architecture virtuelle, ou dans le cas où le comportement de ces canaux est déjà fourni par le concepteur dans l'architecture virtuelle, il a une fonctionnalité simple de transfert de données ou des appels de primitives de communication.

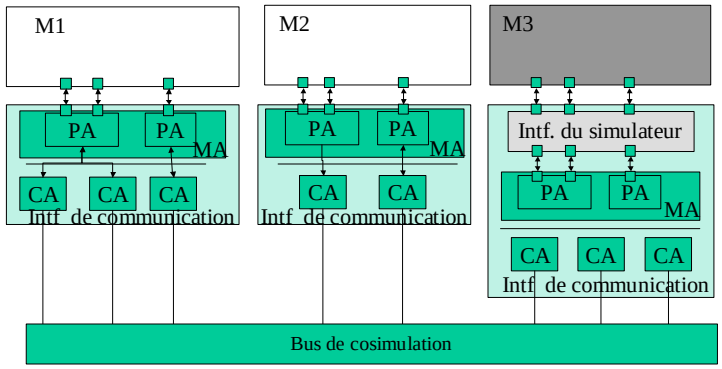
La correspondance entre une architecture virtuelle et la structure de son modèle de simulation est résumée dans le Tableau 6.

<table>
<tr><th colspan="2">Eléments de base de l'architecture virtuelle</th><th>Eléments de base du modèle de simulation correspondant</th></tr>
<tr><td colspan="2">Module</td><td>Simulateur exécutant le comportement du module</td></tr>
<tr><td colspan="2">Enveloppe</td><td>Interface de cosimulation</td></tr>
<tr><td rowspan="3">Port virtuel</td><td rowspan="2">Ports internes groupés éventuellement dans des ports hiérarchiques</td><td>Interface du simulateur</td></tr>
<tr><td>Adaptateurs de port composant l'adaptateur de module</td></tr>
<tr><td>Ports externes groupés éventuellement dans des ports hiérarchiques</td><td>Adaptateurs de canal</td></tr>
<tr><td colspan="2">Canaux de communication virtuels</td><td>Bus de cosimulation ou simulateur exécutant son comportement</td></tr>
</table>

Tableau 6. Concepts de base de l'architecture virtuelle versus concepts de base du modèle de simulation correspondant



(a) Architecture virtuelle **(b) Modèle de simulation**



(c) Modèle de simulation détaillé

Figure 28. Architecture virtuelle et modèle de simulation correspondant

Pour une meilleure illustration la Figure 28 montre un exemple d'architecture virtuelle et le modèle de simulation correspondant.

La Figure 28.a illustre l'architecture virtuelle d'un système composé de trois modules qui sont situés aux niveaux d'abstraction différents du niveau d'abstraction du réseau de communication. Les ports internes et les ports externes des enveloppes des composants de ce système sont donc des ports situés à des niveaux d'abstraction différents. Supposons que le module M3 soit décrit dans un langage de spécification différent du langage utilisé pour la spécification du reste du système. Dans ce cas, l'enveloppe du module M3 est composée des ports internes et externes spécifiques aux langages de spécification différents et situés aux niveaux d'abstraction différents.

La Figure 28.b montre le modèle de simulation correspondant à cette architecture virtuelle. Chaque composant est connecté au réseau de communication par une interface de cosimulation. L'interface de cosimulation de chaque composant correspond en fait au modèle de simulation de l'enveloppe du composant.

La Figure 28.c montre les détails des interfaces de cosimulation, nécessaires pour l'exécution de l'architecture virtuelle. Pour chaque module enveloppé, à chaque port interne hiérarchique de l'enveloppe correspond un adaptateur de port et à chaque canal virtuel correspond un adaptateur de canal. Comme M3 est décrit dans un autre langage de spécification que le reste du système, une

interface du simulateur est aussi nécessaire pour l'exécution. Le bus de cosimulation fournit le comportement des canaux de communication. Nous avons donc considéré que le comportement des canaux de communication n'a pas été fourni dans la description initiale du système. Dans le cas de cet exemple, nous avons fait l'hypothèse que les modules M1 et M2 sont décrits dans le même langage que le langage utilisé pour la réalisation des interfaces de simulation et du bus de cosimulation dans le modèle de simulation du système entier

3.4. Génération automatique des modèles de simulation des systèmes hétérogènes

Dans cette quatrième partie, nous présentons une méthodologie pour la génération automatique des modèles de simulation pour les architectures virtuelles des systèmes hétérogènes. Cette méthodologie est basée sur la génération des interfaces de cosimulation par assemblage d'éléments situés dans une bibliothèque. Ce paragraphe présentera la vue d'ensemble de cette méthodologie et la section 3.5.2 présentera quelques détails dans le cas de l'utilisation de cette méthodologie pour le flot de conception du groupe SLS.

Cette méthodologie reçoit en entrée la description de l'architecture virtuelle d'un système hétérogène et utilise une bibliothèque de cosimulation pour la construction des instances de cosimulation. La génération automatique des instances de cosimulation consiste en fait dans la génération des interfaces de cosimulation et dans leur assemblage avec les composants et le bus de cosimulation dans une instance de cosimulation.

La stratégie que nous utilisons pour la génération automatique des interfaces de cosimulation est basée sur le principe de configuration et d'assemblage d'éléments existants dans la bibliothèque de cosimulation. La bibliothèque que nous utilisons pour la génération automatique des interfaces de cosimulation est constituée de quatre types d'éléments :

- Des éléments pour la génération des interfaces de simulateurs,
- Des modèles pour les adaptateurs de canal,
- Des modèles pour les adaptateurs de port,
- Des modèles pour le bus de cosimulation.

La vue d'ensemble du flot de génération automatique des modèles de simulation est illustrée dans la Figure 29.

L'analyse de l'architecture virtuelle est la première étape du flot. Son but est de fournir les informations requises pour les autres étapes sur un format directement utilisable. L'entrée concernée par cette analyse est évidemment l'architecture virtuelle ; la bibliothèque de cosimulation n'est pas traitée pendant cette étape. A la fin de cette étape, toutes les informations sur la topologie de la spécification (ex. quels composants sont interconnectés avec quels canaux de

communication, etc.) et toutes les adaptations nécessaires pour la simulation de la spécification d'entrée sont déterminées : les adaptations des niveaux d'abstraction, des protocoles de communication et des environnements de simulation. Cette étape est la seule qui soit en contact direct avec la spécification d'entrée, elle serait donc la seule à changer s'il y avait un changement dans la spécification.

La deuxième étape du flot est la sélection des éléments de la bibliothèque de cosimulation et la construction des interfaces de cosimulation. En utilisant les résultats de l'analyse de la spécification, les éléments appropriés de la bibliothèque sont extraits et configurés en concordance avec les besoins de la spécification à simuler. Une fois que les éléments de la bibliothèque de cosimulation ont été extraits et configurés, ils sont assemblés pour construire les interfaces de cosimulation. Cette étape fournit également les informations requises pour l'étape suivante, informations sur la topologie des interfaces de cosimulation qui aideront à l'interconnexion des interfaces de cosimulation avec les composants qu'elles adaptent.

Une dernière étape du flot de génération assemble les composants du système à simuler (les interfaces de cosimulation et le bus de cosimulation) dans une instance de cosimulation. En effet, pendant cette étape les interfaces de cosimulation et le bus de cosimulation sont insérés dans la structure initiale du système. Pour ce faire, les informations sur la topologie de l'architecture virtuelle extraites pendant l'étape d'analyse du système et les résultats de l'étape de construction des interfaces de communication sont utilisés comme une entrée de cette étape.

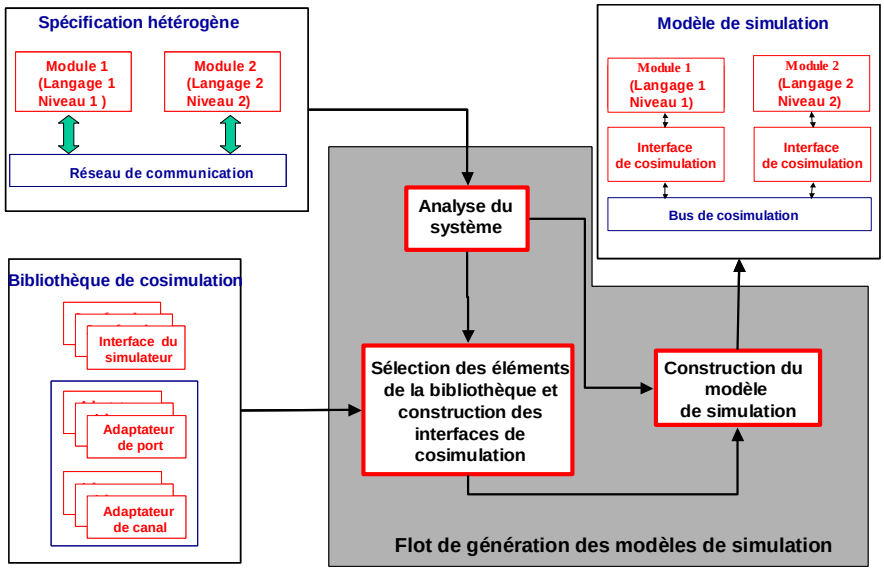


Figure 29. Vue générale de la génération automatique des modèles de simulation

3.5. Application du modèle de simulation et de la méthodologie de génération automatique des modèles de simulation dans le cadre du flot de conception du groupe SLS

Cette section présente l'application du modèle de simulation et de la méthodologie de génération des modèles de simulation proposés dans le cadre du flot de conception du groupe SLS.

Nous commencerons par présenter quelques exemples concrets des interfaces de communication pour l'adaptation des niveaux d'abstraction utilisés dans le flot de conception et nous continuerons en donnant quelques détails sur la réalisation de l'outil de génération automatique des modèles de simulation.

3.5.1. Illustration de l'interface de communication par quelques exemples concrets d'adaptation des niveaux d'abstraction

Cette section présentera quelques exemples concrets d'interfaces de communication pour l'adaptation des niveaux d'abstraction du flot de conception du groupe SLS. Ces niveaux d'abstractions ont été présentés dans la section 2.5.1 de ce mémoire. Plusieurs combinaisons des niveaux d'abstraction sont possibles ; nous illustrerons seulement les cas que nous considérons représentatifs.

Adaptation entre matériel au niveau RTL et canaux de communication de niveau macro-architecture

Ce type d'adaptation des niveaux d'abstraction est nécessaire dans l'hypothèse où un composant matériel de niveau RTL devrait communiquer pendant la simulation avec un réseau de communication de niveau macro-architecture. Nous utilisons comme exemple un module de niveau RTL communicant par un protocole rendez-vous avec un canal de communication de niveau macro-architecture qui encapsule le protocole *fifo* (voir Figure 30.a). Pour une telle adaptation, l'interface de communication est composée d'un adaptateur de port et un adaptateur de canal. Remarquons que dans ce cas, l'adaptateur de port correspond à trois ports groupés logiquement par le protocole rendez-vous. Supposons que le module doive lire une donnée du canal de communication, l'adaptateur de canal va lire la donnée en appelant la primitive de communication (dans notre exemple *receive_fifo*). L'adaptateur de port lit la donnée de l'adaptateur de canal en appelant une procédure (*CA ::get_data*), via le média de communication interne, il effectue la conversion de donnée (*data = (int) temp*) et il l'envoie au module par le protocole rendez-vous.

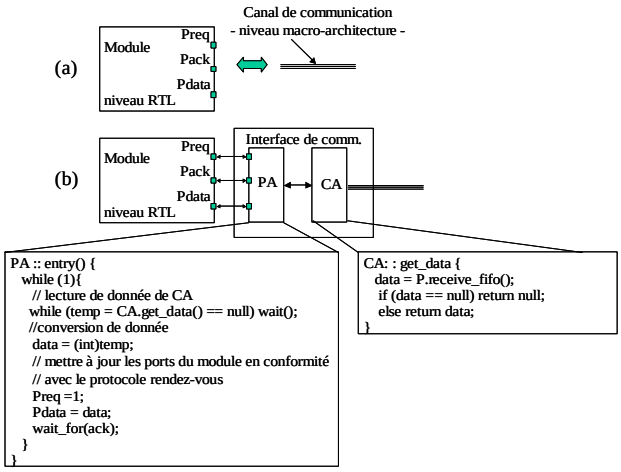


Figure 30. (a) module matériel au niveau RTL et canal de communication de niveau macro-architecture ; (b) module matériel au niveau RTL et canal de communication de niveau macro-architecture interconnectés par une interface de communication

Adaptation entre matériel au niveau comportemental et canaux de communication niveau RTL

Nous utiliserons comme exemple un composant matériel de niveau comportemental connecté à des canaux de communication de niveau RTL correspondant au protocole rendez-vous (voir Figure 31.a). Le module lit une donnée an appellent une primitive de communication de haut niveau (*Get_fifo()*).

L'interface de communication nécessaire dans ce cas est illustrée dans la Figure 31.b. L'adaptateur de canal CA lit la donnée du réseau de communication par des opérations de niveau RTL effectuées sur les signaux de niveau RTL ; ensuite il envoie cette donnée vers le PA par l'intermédiaire du média de communication interne implémenté par des RPC. L'adaptateur de port PA offre les primitives de communication de haut niveau (*Put_fifo(data)* et *Get_fifo()*).

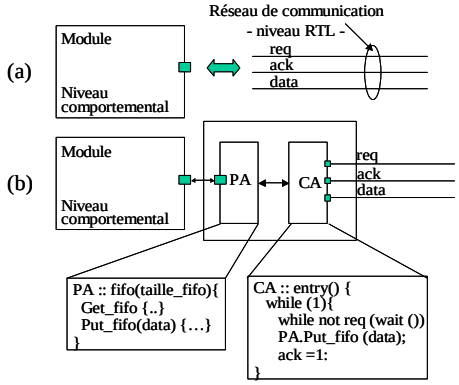


Figure 31. (a) module matériel au niveau comportemental et canal de communication de niveau RTL ; (b) module matériel au niveau comportemental et canal de communication de niveau RTL interconnectés par une interface de communication

Adaptation entre logiciel au niveau ISA et canaux de communication de niveau RTL

Les niveaux ISA et RTL ont été présentés dans le Chapitre 1 de ce mémoire. Dans une présentation sommaire, au niveau d'abstraction logiciel ISA le système d'exploitation est implémenté ; au niveau RTL la communication et les protocoles de communication sont détaillés par des fils physiques. La fonctionnalité de l'interface de communication consiste donc en la transformation des accès mémoires des pilotes du système d'exploitation et les accès sur les signaux du bus du processeur. Cela corresponds à la fonctionnalité d'un BFM tel que présenté à la section 3.2.3.

La Figure 32 illustre la structure de l'interface de communication nécessaire pour adapter les niveaux d'abstraction logiciel ISA et un réseau de communication RTL. Dans ce cas, le comportement des adaptateurs de port est très simple, ils transfèrent seulement les accès mémoire effectués par le logiciel, vers l'adaptateur de canal. L'adaptateur de canal transforme les accès mémoires en communications de niveau RTL. En effet, le CA représente l'implémentation d'un BFM.

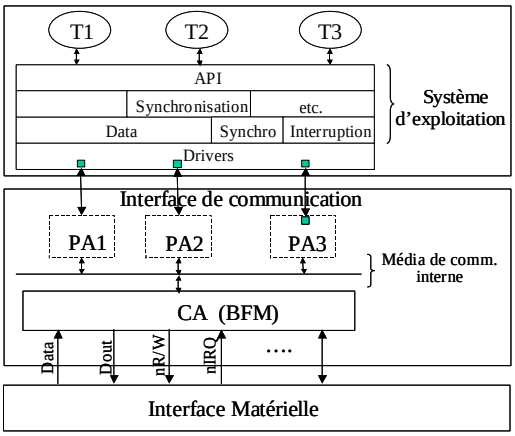


Figure 32. Exemple d'interface de communication pour l'adaptation entre logiciel au niveau ISA et canaux de communication de niveau RTL

Remarques :

- Dans le cas de cette adaptation la fonctionnalité des PA est très simple, l'interface de communication peut-être améliorée. Ainsi le logiciel peut être connecté directement au CA.
- En fonction du modèle de système d'exploitation utilisé, le logiciel peut être exécuté (1) sur un ISS - dans le cas où l'implémentation finale du système d'exploitation est utilisée ou (2) en natif - dans le cas de l'utilisation d'un modèle de simulation (un exemple de modèle de simulation du système d'exploitation sera présenté dans le Chapitre 4 de ce

mémoire). Dans tous ces deux cas, une interface de simulateur peut-être nécessaire. Cette interface n'a pas été représentée pour la clarté de l'explication.

Adaptation entre logiciel au niveau architecture générique et canaux de communication de niveau RTL

La Figure 33 illustre la structure de l'interface de communication pour l'adaptation entre logiciel au niveau architecture générique et canaux de communication niveau RTL. A titre de rappel, au niveau d'abstraction architecture générique le système d'exploitation est implémenté seuls les pilotes étant abstraits ; au niveau RTL, la communication et les protocoles de communication sont détaillés par des fils physiques.

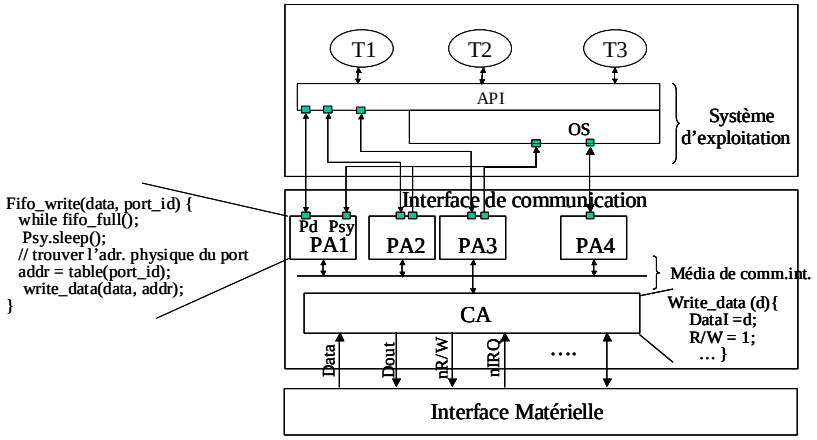


Figure 33. Exemple d'interface de communication pour l'adaptation entre logiciel au niveau architecture générique et canaux de communication de niveau RTL

Dans la Figure 33, pour une illustration plus claire nous avons séparé les API du reste du système d'exploitation. Les adaptateurs de port PA1, PA2, PA3 fournissent les pilotes appelés par le système d'exploitation et transfèrent les données vers/de l'adaptateur de canal via le média de communication interne. Un exemple de pseudo-code du PA1 est donnée dans la Figure 33. L'adaptateur de port PA4 est l'intermédiaire d'appel de la routine du traitement d'interruption (dans le cas où cette routine n'est pas implémentée dans le système d'exploitation, le PA doit la fournir). La fonctionnalité de l'adaptateur de canal CA est de transformer les accès mémoire effectués par les PA en des accès par des signaux physiques. Comme dans le cas de l'exemple précédent, le CA joue le rôle d'un BFM. Par conséquent, le CA de l'exemple précédent peut être également utilisé dans ce cas.

Remarques :

- Nous choisissons d'utiliser un PA pour chaque périphérique connecté au processeur exécutant le logiciel

- Dans cet exemple nous avons supposé que pour la simulation du logiciel un modèle de simulation du système d'exploitation a été utilisé (par exemple le modèle qui sera présenté dans le Chapitre 4 de ce mémoire). Le logiciel est donc exécuté en natif.
- Pour la même combinaison de niveaux d'abstraction, mais dans le cas d'un autre processeur, seul l'adaptateur de canal doit être remplacé dans l'interface de communication

3.5.2. Génération automatique de modèles de simulation dans le flot de conception des systèmes embarqués

Nous avons appliqué la méthodologie de génération automatique des modèles de simulation dans le cadre du flot de conception utilisé dans le groupe SLS. Comme il a été dit dans le Chapitre 2, tout au long de ce flot de conception, les différents composants logiciels/matériels d'un système ainsi que les canaux de communication peuvent être situés à différents niveaux d'abstraction. Nous avons défini :

- Trois niveaux d'abstraction pour le logiciel (le niveau architecture du système d'exploitation, le niveau architecture générique et le niveau ISA) ;
- Deux niveaux d'abstraction pour le matériel (le niveau comportemental et le niveau RTL) ;
- Deux niveaux d'abstraction pour la communication (le niveau macro-architecture et le niveau RTL).

Cela implique que dans notre flot de conception 10 combinaisons de niveaux d'abstraction[16] sont possibles (6 combinaisons niveau d'abstraction du logiciel – niveau d'abstraction des canaux de communication et 4 combinaisons niveau d'abstraction du matériel – niveau d'abstraction des canaux de communication). Toutes ces combinaisons sont résumées dans le Tableau 7. Parmi ces combinaisons, seules trois ne nécessitent pas l'existence d'une interface de communication en vue de l'exécution ; il s'agit des combinaisons notées avec 1, 7 et 10 dans le Tableau 7. Toutes les autres combinaisons de niveaux d'abstraction (les combinaisons 2-6, 8 et 9 dans le Tableau 7) nécessitent une interface de communication. Des exemples de telles interfaces de communication ont été illustrées dans la section précédente.

Notre méthodologie permet la validation des systèmes pendant les différentes étapes du flot de conception, par la génération automatique des instances de cosimulation. Cela implique la génération du bus de cosimulation, des interfaces des simulateurs et des interfaces de communication pour chaque combinaison de niveaux d'abstraction.

