Modélisation et validation des systèmes hétérogènes : définition d'un modèle d'exécution

L. Kriaa

▶ **To cite this version:**

L. Kriaa. Modélisation et validation des systèmes hétérogènes : définition d'un modèle d'exécution. Autre [cs.OH]. Université Joseph-Fourier - Grenoble I, 2005. Français. NNT : . tel-00011219

HAL Id: tel-00011219

https://theses.hal.science/tel-00011219

Submitted on 16 Dec 2005

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.

Université Joseph Fourier, Informatique et Mathématiques Appliquées

N° attribué par la bibliothèque
|__/__/__/__/__/__/__/__/__/__|

T H E S E

pour obtenir le grade de

DOCTEUR DE l'UNIVERSITE JOSEPH FOURIER

Spécialité : Informatique

préparée au laboratoire **TIMA** dans le cadre de
L'Ecole Doctorale de« Mathématique, Sciences et Technologies de l'Information (informatique)»

présentée et soutenue publiquement

par

Lobna KRIAA

Le 10 Novembre 2005

Titre :

Modélisation et Validation des systèmes hétérogènes : Définition d'un modèle d'exécution

---

Directeur de Thèse : Ahmed Amine Jerraya

JURY

Mme. Dominique Borrione	, Présidente
Mr. Alain Vachoux	, Rapporteur
Mr. Jean Luc Dekeyser	, Rapporteur
Mr. Ahmed Amine Jerraya	, Directeur
Mme. Gabriela Nicolescu	, Examinatrice
Mr. Philippe Kajfasz	, Examinateur

Remerciements

Au terme de ce travail, je tiens à exprimer mes remerciements envers toutes les personnes qui ont contribué, de près ou de loin, à l'accomplissement de cette thèse. Je remercie :

M. Ahmed Amine Jerraya, mon directeur de thèse qui m'a offert l'opportunité d'intégrer son équipe et qui a su avec sa patience et sa sagesse me guider tout au long de ce parcours ;

Mes deux rapporteurs M. Alain Vachoux, professeur à l'université de l'Ecole Polytechnique Fédérale de Lausanne et M. Jean Luc Dekeyser professeur à l'Université des Sciences et Technologies de Lille 1, pour leurs analyses pertinentes;

Mme. Gabriela Nicolescu, professeur adjoint à l'école polytechnique de Montréal, pour sa disponibilité et ses précieux conseils théoriques et techniques qui m'ont été d'un grand apport dans mon travail ;

M. Philippe Kajfasz, directeur du laboratoire d'architecture avancée à Thalès pour ses remarques judicieuses portant sur des perspectives industrielles ;

Mme. Dominique Borrione, Professeur à l'Université Joseph Fourier, pour son soutien et pour avoir présidé mon jury.

Pendant cette thèse, plusieurs personnes qui me tiennent à cœur m'ont soutenue aussi bien par la pensée que par leur présence. J'aimerai exprimer ma gratitude et mes remerciements:

A ma source de vie papa et maman,

A mes joyaux, mes frères Mehdi et Hamdi qui ont donné un sens à ma vie,

A mes deux soleils Aimen et Wassim rayonnant mes jours et nuits, mes deux éternels amis avec qui j'ai su l'exactitude de l'expression « *amis pour la vie* »,

A mon bouquet de fleurs : mon lilas Faiza, ma rose Amel, mon jasmin Yasmine, mon iris Leila, mon gardénia blanc Halima, mon orchidée Abir, ma myrte Sameh, mon violet Yosra, mon angélique Saliha, mon camélia Nadia et mon giroflée Karima.