[16] une combinaison de niveaux d'abstraction inclue deux niveaux : un niveau de la communication et un niveau de module logiciel/matériel

No. de la combinaison	Niveau d'abstraction du module		Niveau d'abstraction des canaux communication	Nécessité d'interface de communication
1	Logiciel	Architecture du système d'exploitation	Macro-architecture	Non
2		Architecture générique	Macro-architecture	Oui
3		ISA	Macro-architecture	Oui
4		Architecture du système d'exploitation	RTL	Oui
5		Architecture générique	RTL	Oui
6		ISA	RTL	Oui
7	Matériel	Comportemental	Macro-architecture	Non
8		RTL	Macro-architecture	Oui
9		Comportemental	RTL	Oui
10		RTL	RTL	Non

Tableau 7. Combinaisons de niveaux d'abstraction dans le flot de conception du groupe SLS

La Figure 34 montre une vue globale de la génération automatique des modèles de simulation dans le cadre du flot de conception des systèmes embarqués. Le système que nous avons choisi comme exemple est un système composé de deux modules : un module comportant deux tâches (A et B) ciblé en logiciel et un module (C) ciblé en matériel. Dans la Figure 34 (bas) nous avons illustré également les principaux modèles de simulation nécessaires pour la validation. Dans cette figure, pour la clarté de l'explication nous avons omis les interfaces des simulateurs. Pour tous les modèles de simulation illustrés, le module matériel est exécuté sur un simulateur des modèles matériels (par exemple SystemC ou VHDL) et le comportement des canaux de communication est exécuté sur le simulateur utilisé pour la réalisation du bus de cosimulation (nous avons donc supposé que le comportement du réseau de communication est fourni par le bus de cosimulation). Le type de simulateur sur lequel le module logiciel s'exécute diffère d'un cas à l'autre et il sera donné par la suite, dans la présentation de chacun des trois modèles de simulation illustrés.

La Figure 34.a illustre le modèle de simulation du système dans l'hypothèse où le logiciel est situé au niveau architecture du système d'exploitation, le matériel est situé au niveau comportemental ou au niveau RTL et les canaux de communication sont situés au niveau macro-architecture ou au niveau RTL. Dans ce cas, le logiciel est exécuté sur un simulateur de haut niveau (par exemple SystemC). Les interfaces de communication illustrées dans cette figure peuvent être choisies parmi les interfaces 1 et 4 du Tableau 7 pour le module logiciel, et les interfaces 8 et 9 du Tableau 7 pour le module matériel.

Dans la Figure 34.b nous avons illustré le modèle de simulation où le module logiciel est situé au niveau architecture générique, les canaux de communication sont situés au niveau RTL et le module matériel est situé au niveau comportemental ou au niveau RTL. Dans ce cas, le logiciel est exécuté en natif, sur la machine hôte. Pour ce faire, un modèle de simulation du système

d'exploitation est nécessaire. Un tel modèle sera présenté dans le chapitre suivant de ce mémoire. Les interfaces illustrées dans cette figure correspondent à l'interface 5 du Tableau 7 pour le module logiciel et aux interfaces 8 ou 9 du Tableau 7 pour le module matériel.

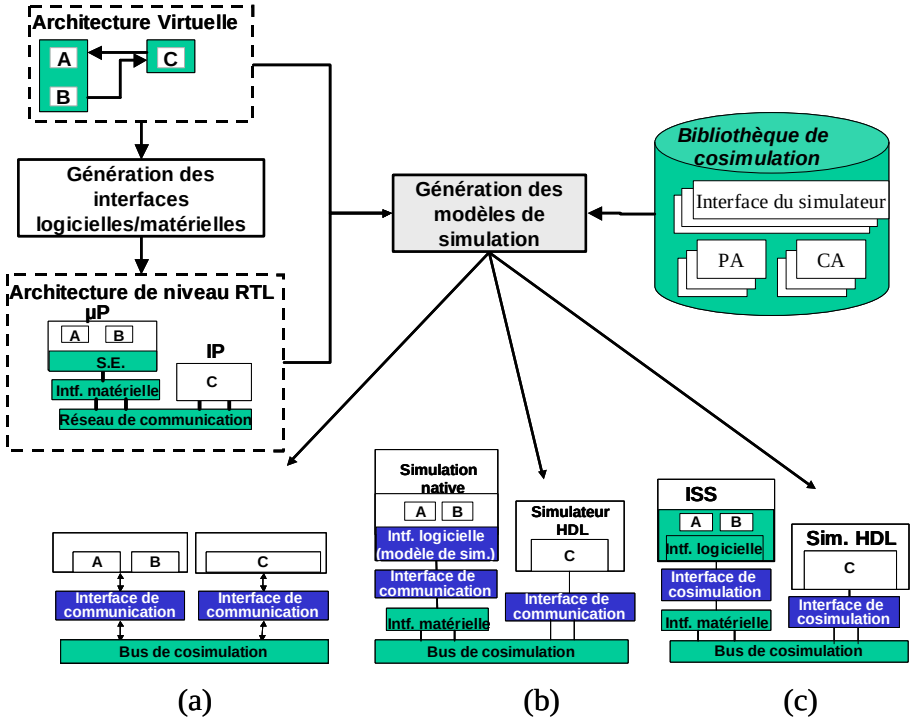


Figure 34. Génération automatique des modèles de simulation dans le flot de conception des systèmes embarqués

Dans la Figure 34.c le module logiciel et situé au niveau ISA, les canaux de communication sont situés aux niveau RTL et le module matériel est situé au niveau RTL ou au niveau comportemental. Dans ce cas, le module logiciel est exécuté sur un simulateur de processeur (ISS). Les interfaces de communication illustrées correspondent aux interfaces 6 du Tableau 7 pour le module logiciel, et les interfaces 8 ou 9 du Tableau 7 pour le module matériel.

L'application de la méthodologie de génération des systèmes de modèles de simulation dans le flot du groupe SLS implique l'adaptation des différentes étapes de la méthodologie présentée (voir la Figure 29) aux représentations spécifiques à ce flot. Le flot de conception du groupe SLS repose sur la représentation des systèmes en utilisant le modèle interne de représentation Colif. Notre objectif est donc de rendre simulable une telle représentation Colif afin de permettre la validation pendant les différentes étapes.

Choix pour le développement de l'outil de cosimulation

Le choix pour le développement de l'outil de cosimulation implique le choix de la réalisation de la bibliothèque de cosimulation et de la réalisation du flot de génération des instances de cosimulation.

Nous avons choisi de réaliser les différents éléments de la bibliothèque de cosimulation en utilisant le langage SystemC. Ainsi le bus de cosimulation, les composants de base de l'interface de communication (les adaptateurs de ports, les adaptateurs de canal et le média de communication interne) ainsi que les éléments pour la génération des interfaces du simulateur ont été spécifiés en SystemC.

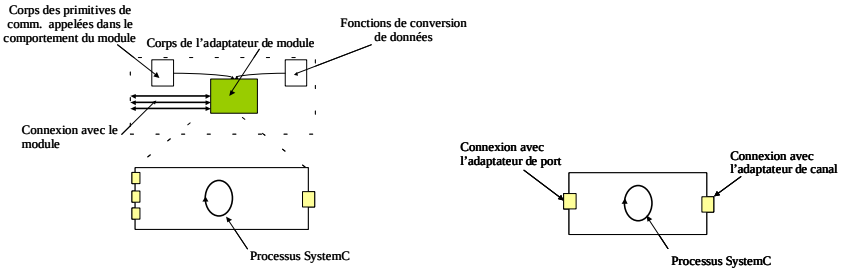
Quelques détails sur la réalisation des éléments de la bibliothèque ainsi que sur la réalisation du flot de génération des instances de cosimulation seront donnés par la suite.

Bibliothèque pour les interfaces de communication

Les différents adaptateurs de canal et adaptateurs de port de la bibliothèque sont des modules SystemC modélisant le fonctionnement défini dans la section 3.1.2.

La Figure 35.a montre la structure de l'adaptateur de port dans la bibliothèque de cosimulation. Nous distinguons les parties principales :

- Le corps de l'adaptateur de module qui est en charge de fournir les primitives de communication appelées dans le comportement du module à adapter et d'effectuer les différentes conversions sous-jacentes, les conversions de données.
- Les primitives de communication demandées dans le comportement du module sont implémentés séparément ; elles sont seulement appelées dans le corps de l'adaptateur de module. De même, pour les fonctions de conversions des types de données.
- L'interconnexion avec le module à adapter et avec le média de communication.



a) adaptateur de module **b) adaptateur de canal**

Figure 35. Structure des éléments de la bibliothèque de communication

Dans la même figure, nous avons détaillé la structure interne du corps de l'adaptateur. Ce dernier est un module encapsulant un processus SystemC. Ce module présente deux types de ports :

les premier type réunit les ports qui servent pour accéder au module et les ports pour l'interconnexion avec le média de communication.

L'adaptation proprement dite est réalisée par le corps de l'adaptateur.

L'adaptateur de canal n'effectue pas un arbitrage et il ne fournit pas des primitives de communication ; il est un simple processus SystemC. Sa structure est présentée dans la Figure 35.b.

Pour le média de communication interne nous utilisons les éléments de communication fournis dans la bibliothèque SystemC : la communication par des RPC, les événements, les signaux physiques.

Les éléments de la bibliothèque sont des modèles (objets *templates*) SystemC, elles peuvent être configurés (par exemple les types des données transférées, la taille des données ou les caractéristiques du protocole de la communication (par exemple la taille d'une FIFO) peuvent être des paramètres de configuration).

Bibliothèque pour les interfaces du simulateur

Concernant les éléments de bibliothèque pour les interfaces des simulateurs, elles sont basées sur la bibliothèque SystemC et la bibliothèque Unix pour la communication inter-processus, IPC (de l'anglais Inter-Process Communication). Chaque simulateur participant à la simulation est en fait un processus fils d'un processus SystemC avec lequel il communique par des IPC (dans le cas de notre réalisation nous utilisons les mémoires partagées et les sémaphores). Pour la communication des différents simulateurs par des IPC nous avons utilisé les différentes bibliothèques fournies par les simulateurs en vue de la communication avec un environnement extérieur (ex. CLI/PLI dans le cas du simulateur VSS, S-fonctions dans le cadre du simulateur Simulink/Matlab, etc.). Ces bibliothèques ont été conçues en utilisant le principe qui est décrit en [Lem 00].

Réalisation du bus de cosimulation

De même que les bibliothèques pour les interfaces de communication et les interfaces du simulateur, le bus de cosimulation est également réalisé en utilisant les bibliothèques de SystemC. Le bus de cosimulation doit pouvoir modéliser la communication de niveau macro-architecture ainsi que la communication de niveau RTL ou micro-architecture.

Pour modéliser la communication de niveau RTL nous utilisons les signaux et les ports fournis par SystemC. Pour la modélisation de la communication au niveau macro-architecture nous avons exploité la bibliothèque maître/esclave pour mettre en œuvre nos propres modèles de ports et de canaux de communication. La construction de ces modèles est présentée plus en détail dans l'Annexe A de ce mémoire.

Enchaînement des étapes du flot de génération des instances de cosimulation

Dans cette partie nous présenterons l'enchaînement des étapes du flot de génération des instances de cosimulation, dans le cas de l'outil de cosimulation que nous avons développé. L'enchaînement des étapes du flot de génération des instances de cosimulation est illustré dans la Figure 36. Nous avons surtout détaillé l'étape de construction des interfaces de cosimulation. Les éléments grisés correspondent aux principales actions effectuées pendant le flot de génération des instances de cosimulation ; les rectangles blancs représentent des informations extraites pendant une action ou bien des informations d'entrée pour une telle action.

L'étape de **l'analyse de la spécification** du système revient à l'analyse de la représentation de l'architecture virtuelle en Colif et sa translation dans une structure de données internes qui sera parcourue pendant les étapes suivantes.

L'étape de **construction des interfaces de cosimulation** inclut la construction des interfaces du simulateur et de l'interface de communication.

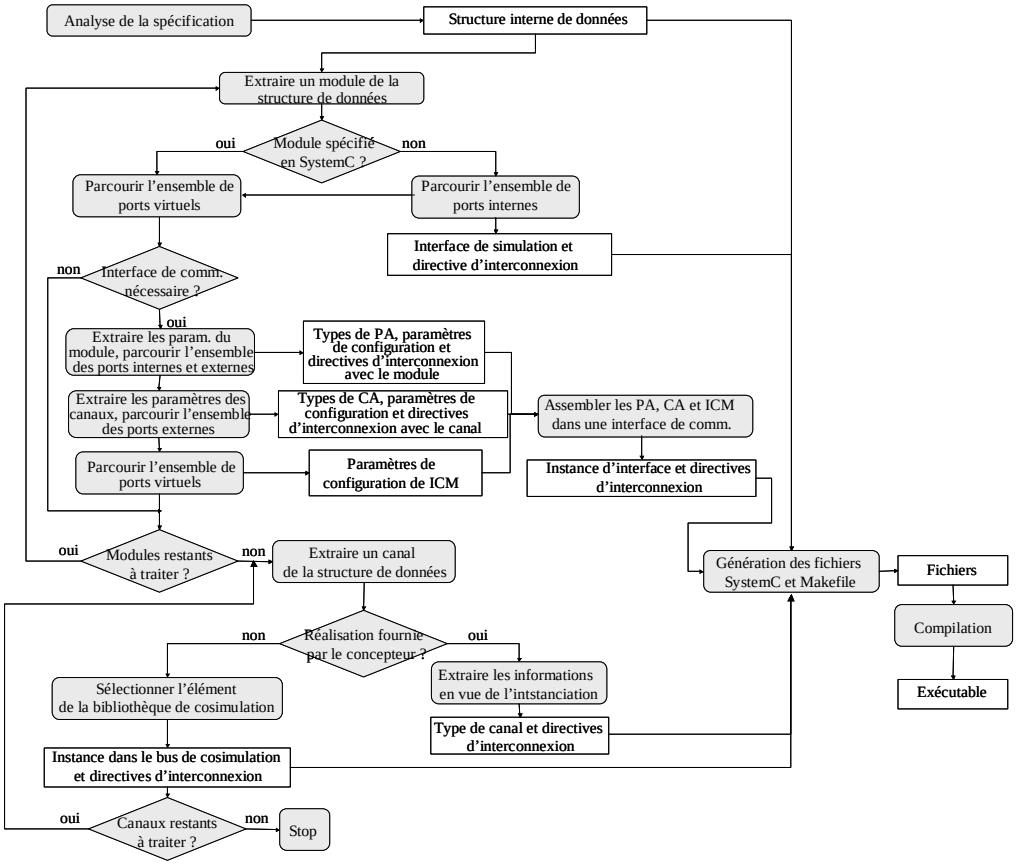


Figure 36. Enchaînement des étapes du flot des étapes du flot de génération des instances de cosimulation

Etant donné que le bus de cosimulation est implémenté en SystemC, une interface de simulateur sera construite pour chaque module qui n'est pas spécifié en SystemC. Pour la construction d'une telle interface, nous extrayons les paramètres des ports internes : le type de données, la direction et le niveau d'abstraction. Les éléments de la bibliothèque correspondant à chaque port sont sélectionnés et ajoutés à la structure interne représentant le système à simuler.

Pour la construction des interfaces de communication nous commençons par parcourir pour chaque module la liste des ports internes et externes du module pour tester la nécessité d'une telle interface. Dans le cas où le module nécessite une interface de communication nous analyserons l'enveloppe du module afin d'obtenir l'information pour la sélection des éléments de la bibliothèque. Ainsi, l'analyse du composant et de ses ports internes guidera la sélection des PA nécessaires. Le niveau d'abstraction du composant, le type du composant (ex. logiciel/matériel), le type de protocole de communication donneront le type du PA. Les types de données et les caractéristiques du protocole de communication (ex. taille de FIFO) donneront des paramètres pour la configuration du PA. Il est important de mentionner que pour la configuration d'un PA, il est nécessaire de parcourir les ports internes ainsi que les ports externes. Par exemple, dans le cas où les types de données des ports internes et des ports externes sont différents, le PA sera configuré avec le type de conversion nécessaire et le corps du PA (voir Figure 35.a) appellera la fonction de conversion correspondante.

De même, l'analyse du réseau de communication et des ports externes de chaque composant virtuel donnera les types de CA nécessaires et leurs paramètres : le niveau d'abstraction et le protocole de communication du canal virtuel donneront le type de CA ; les types de données et les caractéristiques du protocole de communication donneront des paramètres pour la configuration du CA.

Une fois que les types de CA et PA nécessaires ont été décidés, une table d'information sur la topologie de l'interface de communication et la configuration des composants est conçue. Cette table sera l'entrée de l'étape suivante du flot de génération des modèles de simulation. L'analyse de ports virtuels nous donnera les paramètres de configuration du média de communication interne.

L'étape suivante réalise **la construction du bus de cosimulation** de l'instance de cosimulation. Pendant cette étape, dans le cas où la réalisation de la communication n'est pas fournie, les éléments nécessaires sont sélectionnés de la bibliothèque de cosimulation. Cette sélection est guidée par les paramètres des canaux de communication (le niveau d'abstraction, le protocole de communication englobé et le type de données transférées).

L'étape d'intégration des interfaces de communications dans le modèle global du système correspond à la création d'un composant SystemC où les différents CA et PA sont instanciés avec les paramètres nécessaires et interconnectés avec les composants du système. Un fichier de

compilation est aussi généré. Plus précisément cette dernière étape est une étape de **génération de code SystemC**. Elle est donc dépendante du support de cosimulation choisi et elle est la seule à être modifiée dans le cas du changement de ce support.

3.6. Conclusion

Dans ce chapitre nous avons proposé un modèle générique de simulation pour les systèmes hétérogènes embarqués. Dans ce modèle, tous les composants du système sont connectés au réseau de communication par des interfaces de cosimulation qui agissent comme des adaptateurs. La structure modulaire de ces interfaces et la séparation entre l'adaptation du réseau de communication et l'adaptation des différents composants dans le modèle global de simulation nous ont permis la définition d'une méthodologie de génération automatique des modèles de simulation.

Cette méthodologie a été appliquée dans le cadre d'un flot de conception des systèmes hétérogènes embarqués. Elle permet la génération de modèles de simulation pour les systèmes hétérogènes spécifiés en utilisant le modèle de représentation Colif, présenté dans le Chapitre 2 de ce mémoire.

Chapitre 4. Modèle de simulation natif pour les systèmes d'exploitation enfouis

Sommaire

4.1. Introduction

Dans les systèmes embarqués le logiciel prend une place de plus en plus importante et sa complexité augmente d'autant. Pour pouvoir réaliser de tels logiciels, des systèmes d'exploitation sont souvent utilisés pour prendre en charge l'exécution concurrente et la communication des différentes tâches. Cette évolution de la complexité du logiciel rend difficile l'étape de validation en utilisant les techniques classiques.

La plus connue des techniques utilisées actuellement pour la validation du logiciel (l'application et le système d'exploitation enfoui) est l'utilisation des simulateurs de processeurs. Comme nous allons le montrer dans la section suivante de ce chapitre, ces simulateurs ont des vitesses insatisfaisantes pour la complexité des applications courantes. Ainsi ils ralentissent

considérablement le temps de validation et implicitement le processus de conception des systèmes. Pour cette raison il serait préférable de pouvoir utiliser la simulation native pour la validation du logiciel. La difficulté vient du fait qu'une partie du code du système d'exploitation enfoui comporte un code spécifique au processeur (par exemple du code assembleur). Ainsi la simulation native devient un défi : cela nécessite la définition d'un modèle de simulation pour le système d'exploitation. Un tel modèle doit permettre une validation rapide tout en étant cohérent avec la réalisation finale du système d'exploitation ; Il doit aussi offrir la possibilité de mesurer les performances du système d'exploitation au niveau temps d'exécution et de réaliser la simulation conjointe avec le matériel. Plusieurs travaux proposent aujourd'hui des modèles de simulation des systèmes d'exploitation mais comme nous allons le montrer dans la section 4.2.3 de ce chapitre aucun de ces modèles ne présente toutes ces caractéristiques.

Dans ce chapitre nous proposons un modèle de simulation des systèmes d'exploitation. L'idée centrale est de permettre la simulation native de la réalisation finale du système d'exploitation enfoui en utilisant des modèles de simulation seulement pour les services dépendants du processeur. La notion de temps est prise en compte, par l'ajout des annotations temporelles au système d'exploitation. Ce modèle est utilisé pour la validation des systèmes d'exploitation générés automatiquement dans le flot de conception présentés dans le Chapitre 2 de ce mémoire.

Ce chapitre s'articule autour de cinq sections. La section suivante présentera l'état de l'art sur la validation des systèmes d'exploitation pour les systèmes embarqués, et les contributions de notre travail. La troisième section présentera le modèle de simulation des systèmes d'exploitation que nous proposons. La quatrième section sera consacrée à la génération automatique de ce modèle de simulation. Pour terminer, dans la dernière section nous donnerons les conclusions.

4.2. La validation des systèmes d'exploitation enfouis

Pour la validation des systèmes d'exploitation enfouis, plusieurs approches ont été proposées dans la littérature. Ces approches utilisent des techniques de simulation ou des modèles de systèmes d'exploitation différents. Les principales techniques de simulation utilisées actuellement ainsi que les modèles de simulation des systèmes d'exploitation seront présentés par la suite.

4.2.1. Techniques de simulation pour le logiciel

Les deux techniques les plus utilisées actuellement pour la simulation du logiciel sont la simulation à l'aide d'un simulateur de processeur et la simulation native ; elles se distinguent par les environnements de simulation utilisés. Ces deux techniques seront présentées dans cette section.

Simulation à l'aide d'un simulateur de processeur (ISS - Instruction Set Simulator)

Un ISS est un outil qui s'exécute sur la machine hôte et qui émule la fonctionnalité d'un processeur ; il permet ainsi la simulation précise d'une application qui est censée s'exécuter sur le processeur. Un ISS modélise en fait le processeur au niveau de son jeu d'instructions ; chaque instruction définit une relation entre les éléments internes (par exemple les registres, la mémoire interne, etc.) ou externes (par exemple la mémoire externe) du processeur.

Actuellement les ISS sont souvent utilisés dans le processus de conception des processeurs et du logiciel. Leur rôle dans l'exploration de l'architecture, la validation et le développement pre-silicium du logiciel est indiscutable.

Les critères de qualité d'un ISS sont [Zhu 99] :

- La vitesse de simulation : elle est importante pour arriver à simuler des systèmes complexes dans des délais presque temps-réel. La vitesse de simulation dépend de la précision de l'ISS. L'ISS peut être précis au niveau de l'instruction ou au niveau des communications par les ports d'entrée/sortie.
- La vitesse de compilation : c'est la vitesse à laquelle un ISS transforme le code d'une application en une forme directement simulable.
- La traçabilité : la capacité de rassembler des statistiques de simulation comme par exemple le suivi de l'utilisation des instructions processeur.
- La portabilité : la capacité d'être étendu pour supporter de nouveaux processeurs cibles
- L'interopérabilité : la capacité de pouvoir interagir avec d'autres outils (ex. débuggeur, simulateurs matériels, etc.).

La plupart des ISS commerciaux disponibles sur le marché sont des **ISS interprétatifs** (qui utilisent la technique de simulation par interprétation). Un ISS interprétatif construit en mémoire une structure de données qui représente l'état du processeur cible. Il entre ensuite dans une boucle d'exécution dont le corps est constitué des étapes suivantes :

- Chargement : lit une instruction à partir de la mémoire.
- Décodage : analyse l'instruction et extrait le code opération (opcode) et les opérandes.
- Aiguillage : utilise une instruction *switch* pour se brancher, selon les cas, sur le code adéquat.
- Exécution : met à jour l'état du processeur suivant la sémantique de l'instruction à exécuter.

Cette modélisation d'un simulateur logiciel à l'aide des concepts spécifiques au matériel offre une bonne précision et la portabilité[17] mais malheureusement l'interprétation de chaque instruction réduit considérablement les performances en vitesse de ce type d'ISS. Cela est dû essentiellement au sur-coût d'exécution induit par les opérations de chargement, décodage et aiguillage pour chaque instruction du programme simulé, ce qui rend l'outil non productif : actuellement, la plupart des ISS interprétatifs exécute entre 10K et 100K instructions par seconde [Row 94] ce qui n'est pas acceptable pour simuler des logiciels comportant plusieurs milliers de lignes de code.

Par conséquent, pendant les dix dernières années plusieurs travaux de recherche ont été effectués sur les simulateurs de processeurs utilisant la technique de simulation par compilation [Ziv 96]. Dans ce cas, on parle des **ISS avec compilation**.

Dans le cas d'un ISS avec compilation chaque instruction du processeur cible est traduite directement en une suite d'instructions pour le processeur hôte. Ces dernières mettront à jour l'état du processeur simulé (cible). Par exemple, un code écrit pour un processeur MIPS et destiné à tourner sur une machine à base de ce processeur sera traduit en son équivalent en code SPARC si la machine hôte (la machine de simulation) est à la base de processeur SUN SPARC.

Cette traduction peut être réalisée au moment de la compilation du code source, on parle alors de simulation avec compilation statique et dans ce cas le surcoût d'exécution est quasiment éliminé. La traduction en code équivalent peut aussi être réalisée au moment du chargement de l'exécutable, on parle alors de simulation avec compilation dynamique. Dans ce cas, le surcoût est compensé à travers les boucles exécutant un même code. Les ISS avec compilation présentent des bonnes performances de vitesse. Cependant, ils présentent comme inconvénients le manque de portabilité, l'utilisation d'un grand espace de mémoire et une difficulté dans la modélisation des éléments spécifiques au processeur telles que les interruptions.

Par conséquence, malgré leurs problèmes de performances, les ISS interprétatifs sont encore utilisés dans le monde de la conception pour la validation précise au niveau cycle.

Simulation native du logiciel

La simulation native est réalisée sur la machine hôte (machine de développement) sans prendre en compte l'architecture interne du processeur cible sur lequel le logiciel sera exécuté.

Comme cette technique utilise le compilateur natif, le code spécifique au processeur cible (par exemple le code assembleur) ne pourra pas être exécuté, le code du logiciel à valider devant être écrit entièrement dans un langage de haut niveau indépendant du processeur cible (par exemple le langage C). Par conséquence, la simulation native des logiciels complexes (incluant plusieurs

[17] Ici, le terme de portabilité désigne la capacité de couvrir différents types d'architectures et d'applications

tâches et un système d'exploitation) nécessite un modèle de simulation pour le système d'exploitation.

L'avantage de cette technique de simulation est la vitesse, mais le prix à payer est la perte en précision : les codes spécifiques au processeur ne peuvent pas être exécutés. Cette approche requiert des efforts supplémentaires afin de pouvoir extraire des informations sur les performances au niveau temps d'exécution du logiciel à simuler.

4.2.2. Modèles de simulation pour les systèmes d'exploitation

Depuis peu pour la validation rapide du logiciel embarqué les modèles de simulation du système d'exploitation sont de plus en plus souvent utilisés. Cette section présentera les caractéristiques des principaux types de modèles de simulation proposés actuellement dans le monde de la recherche et de l'industrie.

Système d'exploitation abstrait

Le système d'exploitation peut être abstrait et fourni par un environnement de simulation. Dans ce cas, l'application est décrite à l'aide d'un langage de haut niveau comme un ensemble de tâches parallèles. La communication entre les différentes tâches est garantie par des primitives de communication fournies par le langage de spécification et l'ordonnancement est fourni par le moteur de simulation de l'environnement utilisé.

Système d'exploitation virtuel

Le système d'exploitation virtuel est un modèle qui imite la fonctionnalité du système d'exploitation final. Le but est de réaliser une simulation rapide de la fonctionnalité du système d'exploitation final et de l'application qui s'exécute sur ce système d'exploitation. Le système d'exploitation virtuel est donc une abstraction du système d'exploitation finale, où seule la fonctionnalité est respectée. Ainsi, un tel modèle est généralement conçu indépendamment du code final du système d'exploitation.

Réalisation finale du système d'exploitation

Finalement, pour une simulation complète et précise, l'exécution de la réalisation finale du système d'exploitation est nécessaire. Cela permet non seulement de fixer la fonctionnalité du système d'exploitation, mais aussi de débuguer son code final.

4.2.3. Etat de l'art sur la simulation des systèmes d'exploitation

Cette section présente les travaux portant sur la simulation des systèmes d'exploitation pour les systèmes embarqués. Les travaux existants utilisent la technique de validation native ainsi que la technique de validation à base d'un simulateur du processeur (ISS). Pour la validation native, les

modèles utilisés pour le système d'exploitation sont : le système d'exploitation abstrait et le système d'exploitation virtuel. Pour la validation à base d'un ISS, la réalisation finale du système d'exploitation est utilisée.

Simulation native en utilisant un système d'exploitation abstrait

Les systèmes d'exploitation abstraits sont généralement fournis par les environnements de simulation, tels que SystemC, ObjectGeode (pour la simulation du langage SDL).

Un tel modèle du système d'exploitation permet la validation rapide de la fonctionnalité des tâches. Cependant, aucune information de temps et aucune information sur le système d'exploitation s'exécutant sur l'architecture finale ne peuvent être obtenues.

Simulation native en utilisant un système d'exploitation virtuel

Actuellement, plusieurs approches proposent la simulation native du logiciel, en utilisant un système d'exploitation virtuel.

CarbonKernel [Car 02] fournit au concepteur un outil de développement des systèmes d'exploitation autour d'un système d'exploitation temps-réel virtuel. SoCOS [Des 00] permet aussi la modélisation des systèmes d'exploitation virtuels. Malheureusement, ces systèmes d'exploitation virtuels présentent le problème d'équivalence de code : le code du système d'exploitation virtuel est différent du code final du système d'exploitation (par exemple la politique d'ordonnancement), ce qui implique la nécessité d'une autre étape pour la validation du code final du système d'exploitation. Un autre problème que pose le système d'exploitation virtuel est le manque de portabilité pour la validation des différentes réalisations du même système d'exploitation. Pour la simulation d'une implémentation particulière le concepteur doit « personnaliser » le système d'exploitation virtuel, par l'ajout des nouvelles fonctionnalités. Généralement, la personnalisation est réalisée manuellement ; cela constitue une source d'erreurs et ralentit le processus de validation.

WindRiver Systems Inc. fournit VxSim comme modèle de simulation de VxWorks [Win 02]. Les désavantages de cette approche sont (1) de ne pas pouvoir simuler le logiciel avec le matériel situé autour du processeur sur lequel le système d'exploitation s'exécute et (2) une telle simulation n'offre pas la possibilité d'évaluer les performances du temps d'interaction entre le logiciel et le matériel.

Simulation de la réalisation finale du système d'exploitation à l'aide d'un simulateur de processeur (ISS)

Pour l'exécution de la réalisation finale du système d'exploitation, la simulation à base d'ISS est indispensable. Une telle simulation permet une validation précise du logiciel (l'application et le système d'exploitation) s'exécutant sur un processeur, mais comme nous l'avons montré dans la

section 4.2.1, ses performances en vitesse ont des conséquences de plus en plus importantes sur le temps de conception.

En résumé, la simulation native représente une solution avantageuse pour la simulation du point de vue performance en vitesse, mais les modèles de simulation utilisés ne sont pas assez précis (problème d'équivalence de code, l'évaluation des performances de temps ou la communication avec la partie matérielle ne sont pas fournies). La simulation à l'aide d'un simulateur de processeur reste une solution attractive, grâce à sa précision. Elle continue à être utilisée même si elle ralentit considérablement le temps de conception

4.3. Organisation du système d'exploitation

Pour la définition du modèle de simulation que nous proposons, la présentation de quelques éléments de base spécifiques aux systèmes d'exploitation est nécessaire. Ces éléments seront présentés dans la première partie de cette section. A l'aide de ces concepts nous présenterons , dans la deuxième partie le modèle de simulation.

Généralement un système d'exploitation peut être vu comme un ensemble de couches de services (un service étant une unité fonctionnelle élémentaire) : une couche API, une couche des services de base du système d'exploitation et une couche d'abstraction du matériel. Une présentation générale de ces couches a été effectuée dans la section 2.5.2 du Chapitre 2. Pour chaque service, plusieurs réalisations sont possibles.

Les services composant le système d'exploitation peuvent être spécifiques ou non au processeur. Pour les services spécifiques au processeur, leur réalisations diffèrent d'un processeur à un autre (par exemple la spécification est en code assembleur). La réalisation d'un service qui n'est pas spécifique à un processeur reste la même pour tous les processeurs.

La réalisation du système d'exploitation consiste en fait en la réalisation des différents services qui le compose. Généralement, une grande partie de la réalisation du système d'exploitation est indépendante du processeur et seulement une petite partie est spécifique au processeur. Par exemple, dans le cas du système d'exploitation utilisé dans notre approche (cf. section 2.5.2. du Chapitre 2), le code indépendant du processeur représente 90% du code total du système d'exploitation.

La Figure 37 illustre l'organisation du système d'exploitation dans le cas de notre approche :

1. **La couche API** (venant de l'anglais Application Programming Interface) représente les appels système de haut niveau invoqués par les tâches logicielles.
2. **Le noyau** ou kernel représente le cœur du système d'exploitation. Le noyau offre les fonctionnalités de base pour faire tourner aussi bien les applications utilisateurs que système,

ainsi que pour gérer, d'une manière efficace, les ressources matérielles sous-jacentes entre les différentes applications. Il contient les sous-familles de services suivantes :

- **L'amorce (Boot) :** regroupe tous les services liés au démarrage du système d'exploitation ; elle initialise les registres processeurs, la table des vecteurs d'interruptions, les espaces de piles, l'espace d'adressage, etc. Elle charge le noyau en mémoire (elle copie le code du noyau de la mémoire de masse où il est emmagasiné, à la mémoire vive (RAM)).
- **Changement de contexte :** regroupe tous les services liés à la gestion des contextes associés aux tâches (c'est à dire la sauvegarde des registres d'état de la tâche à suspendre et le chargement des registres d'état de la nouvelle tâche prête à être exécutée.
- **Ordonnanceur :** regroupe tous les services liés à l'ordonnancement des tâches. Pour cela, il utilise un algorithme de gestion, généralement par priorité ou tourniquet et gère l'ordre d'exécution des tâches.
- **Tâche :** regroupe tous les services liés à la gestion de tâches. En pratique, cette famille de services fait le lien entre les autres sous-familles de la famille Noyau (Kernel) : elle décrit la structure de la tâche et contient les tables de tâches.

On trouve ensuite des familles de services internes au système d'exploitation qui sont les suivantes :

- **Donnée :** regroupe tous les services liés aux structures de données de communication interne du noyau (comme des FIFO).
- **Interruptions :** regroupe tous les services liés aux interruptions ou aux appels système. Ces services répondent aux interruptions matérielles entrantes en appelant les routines de gestion adéquates.
- **Synchronisation :** regroupe tous les services liés aux mécanismes de synchronisation internes au système d'exploitation.

3. **La couche d'abstraction du matériel (HAL[18])** regroupe tous les services représentant les gestionnaires de périphériques. Elle peut posséder de nombreuses sous-familles, dont une sous-couche pour les services liés aux communications de bas niveau, elle gère étroitement les périphériques matériels (IO, mémoire, etc.).

[18] venant de l'anglais Hardware Abstraction Layer

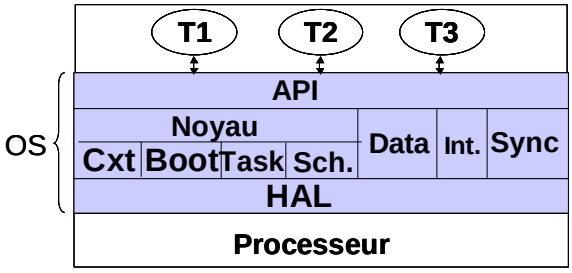


Figure 37. Représentation du système d'exploitation comme un ensemble de couches de services

4.4. Modèle de simulation natif pour la validation du code final du systèmes d'exploitation

Dans cette section nous présenterons un modèle de simulation natif pour la validation des systèmes d'exploitation. L'objectif est (1) de permettre la simulation native du système d'exploitation en respectant sa réalisation finale et en fournissant des informations temporelles, et (2) de permettre la génération automatique de ce modèle de simulation.

4.4.1. Réalisation du système d'exploitation versus modèle de simulation du système d'exploitation

La stratégie utilisée pour le modèle de simulation que nous proposons est :

1. L'utilisation du code final de la réalisation du système d'exploitation et l'ajout des annotations temporelles pour permettre une simulation prenant en compte la notion de temps.
2. Remplacer la réalisation des services spécifiques au processeur cible par un modèle fonctionnel de simulation implémenté d'une manière spécifique à l'environnement de simulation (ex. SystemC, SpecC, C-Unix).

La Figure 38 illustre une comparaison entre la réalisation finale du système d'exploitation et le modèle de simulation correspondant. Dans la Figure 38.a nous avons illustré le code de la réalisation finale du système d'exploitation. Quatre services du système sont utilisés comme exemple : deux services (Service 1 et Service 2) implantés par des codes indépendants du processeur cible et deux autres services (Service 3 et Service 4) sont implantés par le code dépendant du processeur cible. Le modèle de simulation correspondant à cette réalisation du système d'exploitation est illustré dans la Figure 38.b. Dans ce modèle, les même codes avec des annotations temporelles sont utilisés dans le cas des services 1 et 2 et des modèles fonctionnelles avec des annotations temporelles sont utilisés dans le cas des services 3 et 4.

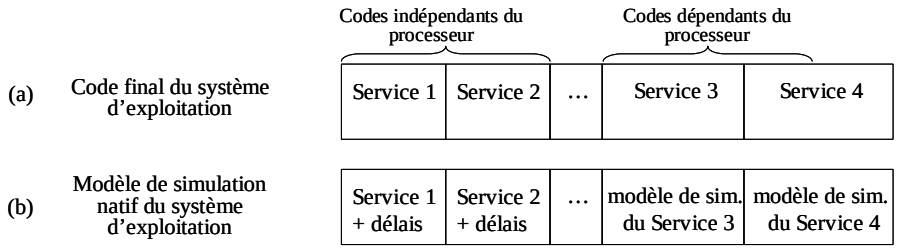


Figure 38. Réalisation du système d'exploitation (a) versus modèle de simulation du système d'exploitation (b)

Comme nous pouvons le remarquer dans la figure, en utilisant les mêmes codes dans la réalisation finale du système d'exploitation et dans son modèle de simulation, le problème d'équivalence de code des modèles virtuels du système d'exploitation peut-être réduit ou résolu partiellement. Si le code dépendant du processeur représente 10% de la réalisation finale totale du système d'exploitation, au moins 90% de notre modèle de simulation respectera la réalisation finale.

4.4.2. Calcul des temps d'exécution pour les annotations temporelles du modèle de simulation

Pour calculer les temps d'exécution nécessaires aux annotations temporelles dans le modèle de simulation nous utilisons des méthodes conventionnelles utilisées pour l'estimation du temps d'exécution du logiciel [Cad 02], [Laj 99], [Suz 96]. Pour obtenir les temps d'exécution spécifiques au processeur cible, ces temps sont calculés pour la cible et puis ils sont insérés dans le modèle de simulation en utilisant une fonction qui reçoit comme paramètre le temps d'exécution nécessaire. Cette fonction, que nous appellerons par la suite *delay* simulera l'avancement de temps qui lui a été donnée en paramètre.

4.4.3. Modèles de simulation pour les services indépendants du processeur cible

Dans le cas des services indépendants du processeur cible, pour obtenir les modèles de simulation nous insérons des annotations temporelles dans sa réalisation finale. Les annotations de fonctions *delay* simulent l'avancement du temps. Ces fonctions sont dépendantes du processeur cible.

La Figure 39 montre un exemple de modèle de simulation d'un service indépendant du processeur cible. Il s'agit du service de communication appelé *fifo_write.*

```
// Un example: service fifo

Void OS::fifo_write(int f_id, int data) {

        disable_interrupt(); delay(10);

        if( fifo_full(f_id) == true ) {
                enable_interrupt(); delay(5);
                block(f_id); // task execution is suspended.
                disable_interrupt(); delay(10);
        }

        write(f_id, data);
        enable_interupt(); delay(5);
}
```

Fonctions *delay* spécifiques *au processeur cible*

Figure 39. Code indépendant du processeur dans le modèle de simulation du système d'exploitation

4.4.4. Modèles de simulation pour les services spécifiques au processeur cible

Les exemples typiques des services spécifiques au processeur cible sont l'amorce (le boot), le changement du contexte, les routines de traitement d'interruption, les pilotes. Pour chaque service dépendant du processeur cible nous utilisons des modèles de simulation avec des annotations temporelles (en utilisant la fonction *delay* qui prend en paramètre le nombre de cycles d'horloge à simuler).

La Figure 40 montre un exemple de code assembleur pour le changement de contexte et son modèle de simulation. Dans la figure, une liste d'événements *wakeup_event* est utilisée pour suspendre une tâche (en utilisant la fonction *wait()*) et pour reprendre une tâche (en utilisant la fonction *notify()*). Le temps d'exécution nécessaire est aussi simulé en utilisant la fonction *delay*.

L'amorce initialise les registres, la table des vecteurs d'interruption, les espaces de piles, etc. Nous utilisons un modèle fonctionnel du code de l'amorce qui sera simulé à l'initialisation. Au démarrage de la simulation, pour sérialiser l'exécution des tâches logicielles, chaque tâche suspend son exécution en attendant l'événement de synchronisation venant de la part du service de l'ordonnancement (ex. *wakeup_event*). Comme dans notre modèle de simulation nous ne simulons pas l'architecture interne du processeur, les états des tâches tels que les registres et les piles ne sont pas simulés.

Pour la partie de notre modèle de simulation contenant les services dépendants du processeur nous avons le problème de l'équivalence du code. Comme le code assembleur représente seulement une petite partie du code total du système d'exploitation, ce problème a été atténué significativement.

```
__cxt_switch          ;r0, ancien pointeur de pile, r1, nouveau pointeur de pile
STMIA   r0!,{r0-r14}     ; sauvegarde des registres de la tâche courante
LDMIA   r1!,{r0-r14}     ; restauration des registres de la nouvelle tâche
SUB     pc,lr,#0         ; return
END
```

Changement de contexte: code assembleur pour le processeur ARM7

```
void context_sw(int cur_task_id, int new_task_id)
{
        delay(34);
        wakeup_event[new_task_id].notify();
        wait(wakeup_event[cur_task_id]);
        delay(3);
}
```

Changement de contexte : modèle de simulation en SystemC

Figure 40. Changement de contexte: code assembleur versus modèle de simulation

4.4.5. Simulation des routines du traitement des interruptions

Afin d'obtenir un modèle de simulation qui respecte les temps d'exécution, le gestionnaire d'interruptions doit être simulé. Chaque processeur présente différents types d'interruptions. Seulement une partie de ces interruptions sont nécessaires pour la validation du système d'exploitation. Par exemple les types d'interruptions spécifiques au processeur ARM sont : reset, instruction non-définie, interruption logicielle (SWI), data abort, prefetch abort, IRQ et FIQ [Jag 96]. Les interruptions qui sont liées à la validation du système d'exploitation sont SWI, IFQ, RQI. Leur modèle est donc nécessaire dans le modèle de simulation du système d'exploitation.

Pour la clarté de l'explication nous allons utiliser l'exemple de l'interruption SWI du processeur ARM7. La Figure 41.a montre un exemple de code assembleur pour la routine SWI (_SWI_Routine) et une section de code C qui appelle une fonction SWI générique appelée *__trap_trap* avec le SWI numéro 0 (défini par __swi(0)). L'appel de la fonction *__trap_trap(0,id,0)* à la ligne 17 déclenche un branchement à l'élément de la table de vecteurs correspondant à SWI, puis la routine SWI (voir ligne 2 dans la Figure 41.a) est exécuté.