A mes amis sincères : Abdel Aziz, Mohamed (connu HAMMA), Jean Pierre, Ludovic, Amine, Christophe, et mon cousin Seif Eddine,

A mon ange gardien et mon étoile polaire des nuits sombres, ma très chère Sonja,

A mes précieux, mes chers Laurent et Brigitte et leurs deux petites jumelles,

A mon archipel brésilien Adriano qui m'a fourni la sécurité nécessaire pour plonger au cœur des travaux de cette thèse,

A mon prince préféré Damien pour ces conseils et son savoir faire princier,

Aux juniors de SLS : Frederic.H, Arnaud, Marius, Iuliana, Youssef, Mohamed, SangIl, YoungChul, Marcio et Ivan,

Aux seniors de SLS : Lovic, Wander, Sungjoo, Ferid, Sami, Anouar, Moez, Amer et Arif,

Aux sages de SLS : Paul, Nacer, Fréderic.R et Frederic.P

Aux autres collègues de SLS et aux autres membres de TIMA,

A mes instituteurs, mes professeurs depuis la maternelle jusqu'à l'université qui m'ont soutenue jusqu'au bout ; une pensée spéciale à mon professeur défunt Mahmoud Ezzaeiri qui était un deuxième père,

A ma chère Isabelle, pour sa patience et son aide précieuse,

A ma grande famille paternelle et maternelle,

A toutes les personnes que j'ai connues.

Pour leur exprimer ma profonde gratitude

A toutes épreuves bénéficiant de leur sollicitude

Par leur soutien réconfort et leur aptitude

Ayant toujours cru en moi avec certitude

Malgré les souffrances et les adversités

A travers le temps, journées et nuitées

Merveilleux parents toujours déterminés

A m'accorder leur amour avec ténacité

Nobles âmes à qui je dois ma prospérité

« Papa, Maman, Je vous aime tant et je vous chérie à jamais »

SOMMAIRE

LISTE DES FIGURES

LISTE DES TABLEAUX

Chapitre 1 Introduction

SOMMAIRE

1.1 Conception des systèmes embarqués

Cette thèse s'inscrit dans le cadre de la conception des systèmes hétérogènes embarqués sur puce.

Dans la dernière décennie a eu lieu une révolution des techniques de conception des circuits intégrés donnant naissance à la conception des systèmes embarqués sur puce (SOC pour « System On Chip » en anglais). Les applications actuelles telles que les systèmes multimédia, les applications des jeux vidéo, les terminaux de mobiles, etc. requièrent un ou plusieurs processeurs ce qui a influencé le marché des semi-conducteurs. Selon les systèmes de prévisions stratégiques du rapport ITRS [Itr01], 70% des ASIC (Application Specific Integrated Circuit) comporteront au moins un CPU embarqué à partir de l'année 2005. Ainsi, la plupart des ASIC seront des systèmes monopuces. De plus, ces puces contiendront des éléments non numériques (par ex. analogiques, RF, etc.) et des mécanismes de communication très sophistiqués (réseau sur puce, réseau de bus, etc.). Il est donc prévu que ces systèmes convoitent des marchés de masse. Il est alors nécessaire de maîtriser la conception de tels systèmes tout en respectant les contraintes de mise sur le marché et les objectifs de qualité.

Les systèmes sur puces sont un ensemble de composants hétérogènes. En effet, afin de maîtriser leur complexité et de réduire le temps de conception, la philosophie « *diviser pour régner* » est adoptée. Car, la conception des différents composants nécessiterait des compétences de différentes équipes de développement selon les besoins, les compétences et les fonctionnalités, etc.

Chaque équipe utilise son propre environnement de conception selon l'application à concevoir (niveaux d'abstraction, modèles de calcul appropriés) et exploite pour chaque domaine d'application les moyens les plus appropriés pour la description (fonctionnelle, logicielle ou matérielle).

Il devient primordial pour l'avenir de travailler à globaliser et uniformiser les systèmes de conception sur puce. Ceci peut être vu sous deux aspects : la conception des composants et l'intégration des différents composants dans le même système. Le travail de cette thèse porte sur l'aspect de l'intégration des composants. Bien que l'intégration facilite à terme la conception des sous-systèmes, elle introduit actuellement d'autres défis pour la conception. Ces défis peuvent être résumés comme suit :

1. **La maîtrise de la diversité des concepts** : La conception des sous-systèmes à différents niveaux d'abstraction avec différents langages de modélisation exige une connaissance multidisciplinaire des différents concepts utilisés. Cependant, cette diversité des langages et la confusion des notations selon le domaine d'application rendent la compréhension des concepts fastidieuse et difficile à établir.