Le modèle de simulation d'une telle routine d'interruption modélisera le nombre minimal d'éléments afin d'obtenir une simulation rapide. Dans le cas du processeur ARM, l'ensemble minimal d'éléments correspond aux registres de mode (CSPR et SPSR), qui contiennent des bits de contrôle tels que le masque d'interruption spécifique à chaque mode de fonctionnement du processeur.

La Figure 41.b illustre un modèle de la routine SWI pour la simulation du système d'exploitation. Les deux fonctions C, appelées *SWI_Enter* et *SWI_Return* modélisent l'entrée dans la routine SWI et le retour de la fonction SWI. Par exemple, la fonction *SWI_Enter* correspond aux lignes 3-7 de la Figure 41.a. Dans cette fonction, seul le changement des registres de mode (CPSR et SPSR) est simulé. Dans le code C qui appelle la routine SWI, les deux fonctions *SWI_Enter* et

SW_Return sont appelées auparavant, respectivement après l'appel de SWI (voir la Figure 41.b).

```
1.  // Assembly code for SWI routine
2.  _SWI_Routine
3.  STMIA   r13,{r0-r14}^       ; Push USER registers
4.  MRS     r0,spsr             ; Get spsr
5.  STMDB   r13!,{r0,lr}        ; Push spsr and lr_svc
6.  LDR     r0,[lr,#-4]         ; Load swi instruction
7.  BIC     r0,r0,#0xff000000
8.  BL      __trap_trap
9.  LDMIA   r13!,{r0,lr}        ; Pop return address and spsr
10. MSR     spsr_cf,r0          ; Restore spsr for swi
11. LDMIA   r13,{r0-r14}^       ; Restore registers
12.                             ; and return to user mode
13. NOP                         ; NOP
14. MOVS    pc,lr               ; Return from SWI

15. // C code to use SWI
16. __swi(0) void __trap_trap(int, int, int);
17. __trap_trap(0, id, 0);
```

(a)

```
// Modèle de simulation
// entrée dans SWI
SWI_Enter() {
CPSR_save = CPSR;
SPSR_save = SPSR;
CPSR = SVC;
}
```

```
// Modèle de simulation
// rentrée de SWI
SWI_Return() {
CPSR = CPSR_save;
SPSR = SPSR_save;
}
```

```
// Modèle de simulation
// Code C pour utiliser SWI
SWI_Enter(); delay(24);
__trap_trap(0,id,0);
SWI_Return(); delay(23);
```

(b)

Figure 41. Routine de traitement d'interruption. Code assembleur et code C correspondant

Afin de pouvoir prendre en compte les interruptions matérielles, la fonctions *delay* (utilisée dans notre modèle pour simuler l'avancement de temps) est sensible à ce type d'interruption. Ainsi, dans l'hypothèse où une interruption est envoyée par le matériel, la fonction *delay* saisira cet événement et appellera le modèle de simulation de la routine d'interruption (ISR). Pour une meilleure explication, la Figure 42.a illustre l'exemple d'une interruption arrivée pendant l'exécution de la fonction *delay* qui simule l'avancement de 10 cycles d'horloge. L'interruption arrive après la simulation des 5 cycles. Suite à cet événement, la fonction *delay* appelle le modèle de simulation de l'ISR. Après l'exécution de l'ISR, le temps global de simulation est avancé avec le temps nécessaire pour son exécution (20 cycles dans le cas de notre exemple). A la suite, la fonction *delay* simule l'avancement du temps restant (5 cycles dans le cas de notre exemple). Dans le cas où aucune interruption arrive pendant l'exécution de la fonction *delay*, la fonction simulera seulement l'avancement du temps avec un nombre de cycles d'horloge donné en paramètre (voir Figure 42.b).

La fréquence avec laquelle la fonction *Delay* est appelée influence la précision de la simulation des interruptions matérielles. Cependant, des appels trop fréquents de la fonction *Delay* peuvent dégrader les performances de la simulation. Ainsi la localisation des appels de la fonction *Delay* peut réaliser un compromis précision-performance.

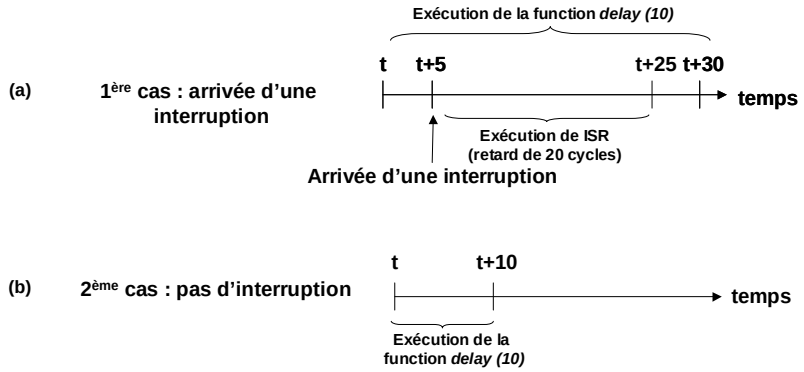


Figure 42. Interruptions matérielles et synchronisation

4.5. Génération automatique des modèles de simulation pour les systèmes d'exploitation

Pour la génération des modèles de simulation des systèmes d'exploitation nous utilisons la même méthode que celle utilisée dans le groupe SLS pour la génération/configuration de la réalisation finale du système d'exploitation. Cette méthodologie de génération des systèmes d'exploitation a été introduite dans le Chapitre 2 de ce mémoire. Elle est proposée et expliquée en détail dans [Gau 01]. A partir de cette méthodologie déjà définie et de l'outil de génération automatique des systèmes d'exploitation [Gau 01], nous avons seulement ajouté la fonctionnalité de génération des modèles de simulation.

4.5.1. Méthode de génération des systèmes d'exploitation utilisée dans le groupe SLS

L'idée de base de cette méthodologie pour la génération/configuration de la réalisation finale du système d'exploitation est de trouver les services nécessaires pour l'application et ensuite de générer leurs codes selon le processeur cible. Le système d'exploitation est obtenu par la composition des services générés.

La Figure 43, montre par un exemple, comment les services nécessaires à une application peuvent être trouvés. Sur cette même figure, les ovales représentent des services (et leurs codes) et les rectangles représentent des sections de code liées aux services. Chaque flèche représente la relation fournisseur - demandeur de service. Par exemple les services *Priority, Load, Wait, Wakeup* utilisent la section de code *Scheduling* et le service *ContextSwitch*. Une telle relation peut-être transitive : les services P et V utilisent les quatre services d'ordonnancement (*Priority, Load, Wait, Wakeup*) par le biais des services *BlockTask* et *UnblockTask* et des sections de code *Sync* et *Semaphore.* Ainsi, si les services P et V sont utilisés par l'application logicielle, en concordance avec la chaîne de dépendance montrée dans la figure, tous les services et les sections de code

montrées dans la figure doivent être inclus dans le système d'exploitation généré.

Les codes des services générés peuvent être dépendants ou non du processeur cible.

La génération des systèmes d'exploitation est réalisée à l'aide d'une bibliothèque. Cette bibliothèque (voir Figure 44.a) respecte l'organisation du système d'exploitation, comme elle a été présentée dans la section 4.3.1.

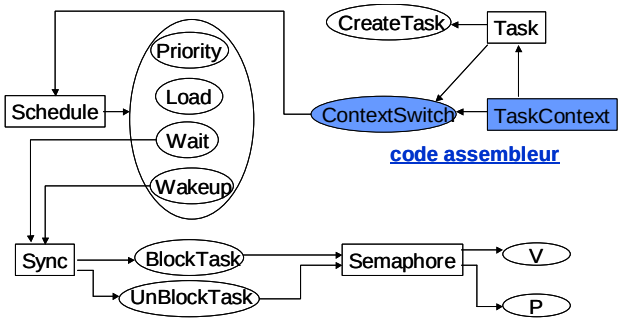
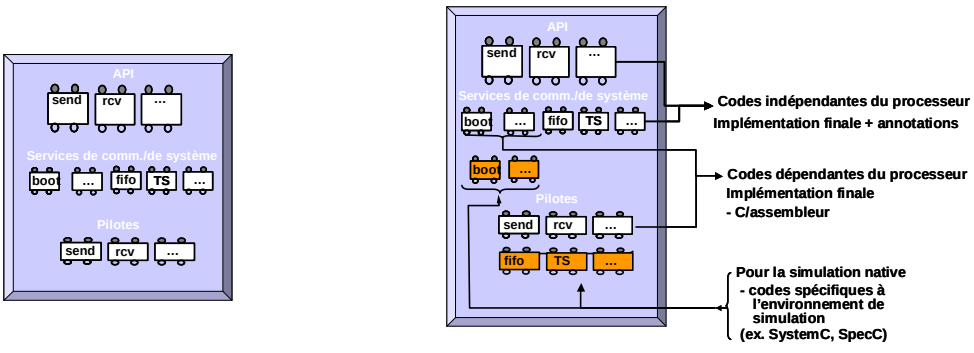


Figure 43. Exemple d'interdépendance des services

4.5.2. Génération des modèles de simulation pour les systèmes d'exploitation à l'aide de la méthodologie de génération des systèmes d'exploitation

La méthodologie de génération des systèmes d'exploitation peut être facilement adaptée pour la génération automatique des modèles de simulation pour les systèmes d'exploitation. Pour cela l'enrichissement des bibliothèques pour la génération de simulation est nécessaire. Contrairement aux services indépendants du processeur cible où seulement l'ajout des annotations temporelles est nécessaire, les codes dépendants du processeur cible nécessitent des modèles de simulation.

La Figure 44.b illustre la bibliothèque pour la génération des systèmes d'exploitation modifiée en vue de la génération des modèles de simulation.



(a) (b)

Figure 44. La bibliothèque de la génération des systèmes d'exploitation (a) versus la bibliothèque pour la génération des modèles de simulation pour les systèmes d'exploitation (b)

La Figure 45 montre l'intégration du modèle de simulation dans le flot du groupe SLS pour la conception de systèmes hétérogènes embarqués. Le modèle de simulation du système d'exploitation permet une validation rapide préliminaire du système d'exploitation généré automatiquement. Seulement une petite partie du système d'exploitation sera validée à l'aide de l'ISS. Concernant l'utilisation du modèle de simulation du système d'exploitation dans un modèle de simulation globale, des interfaces de cosimulation présentées dans la section 3.3 du Chapitre 3 sont utilisées. Le chapitre suivant présentera un exemple d'application du modèle de simulation dans une application concrète.

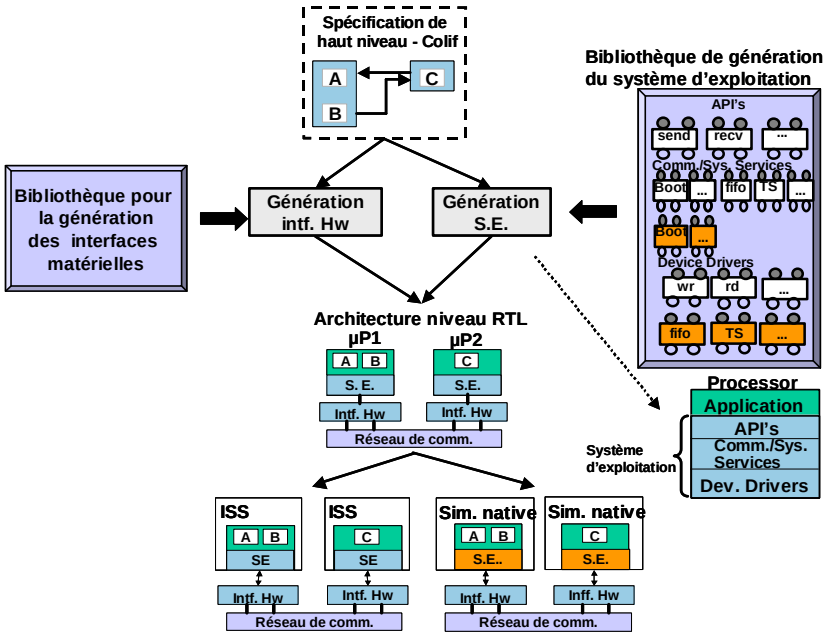


Figure 45. Intégration du modèle de simulation du système d'exploitation dans le flot de conception des systèmes hétérogènes

4.6. Conclusion

Grâce à sa précision, la simulation à l'aide d'un simulateur de processeur est aujourd'hui la solution la plus utilisée pour la validation de la partie logicielle d'un système embarqué. Malheureusement avec l'évolution de la complexité du logiciel, ces simulateurs ne sont plus assez rapides. C'est pourquoi il serait préférable d'exploiter les avantages en terme de vitesse de simulation offerts par la technique de simulation native pour la simulation du logiciel enfoui. Comme le code du système d'exploitation enfoui contient des parties spécifiques au processeur (par exemple du code assembleur), une telle simulation nécessite des modèles de simulation pour les systèmes d'exploitation. Ces modèles de simulation doivent permettre une validation à la fois rapide et précise et doivent être obtenus le plus vite possible.

Les modèles de simulation proposés actuellement ne sont pas assez précis (problème d'équivalence de code ; l'évaluation des performances de temps ou la communication avec la partie matérielle n'est pas fournie).

Dans ce chapitre nous avons présenté un modèle de simulation pour la validation des systèmes d'exploitation. Ce modèle permet la simulation native du système d'exploitation en respectant au maximum sa réalisation finale. Il fournit des informations temporelles et la possibilité de simulation avec le reste du matériel qui se trouve autour du processeur sur lequel le logiciel est exécuté. Le modèle de simulation que nous avons présenté peut être généré automatiquement, à l'aide de l'outil de génération des systèmes d'exploitation utilisé dans le flot de conception du groupe SLS.

Chapitre 5. Résultats expérimentaux

Sommaire

Les concepts mis en oeuvre dans les chapitres précédents pour la spécification et la validation des systèmes, ont été appliqués pour quelques applications complexes dont notamment :

- deux systèmes hétérogènes multiprocesseurs – un modem VDSL et un système de téléphonie mobile IS-95 CDMA
- un micro-système optique – le commutateur optique.

Ce chapitre présentera les expérimentations effectuées à l'aide de ces trois applications.

5.1. Spécification et validation d'un système multiprocesseur : le modem VDSL

5.1.1. Présentation générale du modem VDSL

Le VDSL (Very-high-data-rate DSL) est un protocole de communication utilisant les lignes téléphoniques qui fait partie des techniques xDSL (Digital Subscriber Line). Il est encore au stade de prototype, et de nombreuses entreprises proposent leur propre version du protocole VDSL.

L'application que nous avons utilisée pour nos expérimentations est un modem spécifique au VDSL. Le point de départ a été la réalisation du modem VDSL en utilisant des composants discrets [Mes 00]. Le schéma bloc de ce prototype est illustré dans la Figure 46.a. Dans ce schéma, l'ASIC et le FPGA sont utilisés pour le traitement du signal fixe, le DSP est utilisé pour le traitement du signal configurable et le MCU pour le contrôle du modem et l'interfaçage avec le PC hôte. La partie que nous avons reprise pour nos expérimentations est hachurée dans la Figure 46.a. Nous avons reparti la fonctionnalité du sous-ensemble hachuré sur deux processeurs ARM et un bloc matériel réalisant la chaîne de transmission (voir Figure 46.b). Ce partitionnement nous a été suggéré par l'équipe qui a réalisé le prototype du modem VDSL [Mes 00].

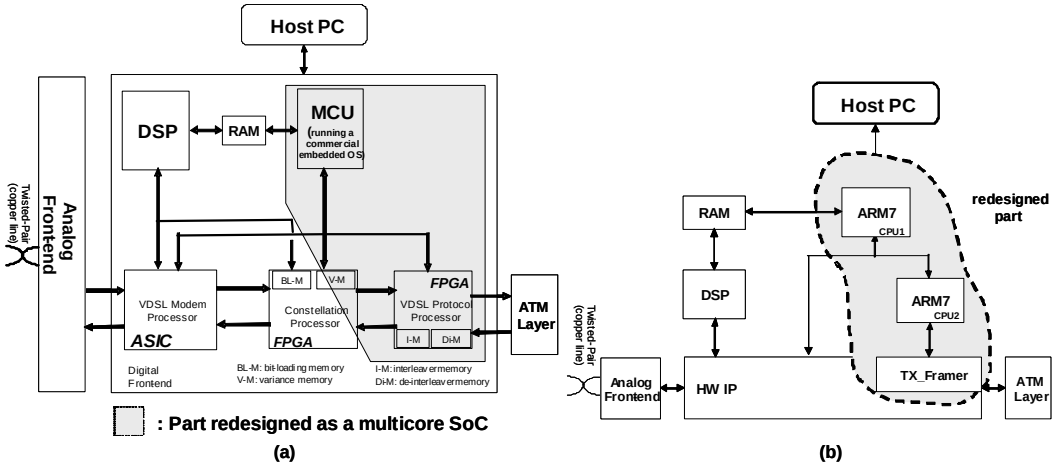


Figure 46. Le modem VDSL – la carte prototype (a) et l'architecture embarquée multiprocesseur (b)

5.1.2. Spécification du modem VDSL : l'architecture virtuelle

La spécification du sous-ensemble du modem VDSL sera composée de trois modules : **M1** et **M2** qui correspondent aux deux modules logiciels qui seront ciblés finalement sur des processeurs ARM et **M3** qui correspondent au bloc matériel réalisant la chaîne de transmission.

En terme de niveaux d'abstraction, les deux modules logiciels, M1 et M2 ont été spécifiés au niveau macro-architecture, le module M3 a été spécifié au niveau RTL et les différents modules du système communiquant par des canaux de niveaux macro-architecture (cf. section 1.2). En termes de langages de spécification, SystemC a été utilisé pour la spécification des différents composants

du système.

L'architecture virtuelle correspondant au modem VDSL est illustrée à la Figure 47. Comme la figure le montre, chaque module de l'application a été encapsulé dans un module virtuel.

Les enveloppes des modules virtuels VM1 et VM2 connectent les modules M1 et M2 de niveau macro-architecture avec les canaux de niveau macro-architecture. Comme entre ces modules et les canaux de communication auxquels ils sont connectés, il n'y a pas d'incompatibilités en terme de protocole de communication, niveau d'abstraction ou langage de spécification, les différents ports virtuels de ces enveloppes contiennent des ports externes et internes identiques. Cependant la présence de ces enveloppes est nécessaire dans la spécification ; elle représente l'abstraction des interfaces logicielles/matérielles qui seront générées automatiquement [Lyo 01], [Gaut 01] pour la mise en œuvre de l'architecture de niveau RTL correspondante à l'architecture virtuelle.

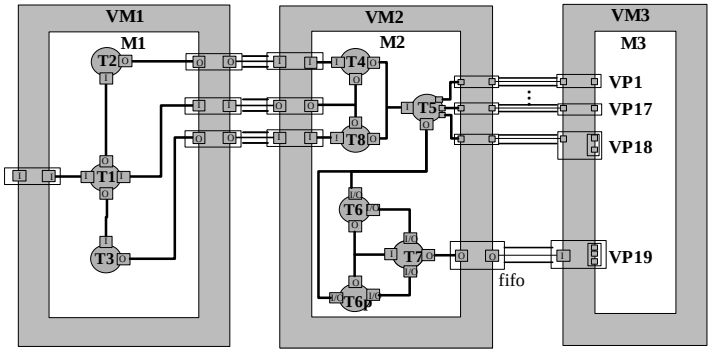


Figure 47. Architecture virtuelle du modem VDSL

L'enveloppe du module virtuel VM3 connecte le module M3 de niveau RTL aux canaux de communication de niveau macro-architecture. Cette enveloppe est donc composée des ports internes et externes de niveaux d'abstraction différents. Chaque port virtuel contient des ports internes de niveau RTL et des ports externes de niveau macro-architecture. Comme la Figure 47 le montre, les ports internes et externes sont groupés en ports virtuels. Par exemple, les trois ports internes du port VP19 du module virtuel VM3 dans la Figure 47 correspondent aux terminaux du protocole rendez-vous (la requête, la donnée et la confirmation de la fin de la communication) tandis que le port externe est un port abstrait permettant l'appel de primitives de communication *Get* et *Put* en conformité avec le protocole FIFO. Le port VP19 permet donc la connexion du module M3 de niveau RTL avec un canal de niveau macro-architecture encapsulant ce FIFO. De même pour les autres ports virtuels du VM3 : le port VP18 connecte les terminaux du protocole registre à verrou à un canal de communication de niveau macro-architecture; les ports VP17 jusqu'à VP1 connectent un port de niveau RTL correspondant à la connexion d'un fil physique à des canaux de niveau macro-architecture fournissant de primitives de communication.

5.1.3. Validation de l'architecture virtuelle – cosimulation multi-niveau

Le modèle de simulation pour la validation de l'architecture virtuelle présentée est illustré à la Figure 48. Une interface de communication (cf. section 3.2.) a été générée pour la connexion du module M3 avec le reste du système.

Les éléments de bibliothèque qui ont été nécessaires pour cette application sont illustrés dans le Tableau 8 ; ces éléments sont caractérisés par le niveau d'abstraction, le protocole et la direction.

Eléments de la bibliothèque	Niveau d'abstraction	Protocole	Direction
PA	RTL	Registre	In
			Out
		Registre gardé	In
			Out
		Rendez-vous	In
			Out
CA	Macro-architecture	Fifo	In
			Out

Tableau 8. Eléments de la bibliothèque utilisés pour la validation multi-niveau du modem VDSL

Les adaptateurs de ports sont des modules SystemC ; ils sont connectés avec le module par des ports de niveau RTL et avec le média de communication interne par un port maître qui appelle des procédures d'adaptateur de canal via le média de communication interne. Autrement dit, après avoir interprété les signaux de niveau RTL qui ont été actualisés par le module, l'adaptateur de port envoie la donnée dépourvue de tout protocole à l'adaptateur de canal, via le média de communication interne. Cet envoi déclenche automatiquement la fonction d'adaptation du canal adaptateur.

La Figure 49 montre un extrait du fichier SystemC généré ; ce fichier spécifie le modèle de simulation globale du modem VDSL. Nous avons sélectionné seulement les détails concernant un PA et un CA, plus exactement, le PA et le CA nécessaires pour adapter les ports internes, respectivement les ports externes du port virtuel VP19. Ces éléments sont illustrés dans la Figure 50. La Figure 49.a illustre l'instanciation du canal de niveau macro-architecture (ni21_VC21 dans la Figure 50) qui doit être connecté aux trois ports de niveau RTL du module M3 (Req, Data Ack dans la Figure 50). La Figure 49.b représente l'instanciation des sous-modules du modem VDSL et la Figure 49.c l'instanciation du port adaptateur et du canal adaptateur sélectionnés de la bibliothèque de cosimulation. La Figure 49.d montre les interconnexions des adaptateurs avec le module et le canal à adapter. La Figure 49.e correspond aux lignes de code nécessaires pour tracer des chronogrammes. Finalement, la section de code de la Figure 49.f lance l'exécution du modèle de simulation obtenu.

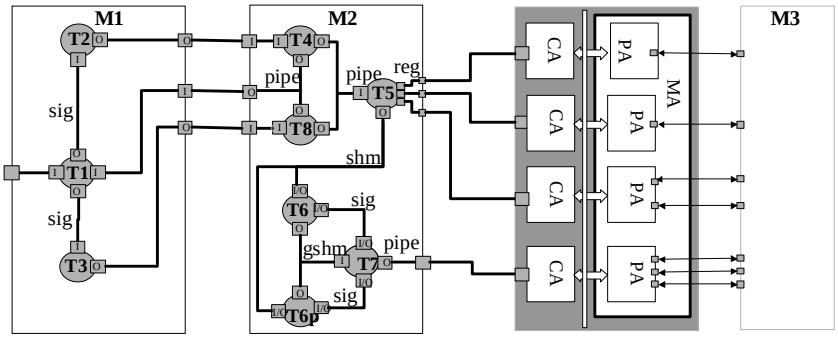


Figure 48. Modèle de simulation multi-niveau pour le modem VDSL

```
int sc_main(int ac, char* av[]){
(a)     ch_mac_pipe<long > ni21_VC21;
        sc_signal<bool> Req_VP19;
        sc_ signal <bool> Ack_VP19;
        sc_ signal <bool> Data_VP19;
        sc_mp_link<long int> icm_VP19;
        sc_clock s_clk("s_clk", 20.0, 0.5, 0.0);
        …
(b)     Ca_in_mac_reg<long int> *Ca_VP19;
        Pa_in_rtl_reg<sc_lv<11> > *Pa_VP19;
        …
(c)     m1 *M1;
        m2 *M2;
        m3 *M3;
        …
(d)     M2->VP2_PutVocWord(ni21_VC21);
        Ca_VP19->P(icm_VP19);
        Ca_VP19->Pch(ni21_VC21);
        Pa_VP19->P(icm_VP19);
        Pa_VP19->req(Req_VP19);
        Pa_VP19->ack(Ack_VP19);
        Pa_VP19->data(Data_VP19);
        …
(e)   sc_trace_file * my_trace_file;
        my_trace_file = sc_create_wif_trace_file("my_trace");
        sc_trace(my_trace_file, s_clk, "clock");
        sc_trace(my_trace_file, Acks_VP19,"Ack");
        sc_trace(my_trace_file, Req_VP19,"Req");
        sc_trace(my_trace_file,Data_VP19,"Data");
        ….
(f)     sc_start(2500);
        return(0);
}
```

Figure 49. Extrait du fichier généré pour la cosimulation multi-niveau

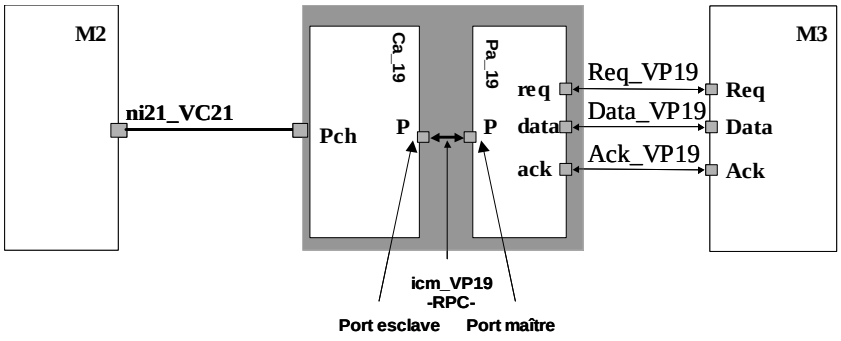


Figure 50. Interface élémentaire

Pour illustrer l'avantage de la génération automatique des modèles de simulation, nous avons évalué le nombre de lignes de code, ainsi que le nombre d'interconnexions pour la spécification initiale et pour le modèle de simulation généré automatiquement. Ces nombres sont groupés dans le Tableau 9. Notons le gain (en terme de temps et d'effort) que procure la génération automatique des modèles de simulation (réduction par un facteur à peu près 4).

Le temps de génération du modèle de simulation a été de 90s.

		Taille du code (no. de lignes)	**No. d'interconnexions**
Spécification de l'architecture virtuelle		150	21
Multi-niveaux	Modèle de sim. généré	475	60
	Temps de génération	90 s	
RTL	Modèle de sim. généré	600	187
	Temps de génération	10 min.	

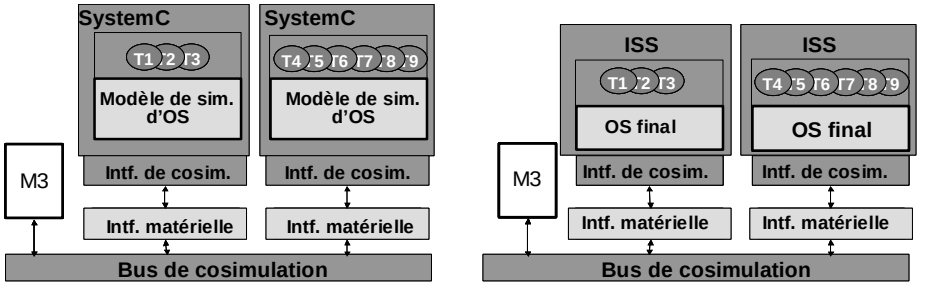
Tableau 9. Résultats de la génération automatique du modèle de simulation multi-niveau pour l'application VDSL

5.1.4. Validation de l'architecture de niveau RTL

La validation de l'architecture de niveau RTL pour le modem VDSL a aussi été effectuée. Pour ce faire, deux modèles de simulation ont été générés :

- Un modèle de simulation permettant la simulation native du logiciel (l'application et la couche du système d'exploitation). Dans ce cas, un modèle de simulation (cf. Chapitre 4) a été utilisé pour chaque système d'exploitation. Ce modèle de simulation pour le modem VDSL est illustré dans la Figure 51.a.
- Un modèle de simulation pour l'exécution du logiciel sur un ISS (dans le cas de notre expérimentation, le simulateur pour le processeur ARM). Ce modèle de simulation pour le modem VDSL est illustré dans la Figure 51.b. Dans ce cas, un BFM (cf. section 3.2) est nécessaire pour chaque processeur.

Comme la Figure 51 le montre, ces deux modèles de simulation ont une architecture semblable, les résultats en termes de génération automatique sont donc comparables. Ces résultats sont illustrés au Tableau 9.



a. Simulation native du logiciel **b. Simulation du logiciel à l'aide d'un ISS**

Figure 51. Modèles de simulation pour la validation de l'architecture de niveau RTL pour le modem VDSL

Par contre la différence se situe au niveau des temps de simulation. La simulation native du logiciel nous a fourni un gain en vitesse de 2 ordres de grandeur : la cosimulation avec le logiciel en natif a été 100 fois plus rapide que la cosimulation à base d'ISS.

5.1.5. Evaluation des résultats

Ces expérimentations nous ont permis de valider les concepts et la méthodologie que nous avons proposé :

- La possibilité d'utiliser le modèle d'architecture virtuelle pour la spécification d'une application complexe, composée des modules décrits aux niveaux d'abstraction différents
- La génération automatique du modèle de simulation d'une architecture virtuelle
- L'avantage du modèle de simulation pour les systèmes d'exploitation : le gain en performance en terme de vitesse.

5.2. Spécification et validation d'un micro-système optique : le commutateur optique[19]

Les micro-systèmes optiques ou MOEMS[20] intègrent sur une seule puce des sous-systèmes électroniques, mécaniques et optiques. Si ces systèmes étaient initialement présentés comme des idées abstraites, récemment ils sont de plus en plus populaires et ils sont commercialisés. Les micro-systèmes optiques promettent déjà de révolutionner les réseaux optiques avec des commutateurs optiques à base de miroirs, grâce aux avantages qu'ils présentent par rapport aux commutateurs classiques : gain en surface et vitesse, ils sont fiables et éventuellement leur conception sera moins coûteuse [Wu 97].

L'hétérogénéité de ces systèmes impose leur spécification en utilisant différents langages de spécification (ex. le langage C et HDL pour la description des parties électroniques, Matlab pour la description des parties mécaniques, etc.) ou différents niveaux d'abstraction. Les micro-systèmes optiques constituent donc un exemple très illustratif pour la méthodologie de spécification et de validation que nous avons proposée dans ce travail. Pour les expérimentations nous avons choisi un commutateur optique à base de miroirs.

5.2.1. Présentation générale du commutateur optique

La vue d'ensemble du commutateur optique que nous avons utilisé est illustrée à la Figure 52. Le système est composé de trois sous-systèmes :

- Le sous-système de contrôle qui commande les mouvements des miroirs mécaniques composant le sous-système optique

[19]Ce travail a été effectué en coopération avec le groupe MNS de laboratoire TIMA et l'Université Pittsburgh
[20] MOEMS – de l'anglais Micro-Opto-Electro-Mechanical systems

§ Le convertisseur électro-mécanique qui transforme le voltage envoyé par le sous-système de contrôle en commandes mécaniques en terme de position pour chaque miroir

§ Le sous-système optique composé par une matrice de miroirs de 2x2, deux sources de flux lumineux (G1 et G2 dans la Figure 52), et deux photo-détecteurs (D1 et D2 dans la Figure 52). Quatre lentilles (L1-L4 dans la Figure 52) sont utilisées : L1 et L2 sont des collimateurs qui limitent la divergence des flux lumineux provenant des sources G1 et G2 et les lentilles L3 et L4 focalisent le flux lumineux vers les photo-détecteurs.

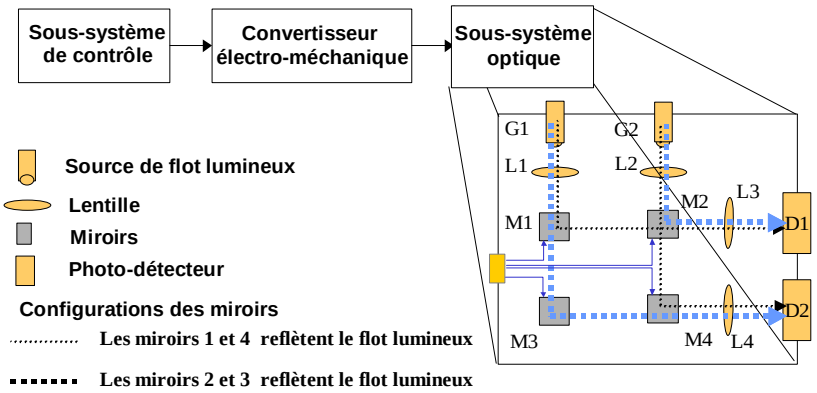


Figure 52. Le commutateur optique

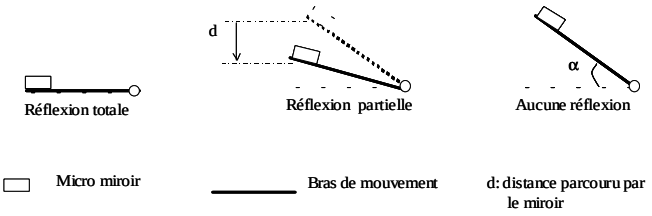


Figure 53. Le mouvement mécanique des miroirs conformément aux ordres du sous-système de contrôle

Les sources et les détecteurs que nous avons utilisés pendant ces expérimentations sont en fait de tableaux 3x3 des sources/détecteurs élémentaires des faisceaux. Ainsi, toute combinaison de 3x3 faisceaux lumineux peut être générée/détectée.

La fonctionnalité de commutation du commutateur optique est réalisée par le mouvement mécanique des miroirs qui guident le flot lumineux des sources des entrées (les sources G1 et G2 dans la Figure 52) vers les sorties (D1 et D2 dans la Figure 52). Le mouvement mécanique des miroirs est contrôlé par le sous-système de contrôle.

Pour assurer la bonne fonctionnalité de commutation, le sous-système de contrôle commande le positionnement des miroirs. Il existe différentes façons de bouger les miroirs et de modifier ainsi la

configuration optique. Par configuration nous désignons un état particulier des positions des miroirs. Pour cette application nous avons utilisé des miroirs fixés au bout d'un bras.

Comme la Figure 53 le montre, suivant l'intensité du courant électrique calculé par le processeur, le bras se colle à la surface en position horizontale (réflexion totale) ou il s'en écarte jusqu'à ne plus rien réfléchir dans la position la plus éloignée.

Ainsi, la matrice des miroirs composant le sous-système optique peut avoir deux configurations illustrées dans la Figure 52 :

- Dans la première configuration les miroirs M1 et M4 reflètent totalement le faisceau et les miroirs M2 et M3 ne saisissent pas le faisceau. Dans ce cas, le faisceau émis par G1 est guidé vers le photo-détecteur D1 et le faisceau émis par G2 est guidé vers le photo-détecteur D2.
- Dans la deuxième configuration les miroirs M2 et M3 reflètent totalement le faisceau incident et les miroirs M1 et M4 ne saisissent pas le faisceau. Dans ce cas, le faisceau émis par G1 est guidé vers le photo-détecteur D2 et le faisceau émis par G1 est guidé vers le photo-détecteur D1.

Remarquons que la commutation optique est réalisée par le passage d'une configuration à une autre.

5.2.2. Architecture virtuelle du commutateur optique

Pour la spécification globale du système optique, son architecture virtuelle a été réalisée. Les trois sous-systèmes interconnectés par l'architecture virtuelle ont été spécifiés indépendamment par trois groupes de travail différents. Ainsi, l'équipe MNS du laboratoire TIMA a réalisé la spécification du convertisseur électro-mécanique et l'Université Pittsburgh a réalisé la spécification de la partie optique. La spécification du sous-sytème de contrôle a été effectuée par le groupe SLS.

Les trois équipes ont seulement coopéré pour fixer la fonctionnalité globale du système mais aucune équipe n'a pris en considération les protocoles/niveaux d'abstraction ou les environnements de simulation utilisés par les autres équipes.

Ainsi, le premier groupe a spécifié le module de contrôle en SystemC, le deuxième groupe a spécifié le convertisseur électro-mécanique en Matlab/Simulink et le troisième groupe a modélisé le sous-système optique en utilisant Chatoyant. Chatoyant est un environnement de modélisation et de simulation des sous-systèmes optiques. L'idée de base est la création de bibliothèques C++ permettant la modélisation des concepts spécifiques à l'optique [Kur 01]. Le convertisseur électro-mécanique envoie les ordres mécaniques par des signaux simples, tandis que dans la spécification des miroirs, les ordres mécaniques sont reçus par un protocole « rendez-vous simple » (en anglais enable hand-shake).

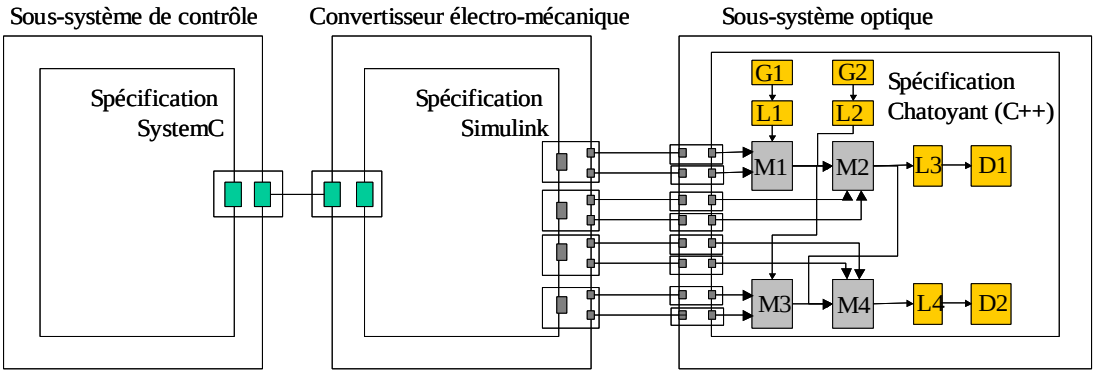


Figure 54. Architecture virtuelle du commutateur optique

L'architecture virtuelle du commutateur optique est illustrée à la Figure 54.

Comme le convertisseur électro-mécanique est décrit en Matlab/Simulink, son enveloppe contient des ports internes spécifiques à l'environnement de simulation Simulink. Ces ports assurent une communication continue par des signaux simples. Pour la connexion avec le reste du système, les ports externes sont des ports spécifiques à SystemC sur lesquelles la communication est réalisée à chaque cycle d'horloge. Comme la Figure 54 le montre chaque port virtuel de l'enveloppe du convertisseur électro-mécanique connecte un port interne du convertisseur avec deux ports externes, les deux ports étant en fait les terminaux du protocole « rendez-vous simple » utilisé par les miroirs. Par conséquent une interface du simulateur Simulink et une interface de communication pour adapter les protocoles de communication seront nécessaires.

Les ports virtuels composant les enveloppes du sous-système de contrôle et du sous-système optique sont constitués des ports de même type[21]. Pour ces deux modules, aucune interface ne sera donc nécessaire pour la simulation.

5.2.3. Validation de l'architecture virtuelle cosimulation multi-niveau multi-langage

Pour la validation de l'architecture virtuelle du commutateur optique, le modèle de simulation correspondant a été réalisé. Comme nous l'avons expliqué dans la section précédente, le modèle de simulation contient une interface de simulation et une interface de communication pour la communication du convertisseur électro-mécanique avec le reste du système.

[21] Pour la construction de l'architecture virtuelle, l'encapsulation des sources C++ Chatoyant de chaque composant optique en des modules SystemC a été nécessaire. Cette étape a été effectuée manuellement.

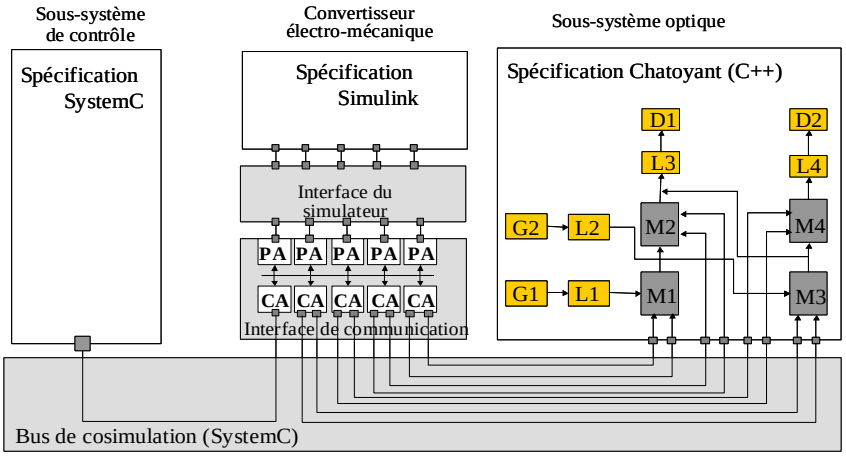


Figure 55. Modèle de simulation pour la validation du commutateur optique

Le modèle de simulation pour la validation du commutateur optique est illustré à la Figure 55.

5.2.4. Résultats

Dans notre expérimentation nous avons simulé l'évolution du commutateur optique entre les deux configurations de la matrice des miroirs (cf. section 5 .2.3), c. à. d. la reconfiguration du commutateur optique. Pour observer l'évolution du système, à chaque pas de simulation nous avons visualisé les faisceaux détectés par les photo-détecteurs D1 et D2.

Comme nous l'avons expliqué dans la section 5.2.1, les sources/détecteurs utilisés pendant cette expérimentation peuvent générer/détecter toute combinaison de 3x3 faisceaux lumineux. Pour faciliter l'analyse des résultats nous avons paramétré différemment les deux sources du flux lumineux : la première source génère un seul faisceau et la deuxième source génère une combinaison de quatre faisceaux lumineux Ceci est illustré à la Figure 56 où les neuf carrés représentent les possibles faisceaux et les points lumineux représentent les faisceaux générés.

Les résultats de détection des photo-détecteurs D1 et D2 sont illustrés à la Figure 57, où chaque ligne d'images correspond à un détecteur. Dans l'état initial de la simulation, le tableau de miroirs se trouvait dans la première configuration, donc après le premier pas de fonctionnement du commutateur, le détecteur D1 a détecté le flux lumineux généré par G1 et le détecteur D2 a détecté le flux lumineux généré par G2. Avec l'avancement de la simulation, suite aux commandes envoyées par la partie contrôle via le convertisseur électro-mécanique, chaque miroir change graduellement sa position ; donc la reconfiguration du commutateur se fera aussi graduellement, en plusieurs pas de fonctionnement. Ainsi, après le deuxième pas de fonctionnement, le miroir M4 a changé légèrement de position et il reflète partiellement le flux lumineux généré par G2 : trois faisceaux lumineux parmi les quatre générés. Finalement, après 11 pas de fonctionnement, les

miroirs sont arrivés en positions de réflexion totale (M2 et M3) ou non-réflexion (M1 et M4). Comme la Figure 57 le montre, à la fin de la simulation le détecteur D1 a détecté le flux lumineux généré par G2 et le détecteur D2 a détecté le flux lumineux généré par G1. La reconfiguration du commutateur optique a été donc réalisée.

Figure 56. Paramétrage des sources des faisceaux lumineux

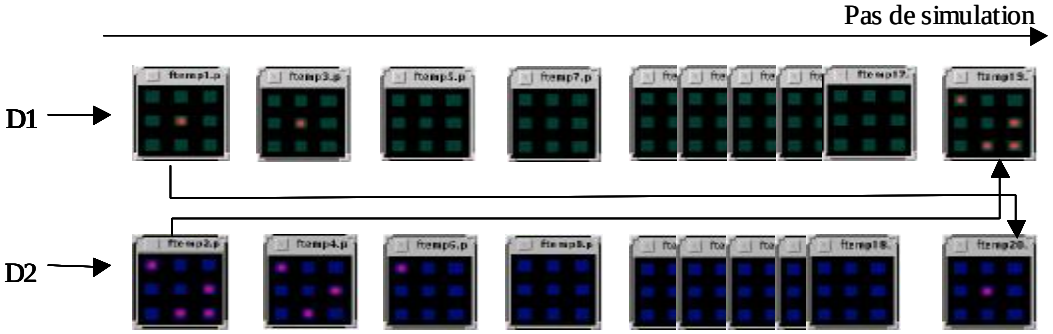


Figure 57. Reconfiguration du commutateur optique

La cosimulation du système a été effectuée sur une machine SUN Ultra-Sparc 1 et elle a demandé un temps de 30s. Cela nous a permis une validation rapide de la fonctionnalité globale du système avant l'implantation de l'architecture finale.

5.2.5. Evaluation des résultats

Ces expérimentations ont nécessité la communication entre les différentes équipes travaillant sur les différents sous-systèmes du commutateur optique. Les concepteurs de systèmes optiques et mécaniques n'étaient pas habitués avec les concepts de haut niveau et d'autre part notre défit a été de comprendre toute la sémantique des concepts optiques utilisés et des concepts physiques mécaniques. Ce problème a été surpassé grâce à l'utilisation de l'architecture virtuelle comme repère commun de tous les concepteurs. La majeure partie du temps a été utilisée à enrichir les bibliothèques de cosimulation et pour encapsuler les modèles C++ des composants optiques en modules SystemC.

La simulation globale du système nous a permis d'améliorer la fonctionnalité des miroirs mécaniques ainsi que la fonctionnalité globale du système.

Comme travaux futurs nous envisageons la simulation de l'architecture finale de ce système,

où le code de contrôle du système s'exécutera sur un simulateur de processeur et la communication avec le reste du système s'effectuera par des interfaces logicielles/matérielles.

5.3. Validation multi-niveau dans le cas du raffinement incrémental d'un système IS-95 CDMA

Cette section présentera l'application de la cosimulation dans le cas du raffinement incrémental d'un système IS-95 CDMA (venant de l'anglais Code Division Multiple Acces) [Yoo 99], [TIA 95]. Le système est initialement spécifié au niveau macro-architecture ; l'architecture cible de niveau micro-architecture de ce système sera obtenue par un raffinement incrémental. Comme nous l'avons expliqué au Chapitre 2 de ce mémoire, le raffinement incrémental permet le raffinement indépendant de la communication et des différents composants d'un système.

La première partie de cette section présentera l'application IS-95 CDMA. La deuxième partie expliquera le raffinement incrémental du système et la troisième partie donnera les modèles de simulation et les résultats de la cosimulation pour la validation de chaque étape du raffinement incrémental.

5.3.1. Présentation générale de l'application

La Figure 58 montre la représentation structurelle du système IS-95. Le système est composé de deux modems CDMA : un modem de transmission (le module modem Tx dans la figure) et un modem d'émission (le module modem Rx dans la figure), un codeur de voix, un décodeur, un processeur d'appel (le module CAP dans la figure). L'environnement du système, la station de base et l'interface avec l'utilisateur, est aussi illustré.

La voix est codée par le modem Tx et puis envoyée à la station de base. La voix codée de l'autre interlocuteur dans la conversation est envoyée par la station de base, reçue par le modem Rx et décodée par le décodeur.

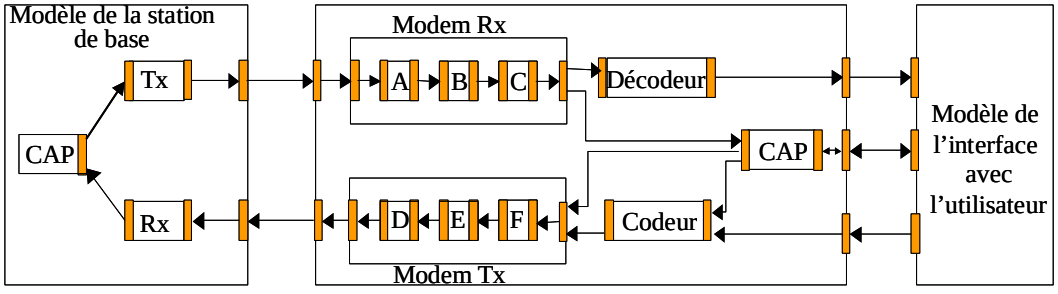


Figure 58. Le système IS-95 CDMA – représentation structurelle

5.3.2. Raffinement incrémental du système IS-95

Dans le cadre de notre expérience, nous avons considéré que la spécification de niveau macro-architecture du système IS-95 CDMA a été raffinée dans une spécification de niveau RTL

(micro-architecture), à travers plusieurs étapes intermédiaires ; pendant toutes ces étapes, seule une partie du système a été raffinée au niveau RTL. L'objectif de notre expérience est de réaliser la validation par cosimulation pendant toutes les étapes des deux raffinements incrémentaux possibles présentées à la Figure 59.

Dans le premier cas, le raffinement incrémental commence par une étape de raffinement au niveau RTL de la communication et du module *Décodeur* : le module *Décodeur* a été ciblé en logiciel et s'exécutera sur un processeur ARM7 ; les interconnexions abstraits de niveaux macro-architecture ont été déployées en interconnexions physiques. Dans une deuxième étape intermédiaire, le module *Codeur* est aussi raffiné au niveau RTL : il est ciblé en logiciel et s'exécutera sur un processeur ARM7. Une dernière étape, raffinera les modules *modem Tx* et *modem Rx* : ce seront des modules logiciels s'exécutant sur des processeurs Motorola 68000 (M68K). Le ciblage en logiciel de chaque module implique l'ajout d'une interface matérielle pour la connexion du processeur au réseau de communication (cf. section 2.5.2 du Chapitre 2). Dans le cas de cette expérience, chaque application logicielle est décrite par une seule tâche ; donc nous n'avons pas utilisé de système d'exploitation. Ce raffinement incrémental est résumé dans la partie gauche de la Figure 59.

Par analogie, la deuxième alternative de raffinement incrémental est illustrée dans la partie droite de la Figure 59.

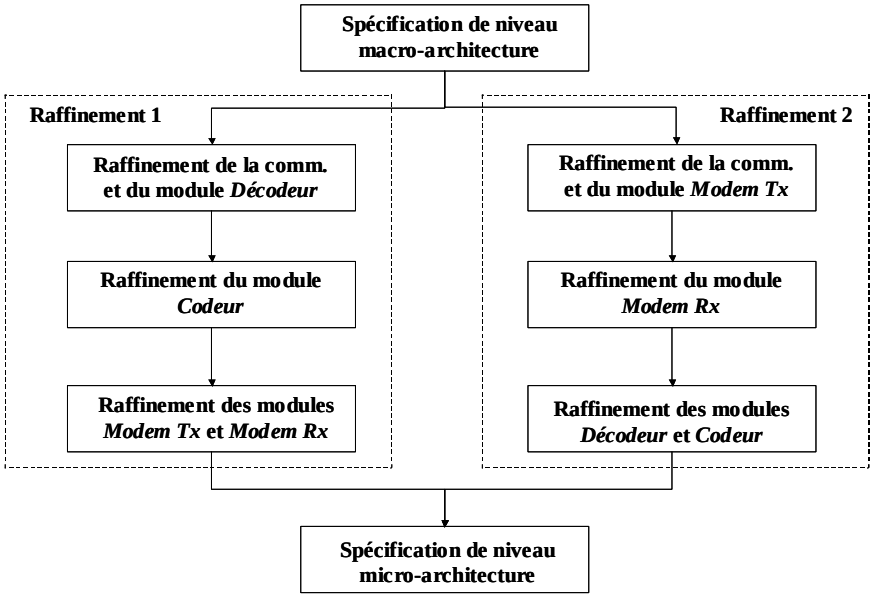


Figure 59. Raffinement incrémental du système IS-95 CDMA

5.3.3. Modèles de simulation pour la validation dans le cas du raffinement incrémental du système IS-95 CDMA

Pour la validation des différentes étapes du raffinement incrémental, la construction des modèles de simulation a été nécessaire. La Figure 60 illustre les modèles de simulation de chaque étape du premier cas de raffinement incrémental présenté dans le paragraphe précédent.

La spécification de niveau macro-architecture a été réalisée en SystemC. La validation de niveau macro-architecture a donc impliqué une simulation pure SystemC. Une fois que le module *Décodeur* a été raffiné et ciblé en logiciel, il s'exécutera sur un ISS et une interface de cosimulation est nécessaire pour l'intégration dans le modèle de simulation du système entier. Cette interface est composé d'une interface du simulateur (pour intégrer l'ISS dans le modèle de simulation) et une interface de communication (cf. section 3.5.1 du Chapitre 3) ayant principalement le comportement d'un BFM (la transformation des accès mémoire fonctionnels en des accès de niveau cycle-près). Comme nous l'avons expliqué dans le paragraphe précédent, pendant la première étape du raffinement incrémental, la communication est aussi raffinée au niveau RTL. Ainsi, pour la cosimulation, une interface de communication a été nécessaire pour connecter chaque module de niveau macro-architecture avec le réseau de communication. Le modèle de simulation du système IS-95 CDMA pour la validation de la première étape du raffinement incrémental est illustré à la Figure 60.b. Les interfaces ajoutées pour la cosimulation sont grisées. La Figure 60.b, c. illustre les modèles de simulation pour les étapes suivantes où le module *Codeur*, ensuite les modules *modem Tx* et *modem Rx* ont aussi été raffinés. Dans une dernière étape tous les modules du système à concevoir ont été raffinés au niveau RTL, ciblé sur des processeurs (ARM7 et M68K). Ils sont simulés à l'aide des ISS intégrés dans le modèle de simulation du système via des interfaces de cosimulation. Les deux modules modélisant l'environnement (le modèle de la station de base et de l'interfaçage avec l'utilisateur) étant resté au niveau macro-architecture sont connectés au réseau de communication via des interfaces de communication. Les interfaces de communication utilisées dans le cas de cette expérience sont des interfaces qui adaptent les modules de niveau macro-architecture au réseau de communication de niveau micro-architecture.

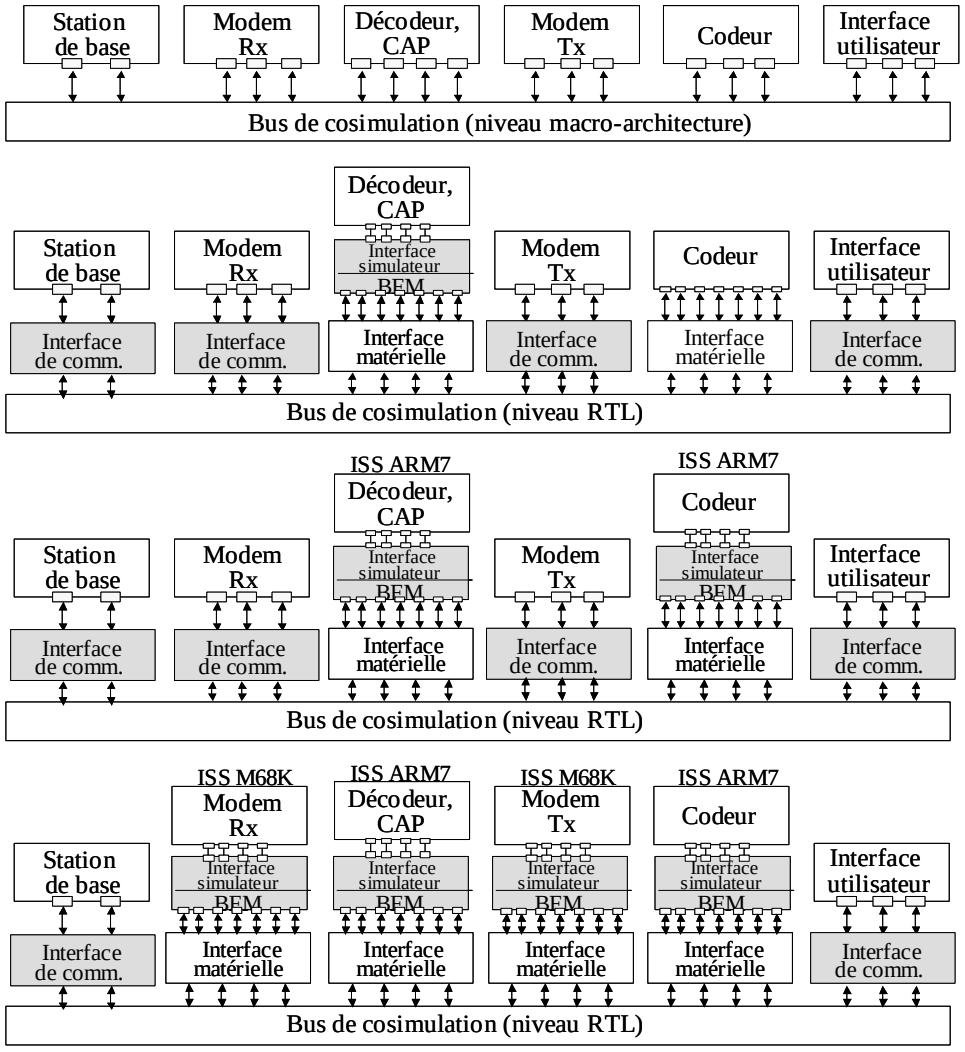


Figure 60. Modèles de simulation pour la validation dans le cas du raffinement incrémental du système IS-95 CDMA

5.3.4. Résultats de la cosimulation

Les résultats de la cosimulation en termes de temps de simulation sont groupés dans le Tableau 10. Chaque ligne du tableau correspond à un cas de cosimulation ; chacune des colonnes 2, 3, 4, 5 groupés sous le nom *Niveaux d'abstraction* correspond à un module du système – ces colonnes indiquent le niveau d'abstraction de chaque module pour chaque cas de cosimulation ; finalement, la dernière colonne montre les temps de cosimulation.

Pour tous les cas de cosimulation, nous avons simulé le comportement du système pour 20 trames de voix (ce qui correspond à 0 .4 secondes de conversation réelle). La fréquence de l'horloge du processeur ARM7 a été de 40 MHz et la fréquence de l'horloge du processeur Motorola M68K a été de 10MHz.

Pour la spécification de niveau macro-architecure, le temps de simulation a été de 42 secondes. Dans le deuxième cas où le module de décodage et la communication ont été raffiné la cosimulation a pris environ 13000 secondes. La validation du système au niveau micro-architecture a nécessité 20 heures de cosimulation. Ces résultats montrent la flexibilité et l'extensibilité du modèle de simulation que nous utilisons ainsi que l'avantage de la cosimulation multi-niveau afin de réaliser un bon compromis entre les performances et la précision de la validation.

No. du cas	Niveaux d'abstraction				Temps de simulation Secondes (heures)
	Décodeur	Codeur	Tx	Rx	
1	MA	MA	MA	MA	42 (0,01)
2	ISA (ARM7)	MA	MA	MA	13068 (3.6)
3	ISA (ARM7)	ISA (ARM7)	MA	MA	26 606 (7.3)
4	MA	MA	ISA (M68K)	MA	4 604 (1.2)
5	MA	MA	ISA (M68K)	ISA (M68K)	11 072 (3.1)
6	ISA (ARM7)	ISA (ARM7)	ISA (M68K)	ISA (M68K)	72 744 (20.2)

Tableau 10. Temps de simulation pour la validation du système IS-95 CDMA dans le flot de raffinement incrémentiel

5.4. Conclusion

Ce chapitre a présenté l'application des concepts proposés sur des applications complexes : un système embarqué multiprocesseur - le modem VDSL, un micro-système optique – le commutateur optique et un système de téléphonie mobile IS-95 CDMA. Les objectifs ont été : (1) de prouver l'utilité et l'efficacité du modèle d'architecture virtuelle, (2) d'appliquer la méthodologie de validation sur des applications complexes réelles et prouver la flexibilité, la modularité et l'extensibilité du modèle de simulation proposé (3) de montrer les avantages du modèle de simulation du système d'exploitation. L'architecture virtuelle s'est avérée un repère très efficace commun pour les concepteurs pendant la spécification d'un système hétérogène est dans le même temps une entrée commune pour différents outils (génération des modèles de simulation, génération des interfaces logicielles/matérielles). Concernant la méthodologie de validation des systèmes hétérogènes, le temps de génération des modèles de simulation a prouvé un gain important de temps de conception. Ce gain vient s'ajouter à l'augmentation de la fiabilité du modèle de simulation. L'expérience sur la validation du système IS-95 CDMA a prouvé la flexibilité, la modularité et l'extensibilité du modèle de simulation utilisé.

Finalement, la simulation globale du modem VDSL au niveau RTL en utilisant un modèle de simulation du système d'exploitation a prouvé un gain de vitesse de deux ordres de grandeur et la faisabilité d'intégration d'un tel modèle de simulation dans un flot de conception.

Conclusion et perspectives

Conclusion

La tendance très récente dans la conception des systèmes embarqués est l'assemblage des composants standards existants. La difficulté d'un tel flot de conception provient du fait que ces composants sont hétérogènes en termes de protocoles de communication, niveaux d'abstraction et langages de spécification. Dans cette étude nous avons abordé deux principaux problèmes liés à la conception des systèmes embarqués hétérogènes : la spécification et la validation.

Le chapitre 1 a présenté une étude pragmatique et synthétique sur la spécification des systèmes embarqués. La première partie a défini les concepts structurels de base et les niveaux d'abstraction utilisés au cours de la conception des systèmes embarqués. Ensuite, la variation des concepts de base à travers les différents niveaux d'abstraction a été présentée. A la lumière de ces concepts, dans une deuxième partie de ce chapitre nous avons entrepris une étude des solutions proposées pour la spécification des systèmes. Dans un premier temps nous avons étudié les langages de spécification le plus souvent utilisés pendant le processus de conception. En montrant qu'aucun langage ne convient parfaitement pour la modélisation de tous les concepts de base, nous avons effectué une analyse des solutions innovatrices proposées pour surpasser cet inconvénient. Cette analyse montre qu'aucune de ces solutions n'apporte de réponse complète, mais l'approche indépendante des langages de spécification en utilisant un modèle de représentation s'avère être une solution attractive pour la conception des systèmes hétérogènes par raffinement à travers plusieurs niveaux d'abstraction. L'avantage le plus important d'un modèle de représentation des systèmes est de pouvoir faire évoluer sa sémantique au fur et à mesure du raffinement.

Dans le Chapitre 2, nous avons proposé un modèle de représentation pour la spécification des systèmes hétérogènes. Ce format intermédiaire couvre les différents niveaux d'abstraction et permet la spécification des systèmes hétérogènes où les différents composants sont décrits en des langages de spécification différents et/ou aux niveaux d'abstraction différents. Le concept central est la séparation entre la communication et le comportement. Pour ce faire, le modèle de représentation présenté propose l'enveloppement de chaque composant hétérogène dans une enveloppe qui sépare la communication et le comportement en abstrayant l'interconnexion d'un

composant du reste du système. Un composant et son enveloppe forment un composant virtuel. Ainsi, les systèmes hétérogènes sont représentés comme un ensemble de composants virtuels interconnectés, dans une architecture virtuelle.

Le Chapitre 3 a présenté une méthodologie pour la validation par cosimulation des systèmes hétérogènes embarqués. L'idée de base est la définition d'un modèle de simulation où chaque composant communique avec le reste du système par une interface de simulation. La définition de la structure interne générique de cette interface permet la génération automatique du modèle de simulation des systèmes hétérogènes. Ainsi, les interfaces sont obtenues par composition des éléments de base situés dans une bibliothèque de cosimulation. Cette méthodologie nous a permis la construction d'un outil de génération automatique des modèles de simulation pour les systèmes hétérogènes. Cet outil reçoit en entrée le modèle de spécification présenté dans le deuxième chapitre de ce manuscrit et génère automatiquement le modèle de simulation correspondant, où l'interface de simulation de chaque composant représente le modèle d'exécution de son enveloppe.

Le Chapitre 4 a abordé la validation du logiciel dans les systèmes embarqués. Dans une première partie de ce chapitre, nous avons étudié les techniques de base utilisées actuellement pour la validation du logiciel, ainsi que les modèles de simulation proposés récemment pour les systèmes d'exploitation. Cette étude démontre que la simulation native du logiciel en utilisant un modèle de simulation pour le système d'exploitation offre des performances en vitesse qui la rend beaucoup plus adaptée à la complexité des systèmes embarqués actuels et aux contraintes de temps de mise sur le marché imposées. Cependant aucun des modèles de simulation proposés n'offre la possibilité d'une validation suffisamment précise du système d'exploitation, ceci étant dû principalement à deux facteurs : (1) le problème d'équivalence entre le code du modèle de simulation et du code de la réalisation finale du système d'exploitation et (2) le manque de précision temporelle afin de pouvoir mesurer les performances de temps du système d'exploitation. Nous avons proposé un nouveau modèle de simulation pour le système d'exploitation. Ce modèle respecte au maximum la réalisation finale du système d'exploitation embarqué, en présentant des modèles de simulation seulement pour une petite partie du code (estimée à 10%) du système d'exploitation et offre la possibilité de mesurer les performances de temps du système d'exploitation. Ce modèle offre un compromis vitesse de simulation – précision très satisfaisant Finalement, nous avons présenté l'adaptation d'une méthodologie déjà existante pour la génération automatique des systèmes d'exploitation afin de pouvoir générer automatiquement le modèle proposé pour la simulation du logiciel.

Le Chapitre 5 a présenté l'application des concepts proposés sur des applications complexes : deux systèmes embarqués multiprocesseurs - le modem VDSL et le système de téléphonie mobile

IS-95 CDMA et un micro-système optique – le commutateur optique. Les objectifs ont été : (1) de prouver l'utilité et l'efficacité du modèle d'architecture virtuelle, (2) d'appliquer la méthodologie de validation sur des applications complexes réelles et (3) de montrer les avantages du modèle de simulation du système d'exploitation. L'architecture virtuelle s'est avérée un repère commun très efficace pour les concepteurs pendant la spécification d'un système hétérogène et en même temps une entrée commune pour les différents outils (génération des modèles de simulation, génération des interfaces logicielles-matérielles). Concernant la méthodologie de validation des systèmes hétérogènes, le temps de génération des modèles de simulation a prouvé un gain important de temps de conception. Ce gain vient s'ajouter à l'augmentation de la fiabilité du modèle de simulation. Finalement, la simulation globale du modem VDSL au niveau RTL en utilisant un modèle de simulation du système d'exploitation a prouvé un gain de vitesse de deux ordres de grandeur et la faisabilité d'intégration d'un tel modèle de simulation dans un flot de conception. La validation de l'application IS-95 à travers les étapes du raffinement incrémental a prouvé la flexibilité, l'extensibilité et la modularité du modèle de simulation proposé.

Perspectives

Notre étude sur la spécification des systèmes hétérogènes qui a établi le cadre conceptuel de ce travail a été focalisée sur la communication. Il est important de poursuive cette étude en tenant compte aussi du comportement des différents composants d'un système. Ainsi, cette étude consistera le fondement d'un travail de recherche sur la définition des différentes classes de modèles de calcul ainsi que sur la taxinomie des langages de spécification selon les différentes classes des modèles de calcul définis.

Concernant la cosimulation, cette technique devient de plus en plus importante dans le domaine de conception des systèmes embarqués. Une première difficulté dans l'achèvement de la cosimulation vient de l'absence d'un langage d'assemblage des composants hétérogènes décrit au différents niveaux d'abstraction et en utilisant différents langages de spécification. Les perspectives à long terme sont donc le développement d'un langage de description d'architectures permettant de donner une syntaxe pour le modèle de représentation des systèmes proposés. Notre expérience dans le domaine de la cosimulation et de la spécification nous a montré que l'évolution de SystemC dans cette direction est une perspective très prometteuse.

Les expérimentations que nous avons effectué pour la validation par cosimulation d'un micro-système optique a marqué le début d'un projet de coopération entre les groupes SLS et MNS du laboratoire TIMA. La cosimulation sera adoptée par la groupe MNS comme technique de validation des MEMS/MOEMS ; elle sera exploitée pour la simulation globale de ces systèmes en intégrant également la simulation des fautes dans le cas de ces systèmes.

Notre analyse sur la simulation rapide à l'aide des modèles de simulation pour le système d'exploitation constitue un premier pas dans cette direction qui s'annonce être très importante pour la conception des systèmes monopuces. Ainsi, ce travail inaugure un axe de recherche important dans le groupe SLS. Plusieurs thèses étudieront la formalisation, la modélisation et la génération automatique des modèles de simulation du logiciel aux différents niveaux d'abstraction ainsi que l'interaction de ces modèles avec la partie matérielle du système.

Annexe A. Conception du bus de cosimulation

Cette annexe présente la conception du bus de cosimulation nécessaire pour la création des instances de cosimulation dans le flot de conception du groupe SLS. La conception du bus de cosimulation implique la mise en œuvre des concepts de communication nécessaires tout au long de ce flot : les concepts de niveau macro-architecture et les concepts de niveau RTL. Concernant les concepts de niveau RTL nous avons réutilisé les concepts fournis par la version 1.0 SystemC. Pour les concepts de niveau macro-architecture, nous avons réalisé nos propres classes. Ces classes sont construites à partir de la bibliothèque maître-esclave de SystemC. Leur réalisation sera présentée par la suite.

Conception des ports de niveau macro-architecture

Un port sera une classe C++ qui peut encapsuler un ou plusieurs ports de la bibliothèque SystemC et qui fournit des fonctions (méthodes) d'accès au canal correspondant.

La Figure 61 illustre la structure d'un port de la bibliothèque d'extension.

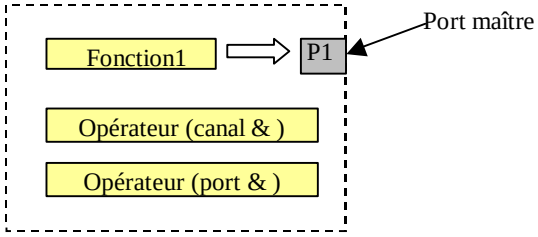


Figure 61. Structure d'un port de la bibliothèque

Dans la figure, `Fonction1` n'est autre que la fonction qui sera appelée par le processus voulant accéder au port (par exemple writeFifo). Cette fonction aura à chaque fois pour rôle d'invoquer le port maître (P1 dans la figure) en lui passant les bons paramètres.

L'opérateur *( )* permet dans sa première version de connecter le port à un canal externe, dans ce cas son argument sera une référence sur ce canal. Dans sa deuxième version, il aura comme argument une référence sur un autre port et permet alors d'associer directement deux ports d'un module hiérarchique.

Remarquons que dans le cas où le port doit fournir plus qu'une fonction d'accès au canal correspondant (ce qui est tout à fait légitime à des niveaux d'abstraction aussi élevés que le niveau macro-architecture) la structure du port aura autant de ports maîtres que de fonctions.

Pour mieux organiser la structure de notre bibliothèque, toutes les classes de ports dérivent d'une seule classe de base appelée Va_port_base. Il s'agit d'une classe virtuelle pure qui ne peut être instanciée directement et qui sert de 'récipient' pour les classes dérivées.

```
/* Classe de base pour tous les ports */
class Va_port_base
{
    /* déclaration des méthodes virtuelles */

      /* Connexion port-canal */
      virtual void operator ()(Va_channel_base)= 0 ;

      /* Connexion port-port */
    virtual void operator ()(Va_channel_port) = 0;

      /* destructeur virtuel */
    virtual void ~Va_port_base(){};
}
```

Dans la déclaration de l'opérateur *( )*, Va_channel_base désigne la classe de base des canaux. Cette classe sera définie dans le paragraphe suivant.

Conception des canaux de niveau macro-architecture

D'une façon analogue aux ports, les canaux seront aussi représentés par une classe C++ qui encapsule un certain nombre d'éléments de base, qui seront, pour les mêmes raisons que précédemment, choisis dans la bibliothèque de communication maître-esclave.

Les canaux doivent fournir un moyen pour implémenter les fonctions appelées à partir des ports. Pour cela, un canal dispose normalement d'une partie 'comportement' qui n'est autre que la définition de ces fonctions. La Figure 62 illustre la structure d'un canal de la bibliothèque d'extension

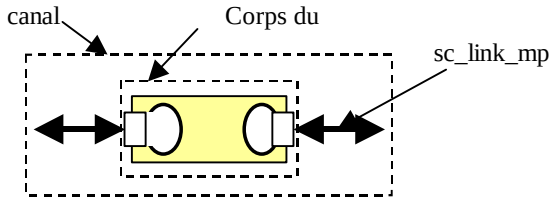


Figure 62. Structure d'un canal de la bibliothèque

La figure montre qu'un canal est une structure plus ou moins complexe qui renferme des signaux de la bibliothèque maître-esclave et une sous structure appelée 'Corps du canal' (channel core).

Les signaux de type 'sc_link_mp' sont utilisés pour établir la liaison avec les ports maîtres

contenus dans les ports de la bibliothèque (paragraphe précèdent) d'une part et le corps du canal de l'autre.

Le 'corps du canal' est l'élément responsable du comportement du canal. Sa structure est détaillée dans la Figure 63.

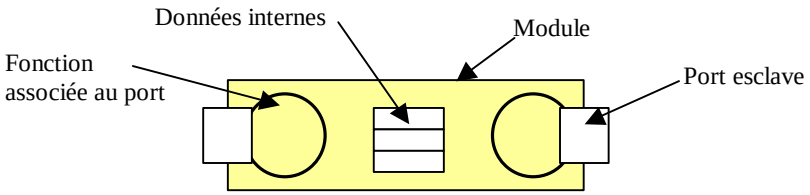


Figure 63. Structure d'un corps de canal

L'élément de base dans la structure d'un corps de canal est un module contenant un ou plusieurs ports esclaves. Le nombre de ces ports dépend du nombre des signaux du type 'sc_link_mp' dans le canal (un port pour chaque signal).

L'activation d'un port esclave provoque l'exécution immédiate de la fonction qui lui est associée. Ce sont ces fonctions qui définissent le comportement du canal.

La Figure 63 montre aussi la présence d'une structure de données internes spécifique à chaque type de canal. Ces données peuvent être de différentes natures et entrent dans la spécification du comportement du canal.

De la même façon que pour les ports, tous les types de canaux dérivent d'une seule classe de base appelée Va_channel_base. Il s'agit d'une classe virtuelle pure qui ne peut être instanciée directement et qui sert de 'récipient' pour les classes dérivées.

```
/* Classe de base pour tous les canaux */
class Va_channel_base
{

      /* destructeur virtuel */
    virtual void ~Va_port_base(){};
}
```

Fonctionnement de l'ensemble

Les ports et les canaux de la bibliothèque d'extension ont été conçus de telle sorte qu'ils « collent » parfaitement et d'une manière assez naturelle.

La Figure 64 illustre bien cette correspondance.

Remarquons qu'en règle générale, les sous ports internes d'un port de la bibliothèque d'extension sont de type maître alors que les sous ports contenus dans un canal de la bibliothèque sont toujours de type esclave.

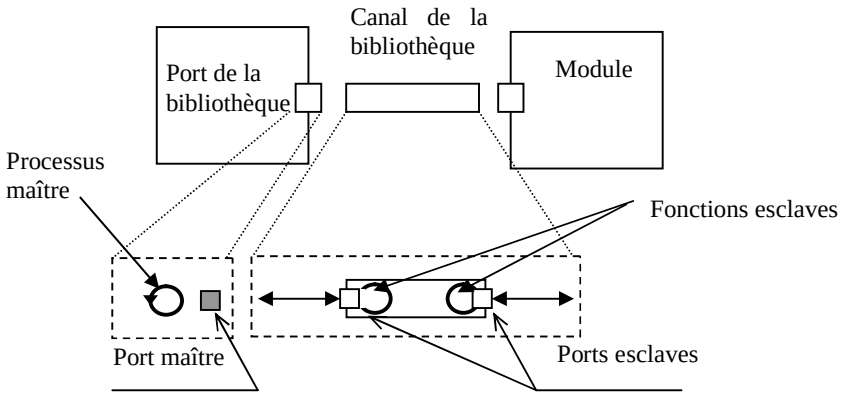


Figure 64. Correspondance canal - port

Bibliographie

[Abr 97] J.R. ABRIAL, "*The B book. Assigning Programs to Meaning"*, Cambridge University Press, 1997.

[Alb 93] T . W ALBERCHT, J. NOTBAUER, and S. ROHRINGER, "*Hw/Sw CoVerification Performance Estimation & Benchmark for a 224 Embedded RISC Core Design*", Proc. Design Automation Conf.(DAC), p. 801-811, June 1998.

[All 97] R. ALLEN and D. GARLAN, "*A Formal Basis for Architectural Connection*", ACM Transactions on Software Engineering and Methodology, 6(3), July, 1997.

[And 91] G.R ANDREWUS, "*Concurrent Programming, Principles and Practice*", Benjamin /Cummings (eds), Redwood City, Calif., p. 484-494, 1991.

[Bag 01] A. BAGHDADI, D. LYONNARD, N-E ZERGAINOH, A-A. JERRAYA, "*An Efficient Architecture Model for Systematic Design of Application-Specific Multi-processor SoC*', Design Automation and Test in Europe *(DATE)*, March, 2001.

[Ber 84] 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.

[Boo 91] G. BOOCH, "*Object-Oriented Design With Applications"*, Benjamin/Cummings, Menlo Park, California, 1991.

[Bru 00] J-Y. BRUNEL, W.M. KRUIJTZER, H.J.H.N. KENTER, F. PÉTROT, L. PASQUIER, , E.A. de KOCK, W.J.M SMITS, "*COSY Communication IP's*", proc. Design Automation Conference, Los Angeles, CA, June 2000

[Buc 00] J. BUCK at R. VAIDYANATHAM, "*Heterogeneous Modeling and Simulation of Embedded Systems in El Greco*", CODES, San Diego, CA, 2000

[Cad 02] Cadance Design Systems, Inc., Virtual Component Design, available on line at http://www.cadence.com/products/vcc.html

[Cal 97] J.P. CALVEZ, D. Heller, O. PASQUIER, "*Hardware/Software System Design Based on the MCSE Methodology*", Current Issues in Electronic Modeling, Volume 9: System Design, Editors J.M. Bergé, O. Levia, J. Rouillard, Kluwer Academic Publishers, 1997 – Ch 6 – p. 115-150.

[Cal 96] J.P. CALVEZ, D. Heller, O. PASQUIER, "*Uninterpreted Co-Simulation for Performance Evaluation of Hw/Sw Systems*", Proc. CODES/CASHE'96, Pittsburgh, Pennsylvania, USA, 18-20 mars, 1996 p. 132-139.

[Cal 95] J.P. CALVEZ, D. Heller, O. PASQUIER, "*System Performance Modeling and Analysis with VHDL: Benefits and Limitations*", Proc. VHDL Forum for CAD in Europe Conference, 1995.

[Car 02] « Carbonkernel », available at http://www.carbonkernel.org/

[Cle 00] C-level Design Home Page, available on-line at, http://www.cleveldesign.com**,** 2000.

[Ces 01] W.O. CESARIO, G. NICOLESCU, L. GAUTHIER, D. LYONNARD, and A.A. JERRAYA, "*Colif: a Multilevel Design Representation for Application-Specific Multiprocessor System-on-Chip Design*", 12th IEEE International *Workshop on Rapid System Prototyping*, June 25-27, 2001, California, USA.

[Cho 95] P.H. CHOU, R.B. Ortega, G. BORIELLO, "*The Chinook Hardware/Software Co-Synthesis System*", Proc. of the International Symposium on System Synthesis, 1995

[Cos 99] P. COSTE, F. HESSEL, Ph. LEMARREC, *et al.* "*Multilanguage Design of Heterogeneous systems*", Proc. Int. Workshop on Hardware-Software Codesign, May 1999.

[Cow 02] COWARE. Inc., "N2C" available at http://www.coware.com/cowareN2C.html.

[Cyr 01] G. CYR, G. BOIS, and M. ABOULHAMID, "*Synthesis Of Communication Interface For Soc Using VSIA Recommendations*," Proc. DATE Design Forum, Munich, Germany, 13-16 March, 2001, pp. 155-159

[Dav 97] J.M. DAVEAU, G. MARCHIORO, T. BEN ISMAIL, A.A. JERRAYA, "*Protocol Selection and Interface Generation for Hw-Sw Codesign*", IEEE Trans. on VLSI Systems, vol. 5, nº. 1, p. 136-144, 1997.

[Del 95] C. DELGADO KLOOS and MARIN LOPEZ et all., "*From Lotos to Vhdl*", Current Issues in Electronic Modeling, 3, September, 1995.

[Des 00] D. DESMET, D. VERKEST, and H. de MAN, "*Operating System Based Software Generation for Systems-on-chip*", Proc. Design Automation Conf. (DAC), June 2000

[Eke 01] J. EKER, C. FONG, Jorn W. JANNECK, and J. LIU, "*Design and Simulation of Heterogeneous Control Systems using Ptolemy II*," Proc. IFAC Conference on New Technologies for Computer Control (NTCC'01), Hong Kong, China, November 2001.

[Ern 99] R. ERNST, et al., "*Hardware-Software Co-Design of Embedded Systems – The SPI Workbench*", Proc. IEEE Workshop on VLSI, pp. 9-17, Orlando, 1999.

[Edw 97] S. EDWARDS, L. LAVAGNO, E. A. LEE, and A. SANGIOVANNI-VINCENTELLI, "*Design of Embedded Systems: Formal Models, Validation, and Synthesis*", Proceedings of the IEEE, Vol. 85, No. 3, March 1997.

[Fer 99] A. FERRARI et A. SNAGIVANNI-VICENTELLI, "*System Design : Traditional Concepts and New Paradigms*", proc. of ICCD, Austin, TX, USA, October 1999.

[Fil 02] L. FILION, J. CHEVALIER, G. BOIS, E. M. ABOULHAMID, "*The Syslib-Picasso Methodology for the Co-Design Specification Capture Phase*", International Workshop on SoC for Real-Time Application, Banff, AB, Canada, July 2002

[Fil 02a] L. FILION, G. BOIS, E. M. ABOULHAMID. "*SYSLIB: A System-Level Library Extended from Cynlib for So*C", The International HDL Conference and Exhibition. San Jose, CA, USA, 03/2002

[Fil 02b] FILION, L., BOIS G., ABOULHAMID, M. "*Picasso: a complete methodology for designing embedded systems*", ACFAS: 70th French Canadian Convention for Sciences. Québec City, QC, Canada. May 2002

[For 01] FORTE DESIGN SYSTEMS, Cynlib Users Manual Release, available on-line at http://www.forteds.com/products/index.html

[Gaj 00] D. D. GAJSKI, J. ZHU, R. ZÖMER, A. GERSTLAUER, S. ZHAO, "*SpecC Specification Language and Methodology",* Kluwer Academic Publishers, Boston, MA, ISBN 0-7923-7822-9, March 2000.

[Gau 01] L. GAUTHIER, S. YOO and A.A. JERRAYA, "*Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software*", Proc. Design Automation and Test in Europe, mars, 2001.

[Gaut 01] L. GAUTHIER, "*Génération de système d'exploitation pour le ciblage de logiciel multitâche sur des architectures multiprocesseurs hétérogènes dans le cadre des systèmes embarqués spécifiques*", Thèse de doctorat, INPG, laboratoire TIMA,

décembre, 2001.

[Gho 95] A. GHOSH, M. BERSHTEYN, et al. "*A Hardware-Software Co-Simulator for Embedded Systems Design and Debugging*", Proc. Asia South Pacific Design Automation Conference, 1995.

[Goe 99] R. GOERING, "*Foundation Laid for Next-Generation Languages*", Electronic Engineering Time, December 3, 1999

[Har 87] D. HAREL, "*Statecharts : A Visual Formalism for Complex Systems*", Science of Computer Programming, 1987, 8, p. 231-274.

[Hal 92] N. HALBWATCHS, F. LAGNIER, and C. RATEL. "*Programming and verifying real-time systems by means of the synchronous data-flow luster*", IEEE Transactions on Software Engineering, 18(9), September 1992.

[Hen 99] Y. HÉNEAULT, G. BOIS, E. M. ABOULHAMID, J. BAILLARGÉ, and P. YOUSEFPOUR, "*Picasso : A H/S Capture Tool Based on VSIA Recommendations*", Proc. International Workshop on IP-Based Synthesis and System Design, Grenoble, France, December 14-15, 1999

[Hen 01] Y. HÉNEAULT, G. BOIS, and E. M. ABOULHAMID, "*A Fast Hardware Co-Specification and Co-Simulation Methodology Integrated in a H/S Co-Design Platform*", Proc. The 13th International Conference on Microelectronics, Rabat, Morocco, Oct. 29-31, 2001, pp. 249-252

[Hoa 85] C.A.R. HOARE, *Communicating Sequential Processes*, 1985, Prentice Hall.

[Iee 93] Institute of Electrical and Electronically Engineers, *IEEE Standard VHDL Language Reference Manual*, 1993, STD 1076-1993. IEEE.

[Itr 01] ITRS, International Technology Roadmap for Semiconductors, Design, Edition 2001.

[Jac 94] I. JACOBSON, "*Object-Oriented Software Engineering : A Use Case Driven Approach"* (Addison-Wesley Object Technology Series), Mars,1994, Hardcover.

[Jag 96] D. JAGGAR, "*Advanced RISC Machines Architectural Reference Manual*", Prentice Hall, 1996.

[Jan 00] A. JANTSCH, S. KUMAR, A. HEMANI, "*The Rugby Model: A Metamodel for Studying Concepts in Electronic System Design*", *IEEE Design & Test of Computers*, 2000, p. 78-85.

[Jer 97] A.A. JERRAYA, M. ROMDHANI et al., "*Languages for System-Level Specification and Design"*, chapitre dans Hardware/software Co-Design, Principles and Practice, 1997, Kluwer Academic Publishers.

[Kea 99] M. KEATING, P. BRICAUD, "*Reuse Methodology Manual*", Kluwer Academic Publisher, 1999.

[Keu 99] K. KEUTZER *et al.* "*The MACRO/DARPA Gigascale Silicon Research Center*", Proc. ICCD, Austin, TX, USA, October 1999.

[Kur 01] T. KURZWEG, J. MARTINEZ, S. LEVITAN, P. MARCHAND, D. CHAIRULLI, "*Dynamic Simulation of Opitcal MEM Switches*", Proc. DTIP France, April, 2001.

[Laj 99] M. LAJOLO, M. LAZARESCU, and A. SANGIVANNI-VICENTELLI, "*A Compilation-based Software Estimation Scheme for Hardware/Software Co-Simulation*", Proc. Int'l Workshop on Hardware Software Codesign, May 1999.

[Lee 99] E.A. LEE *et al*, "*Ptolemy –II: Heterogeneous Concurrent Modeling and Design in Java*" Tech. Rep. UCB/ERL M99/40, July 19, 1999

[Lee 00] E.A. LEE and S. NEUENDORFFER, "*MoML – A Modeling Markup Language in XML – Version 0.4*", Memorandum M00/12, University of California, at Berkeley, March 14, 2000.

[Lee 97] A. LEE and A. SANGIOVANNI-VICENTELLI, "*A Denotational Framework for Comparing Models of Computation*", ERL Memorandum UCB/ERL-M97/11, University of California, Berkley, CA 94720, January 1997.

[Lem 00] Ph. LEMARREC, "*Cosimulation multi-niveaux dans un flot de conception multi-langage*", Thèse de doctorat, INPG, Tima Labratory, juin 2000.

[Len 00] C. K. LENNARD, "*System-level models explore terrain*", EE Times, June 13, 2000

[Luk 95] D. LUCKMAN et al., "*Specification and Analysis of system architecture using RAPIDE", IEEE on software engineering – special issue on software architecture*, 21(4), April, 1995, p. 336-355.

[Lyo 01] D. LYONNARD, S. YOO, A. BAGHDADI, A. A. JERRAYA, "*Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip*", Proceedings DAC 2001, June 2001, Las Vegas, USA

[Med 96] N. MEDVIDOVIC, "*A Clasification and Comparison Framework for Software Architecture Description Languages*", Technical Report UCI-ICS-97-02, University

of California, February 1996.

[Med97] N. MEDVIDOVIC and D. S. ROSENBLUM. « Domains of Concern in Software Architectures and Architecture Description Languages », U*SENIX Conference on Domain-Specific Languages*, Santa Barbara, CA, October 1997.

[Moo 98] PHILIP R. MOORBY, DONALD E. THOMAS, "*The Verilog Hardware Description Language"*, May 1998, Hardcover.

[Mat 00] MATHWORKS. 2 000, Matlab, http://www.mathworks.com.

[Men 02] Mentor Graphics, Inc, "Seamless CVE", available at http://www.metorg.com/semless

[Mar 96] P. MARIATOS, A.N. BIRBAS et al., "*Integrated System Design with an Object-Oriented Methodology*", vol.7 . Object-Oriented Modeling, J.M. Berge, Oz Levia, J. Rourillard Editors, CIEM : Current Issues in Electronic Modeling, September, 1996, Kluwer Academic Publishers.

[Mes 00] D.J.G. MESTDAGH, M.R. ISAKSSON, P. ODLING, "*Zipper VDSL: A Solution for Robust Duplex Communication over Telephone Lines*", IEEE Communication Magazine, pp. 90-96, May 2000.

[Max 00] MATRIXx, Integrated Systems Inc., 1998, http://www.Isi.com/Products/Matrixx.

[Mod 01] MODELSIM, available on line to http://www.model.com

[Obj 00] OBJECTIME, available on-line at http://www.objectime.on.ca/, 2000.

[Omg 97] OBJECT MANAGEMENT GROUP, CORBA Services; Common Object Services Specification, Technical Rapport, OMG, July, 1997.

[Pas 99] O. PASQUIER, J.P. CALVEZ, "*An Object-Based Executable Model for Simulation of Real-Time Hw/Sw Systems*", in Proceedings of the Design Automation and Test in Europe (DATE), 1999.

[Par 95] Thomas M. PARKS, José Luis PINO and Edward A. LEE, "*A Comparison of Synchronous and Cyclo-Static Dataflow*", Proceedings of the Asilomar Conference on Signals, Systems, and Computers Pacific Grove, CA, October 29-November 1, 1995

[Pto 02] *PTOLEMY project, available on-line at http://ptolemy.eecs.berkeley.edu, 2002.*

[Row 94] *J. ROWSON, "Hardware/Software Cosimulation co-simulation",* Proc Design Automation Conference (DAC), 1994.

[Sem 00] L. SEMERIA et A. GHOSH, "*Methodology for Hardware/Software Co-verification in C/C++*", Proc. Asia South Pacific Design Automation Conference (ASPDAC), Jan. 2000.

[Sdl 87] Computer Networks and ISDN Systems. CCITT SDL, 1987.

[Sgr 00] M. SGROI, L. LAVAGNO and A.S. VICENTELLI, "*Formal Models for Embedded System Design*", IEEE Design & Test of Computers, vol. 17 no. 12, April-June 2000.

[Sld 02] SLDL Committee Homepage, available on-line at http://www.inmet.com/SLDL, 2002.

[Sha 95] M. SHAW and al., "*Abstractions for Software Architecture and Tools that Support Them",* IEEE Transactions on Software Engineering, 21(4), 1995.

[Spi 89] J.M. SPIVEY, "*An Introduction to z Formal Specifications*", Software Engineering Journal, January 1989, p. 40-50.

[Spw 02] SPW, available on line to www.cadence.com/products/spw.html, 2002

[Sup02] CO-DESIGN AUTOMATION, Inc 1998-2002, available at http://www.superlog.org/

[Syn 02] SYNOPSYS Eaglei, available on line at http://www.synopsys.com/products/hwsw/eagle_ds.html**.**

[Syn 97] SYNOPSIS. 1997 (Jan.) COSSAP (Reference Manuals), Synopsys.

[Sys 00] SYNOPSYS, Inc., "*SystemC, Version 2.0*", available at http://www.systemc.org/.

[Suz 96] K. SUZUKI and A. SANGIVANNI-VICENTELLI, "*Efficient Software Performance Estimation Methods for Hardware/Software Design*", Proc. Design Automation Conference, June 1996.

[TIA 95] TIA/EIA-95A, "*Mobile Station-Base Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular Systems*", 1995

[Uml 02] UML, available on-line at http://www.rational.com/uml/, 2002.

[Val 94] C. VALDERRAMA, F. NACABAL, P. PAULIN et A.A. JERRAYA, "*Automatic VHDL-C Interface Generation of Distributed Cosimulation : Application to Large Design Examples*", Design Automation for Embedded Systems vol. 3, no. 2/3, p. 199-217, Apr. 1994.

[Ver 96] D. VERKEST, K. VAN ROMPAEY, I. BOLSENS, H. DE MAN, "*CoWare – A Design environment for heterogeneous hardware/software systems*", Design Automation for Embedded Systems, vol. 1, nº. 4, p. 357-386, 1996.

[Win 02] WINDRIVER Systems, Inc., "*VxWorks 5.4*", available at http://www.wrs.com/products/html/vxwks5.4.html/.

[Wu 97] WU, M.C., "*Micro machining for Optical and Optoelelctronic Systems*", Proc. of the IEEE, Vol. 85 No. 11, Nov. 1997

[XML00] XML 1.0 Specification, 2d ed., W3C Recommendation, Oct. 2000, http://www.w3c.org/XML

[Yoo 99] S.YOO, J. LEE, J.JUNG, K. RHE, Y.CHO et K.CHOI, "*Fast Prototyping of an IS-95 CDMA Cellular Phone : a Case Study*", Proc. 6th Conference of Asia Pacific Chip Design Languages, Oct. 1999

[Zhu 99] J. ZHU, D. D. GAJSKI, "*A Retargetable, Ultra-fast Instruction Set Simulator*", DATE Conference 99, Munich, Germany

[Ziv 96] V. ZIVOJNOVIC, H. MEYR, "*Compiled Hw/Sw Cosimulation*", Proc. Design Automation Conference (DAC), 1996.

Glossaire

A

ADL **Architecture Description Languages**
Langages permettant la description formelle de l'architecture de haut niveau d'un système

API **Application Program Interface**
Ensemble de routines standards destinées à faciliter au programmeur le développement d'applications

ASIC **Application Specific Integrated Circuit**
Circuit intégré développé Spécifiquement pour une Application

B

BCA **Bus Cycle Accurate**
Niveau d'abstraction pour la conception utilisé dans l'environnement CoWare N2C

BCASH **Bus Cycle Accurate Shell**
Niveau d'abstraction pour la conception utilisé dans l'environnement CoWare N2C

BFM **Bus Functional Model**
Interface pour la simulation, permettant de transformer les accès mémoire fonctionnels en des accès mémoire cycle-près

C

CA **Channel Adapter**
Composant élémentaire d'une interface de cosimulation

CORBA **Common Object Request Broker Architecture**
Architecture commune de courtier d'objets de requête : une norme établie par le Object Management Group (OMG) (groupe de gestion des objets), destinée à la communication entre objets répartis.

CDMA **Code Division Multiple Access**
Technique de codage d'information utilisée dans la téléphonie mobile

CLI **C Langage Interface**
Bibliothèques fournies par le simulateur de matériel VSS afin de permettre la simulation des systèmes incluant de composants décrites dans le langage C

CPU **Central Processor Unit**
Partie principale d'un système, réservée aux traitements

CSP **Communicating Sequential Processes**
Langage formel utilisé pour la description des systèmes parallèles

D

DSP **Digital Signal Processor**
Processeur spécialisé pour le calcul d'algorithmes de traitement de signal

F

FSM **Finite State Machine**
Modèle de calcul définit par en un ensemble d'états, un ensemble d'événements d'entrée, un ensemble d'événements de sortie et une fonction de transition entre les états

FIFO **First In First Out**
Classe ou protocole de communication qui assure que les premières données envoyées sont les premières reçues

FPGA **Field Programmable Gate Array**
Réseau prédiffusé programmable par l'utilisateur : Réseau prédiffusé de portes logiques qui est destiné à être programmé sur place, avant d'être utilisé pour une fonction particulière

H

HAL **Hardware Abstraction Layer**
La couche basse de l'organisation du logiciel fournissant les pilotes et les contrôleurs pour la gestion de la communication

HDL **Hardware Description Language**
Catégorie des langages utilisée pour la spécification des systèmes matériels

I

ICM **Internal Communication Média**
Média de communication assurant la communication entre l'adaptateur de module et les adaptateurs de canal composant une interface de cosimulation

IP **Intellectual Property**
Élément (logiciel ou matériel) dont le fonctionnement est connu et documenté, mais dont la structure interne est inconnue

IPC **Inter-Process Communication**
Echange de données entre des processus s'exécutant sur une même machine où sur des machines différentes

ISA **Instruction Set Architecture**
Niveau d'abstraction pour le logiciel

ISS **Instruction Set Simulator**
Outil qui s'exécute sur la machine hôte et qui émule la fonctionnalité d'un processeur

M

MA **Module Adapter**
Composant élémentaire dans la structure de l'interface de cosimulation. Ce composant est spécifique au module à intégrer dans la cosimulation

MCU **Micro-Controller Unit**
Puce constituée d'éléments intégrés lui permettant de fonctionner comme une unité autonome, et conçue spécifiquement pour contrôler un dispositif donné

MOEMS **Micro-Opto-Electro-Mechanical Systems**
Micro-systèmes optiques intégrant sur une seule puce des sous-systèmes électroniques, mécaniques et optiques

MoML **Modeling Markup Language**
Sous-ensemble de XML spécialisé pour la spécification des interconnexions d'un système

N

N2C **Napkin-to-Chip**
Extension de C utilisée dans l'environnement de conception CoWare pour la description des systèmes

O

OOA/OOD **Object Oriented Analysis/Object Oriented Design**
Méthodes orientés objet pour l'analyse et la conception des systèmes complexes

OMT **Object Modelling Technique**
Méthode d'analyse et de conception orienté objet

OVI-SRM **Open Verilog International Semantic Reference Manual**
Manuel référentiel pour la sémantique qui décrit d'une façon indépendante des langages le mécanisme d'ordonnancement des différentes tâches dans un système embarqué multiprocesseur

P

PA **Port Adapter**
Composant élémentaire d'une interface de cosimulation

R

RPC **Remote Procedural Call**
Appel de procédure à distance : appel de procédure permettent à un programme local d'utiliser des procédures situées dans un programme distant.

RTL **Register Transfer Level**
Niveau d'abstraction pour la spécification des systèmes.

S

SDL **Specification and Design Language**
Langage de haut niveau pour la spécification non-ambiguë des systèmes comportant des processus parallèles

SoC **System on Chip**
Système monopuce – circuit intégrant sur une même puce différents composants fonctionnels (ex. mémoires, processeurs)

SPW **Signal Processing WorkSystems**
Environnement pour la conception des systèmes orientés traitement signal

SLS **System Level Synthesis**
Groupe de recherché du laboratoire TIMA/INP Grenoble

U

UML **Unified Modelling Language**
Langage de modélisation, permettant de déterminer et de présenter les composants d'un système lors de son développement, ainsi que d'en générer la documentation

UT **UnTimed**
Niveau d'abstraction pour la conception utilisé dans l'environnement CoWare N2C

V

VDSL **Very high data rate Digital Subscriber Line**
Standard de communication par paires torsadées

VHDL **Very high-scale integrated Hardware Description Language**
Langage de description de système électronique numérique, basé sur une extension de Ada

VSIA **The Virtual Socket Interface Alliance**
Consortium d'industriels qui travaille sur la définition d'un standard des interfaces des composants d'un système

VSS **VHDL System Simulator**
Simulateur de VHDL fourni par Synopsys

X

XML **Extensible Markup Language**
Métalangage standardisé par W3C

Publications

1. G. Nicolescu, S. Yoo, A. Bouchihma, A.A. Jerraya, “Validation in a Component-Based Design Flow for Multi-core SoCs”, to appear ISSS’02, October 2-4, Kyoto, Japan.

2. W. Cesario, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. PAVIOT, S. YOO, A.A. Jerraya, M. Diaz-Nava, "*Component-Based Design Approach for Multicore SoCs*", DAC'02, June 10-14 2002, New Orleans, USA, 2002.

3. L. Kriaa, W. Youssef, G. Nicolescu, S. Martinez, S. Levitan, J. Martinez, T. Kurzweg, A.A. Jerraya, B. Courtois, "*SystemC-Based Cosimulation for Global Validation of MOEMS*", DTIP2002, 5-8 May 2002, Cannes-Mandelieu, France.

4. S. Yoo, G. Nicolescu, L. Gauthier, A.A. Jerraya, "*Automatic Generation Including Fast Timed Simulation Models of Operating Systems in Multiprocessor SoC Communication Design*", DATE 2002, Paris, France, March 2002.

5. W. Cesario, Y. Paviot, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, S. Yoo, A.A. Jerraya, M. Diaz-Nava, "*HW/SW Interfaces Design of a VDSL Modem using Automatic Refinement of a Virtual Architecture Specification into a Multiprocessor SoC: a Case Study* ", DATE 2002, Paris, France, March 2002.

6. G. Nicolescu, K. Svarstrad, W. Cesario, L. Gauthier, D. Lyonnard, S. Yoo, P. Coste, A.A. Jerraya, "*Desiderata pour la spécification et la conception des systèmes électroniques*", Technique et Science Informatiques, March 2002.

7. G. Nicolescu, S. Martinez, L. Kriaa, W. Youssef, S. Yoo, B. Charlot, A.A. Jerraya, "*Application of Multi-domain and Multi-language Cosimulation to an Optical MEM Switch Design”*, ASP-DAC 2002, Bangalore, India, January 2002.

8. P. Gerin, S. Yoo, G. Nicolescu, A. A. Jerraya, "*Scalable and Flexible Cosimulation of SoC Designs with Heterogeneous Multi-Processor Target Architectures*", Proceedings ASP-DAC 2001, January 2001, Yokohama, Japan.

9. K. Svarstrad, N. BEN Fredj, G. Nicolescu, A. A. Jerraya, "*A Higher Level System Communication for Object Oriented Specification and Design of Embedded Systems*", Proceedings ASP-DAC 2001, January 2001, Yokohama, Japan.

10. G. Nicolescu, S. Yoo, A. A. Jerraya, "*Mixed-Level Cosimulation for Fine Gradual Refinement of Communication in SoC Design*", Proceedings DATE 2001, March 2001, Munich, Germany.

11. K. Svarstrad, G. Nicolescu, A. A. Jerraya, "*A Model for Describing Communication between Aggregate Objects in the Specification and Design of Embedded Systems*", Proceedings DATE 2001, March 2001, Munich, Germany.

12. S. Yoo, G. Nicolescu, D. Lyonnard, A. Baghdadi, A. A. Jerraya, "*A Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design*", Proceedings CODES 2001, Copenhague, Danemark, April 2001.

13. W. Cesario, G. Nicolescu, L. Gauthier, D. Lyonnard and A.A. Jerraya, "*Colif: a Multilevel Design Representation for Application-Specific Multiprocessor System-on-Chip Design*", 12th IEEE International Workshop on Rapid System Prototyping, California, USA, June 2001.

14. W. Cesario, G. Nicolescu, L. Gauthier, D. Lyonnard, A. A. Jerraya, "*Colif: A Design Representation for Application-Specific Multiprocessor SOCs*", IEEE Design & Test of Computers, Vol. 18 n° 5, Sept/Oct 2001.

15. Z. Juneidi, K. Torki, S. Martinez, G. Nicolescu, B. Courtois, A. Jerraya, "*Global modelling and simulation of System-on-Chip embedding MEMS devices*", 4th International Conference on ASIC, ASICON 2001, Beijing, China, October 23-25, 2001

16. A. A. Jerraya, A. Baghdadi, W. Cesario, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, Invited Paper "*Application-Specific Multiprocessor Systems-on-Chip*", SASIMI 2001, The Tenth Workshop on Synthesis And System Integration of Mixed Technologies, Nara, Japan, October 18-19 2001.

17. S. Yoo, G. Nicolescu, L. Gauthier, A.A. Jerraya, "*Fast Timed Cosimulation of HW/SW Implementation of Embedded Multiprocessor SoC Communication"*, HLDVT '01, Monterey, USA, Oct. 2001.

18. 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.

19. S. Mir, B. Cahrlot, 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.

20. F. Hessel, P. Coste, G. Nicolescu, Ph. Le Marrec, N-E. Zergainoh, A.A. Jerraya, "*Multi-level Communication Synthesis of Heterogeneous Multilanguage Specifications*", Proc. of ICCD, Texas, USA, September 2000.

21. 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.3/4, pp. 223-236, Kluwer Academic Publishers, August 2000.

22. W.O. Cesario, L. Gauthier, D. Lyonnard, G. Nicolescu, A.A. Jerraya, "*An XML-based Meta-model for the Design of Multiprocessor Embedded Systems*", VHDL International User's Forum (VIUF) Fall Workshop, Orlando, FL, October 2000.

RESUME

La tendance très récente dans la conception des systèmes embarqués est l'assemblage des composants standards existants. La difficulté d'une telle conception provient du fait que ces composants sont hétérogènes. Ainsi, dans un système embarqué plusieurs processeurs, DSP, bus, etc. utilisant de protocoles de communication différents doivent communiquer. Dans ce cas nous parlons d'une hétérogénéité des composants en terme de protocoles de communication. Chaque composant peut être décrit dans un langage de spécification approprié ce qui implique l'hétérogénéité des langages de spécification. Il se peut aussi que les différents composants se trouvent dans des stades différents de raffinement, dans ce cas les composants sont hétérogènes en termes de niveaux d'abstraction.

Dans ce travail nous avons abordé deux principaux problèmes liés à la conception des systèmes embarqués hétérogènes, la spécification et la validation. Nos contributions sont : (1) une étude des langages de spécification et des niveaux d'abstraction, étude qui a aidé à la définition d'un modèle de représentation pour la spécification en vue de la conception des systèmes hétérogènes embarqués, (2) la proposition d'une méthodologie de validation par simulation des systèmes hétérogènes embarqués et (3) la proposition d'un modèle de simulation pour la validation rapide des interfaces logicielles (les systèmes d'exploitation) dans les systèmes embarqués.

Les concepts proposés ont été validés sur des applications complexes : deux systèmes embarqués multi-processeurs - le modem VDSL et le système de téléphonie mobile IS-95 CDMA et un micro-système optique – le commutateur optique.

TITRE EN ANGLAIS

SPECIFICATION AND VALIDATION FOR HETEROGENEOUS EMBEDDED SYSTEMS

ABSTRACT

Currently the design of heterogeneous embedded systems requires the integration of existing standard components. The difficulty of such design resides in the heterogeneity of components. Thus, such a system may be composed of several DSP, buses, etc. using different communication protocols. This is the case of heterogeneity in terms of communication protocols. Each component may be described using the appropriate specification language. This implies heterogeneity in terms of specification languages. The different components may be specific to different design stages. In this case, the components are heterogeneous in terms of abstraction levels.

This work approaches two main aspects related to the design of heterogeneous embedded systems: the specification and the validation. Our contributions are: (1) the study on the specification languages and abstraction levels for heterogeneous system design, this study was the support for the definition of a representation model for heterogeneous systems design; (2) the definition of a methodology for simulation based validation of heterogeneous embedded systems and (3) a native simulation model for the fast validation of operating systems in embedded systems.

The proposed concepts where validated using complex applications : two multi-processor embedded systems - the VDSL modem and the IS-95 CDMA mobile phone system and an optical micro-system - the optical switcher.

SPECIALITE Microélectronique

MOTS CLES

Systèmes hétérogènes, langages de spécification, validation, cosimulation, systèmes d'exploitation, architectures logicielles/matérielles

INTITULE ET ADRESSE DU LABORATOIRE

Laboratoire **TIMA**, Techniques de l'Informatique et de la Micro-électronique pour l'Architecture des ordinateurs, 46 Avenue Félix Viallet, 38031 Grenoble

ISBN 2-913329-91-8 Broché
ISBN 2-913-329-91-6 Format Electronique