2. **La modélisation globale du système** : La modélisation globale du système est indispensable pour la compréhension de certaines caractéristiques du système comme la fonctionnalité ou l'analyse des performances. Cependant, l'hétérogénéité des composants d'un système embarqué rend difficile leurs descriptions et la description de leurs interactions dans un langage unifié. Ceci est dû d'une part à l'insuffisance des modèles actuels pour couvrir toutes les caractéristiques et d'autre part à l'existence des modèles bien définis, et qui sont utilisés par la plupart des communautés. Il serait alors difficile de les remplacer par d'autres.
3. **La validation globale d'un système sur puce** : la validation globale d'un système est nécessaire afin de vérifier ses différentes fonctionnalités tout au long du flot de conception. Toutefois, l'hétérogénéité des composants des systèmes embarqués rend cette validation globale très difficile. Cela nécessite **un modèle d'exécution** des systèmes hétérogènes qui puisse s'accommoder de différents protocoles de communication, à différents niveaux d'abstraction ou de langages de modélisation. Cependant, même si des travaux sont en cours actuellement, un tel modèle d'exécution n'existe pas encore. De plus, les modèles d'exécution et de validation actuels (par ex : simulateurs, émulateurs, compilateurs, etc.) sont très souvent validés et devenus des standards. Dans ces cas, il serait dommage de les abandonner.

1.2 Objectifs

L'objectif de cette thèse est de généraliser le concept de modélisation et de validation globale des systèmes embarqués sur puce afin de définir un modèle d'exécution global de ces systèmes en se basant sur l'assemblage des modèles existants. Pour élaborer cette généralisation, il est nécessaire de :

1- Comprendre les concepts des différents modèles et des langages existants pour réussir à faire évoluer les phases de la modélisation et de la validation. Cette compréhension va permettre d'élaborer une terminologie commune qui permettra une spécification globale des systèmes hétérogènes.

2- Etudier les modèles de spécification et les modèles d'exécution existants pour définir et généraliser un modèle d'exécution global des systèmes embarqués sur puce. Les modèles existants ne s'appuient pas sur des stratégies globales pour la validation ce qui réduit leur utilisation à des cas particuliers (logiciels, ou matériels). Le but consiste alors à essayer de définir rigoureusement tous les concepts de base pour réaliser une validation conjointe des différents modules appartenant à plusieurs domaines d'application et pouvant être décrits à plusieurs niveaux d'abstraction, tout en définissant un modèle ayant des caractéristiques de modularité et de flexibilité.

a. *Flexibilité* : La flexibilité d'un modèle reflète sa puissance pour l'adaptation des différents besoins. En effet, le modèle d'exécution doit permettre un changement aisé du niveau d'abstraction ou du langage de spécification d'un composant. Cette flexibilité doit permettre de choisir le meilleur compromis vitesse/précision en choisissant le niveau d'abstraction et le langage de spécification approprié à chaque composant.
b. *Modularité et extensibilité* : Le modèle d'exécution doit donc nous permettre une adjonction aisée d'autres modèles. Il offre ainsi une certaine facilité d'intégration dans des environnements différents et une meilleure exploration d'architectures.

3- Valider l'ensemble du système à toutes les étapes de sa conception.

1.3 Contributions

Cette thèse se focalise sur la description et la simulation des systèmes hétérogènes embarqués dans le but d'évaluer leurs performances avant d'en dériver les réalisations finales matérielles et/ou logicielles. Une description est définie par rapport à des modèles décrits à différents niveaux d'abstraction et en utilisant plusieurs langages en différents modes (textuel, graphique normalisé, etc.). De plus, cette thèse s'intéresse aux mises en œuvre (aux implémentations) de ces descriptions. Cette mise en œuvre peut être de trois types : logicielle, matérielle ou indéterminée. Dans le dernier cas, la mise en œuvre est dite fonctionnelle.

Ce travail apporte trois contributions. La première contribution est l'étude des différents concepts relatifs à la modélisation à travers différents langages. Ceci pour définir une **terminologie commune** et pour développer les concepts de base pour la modélisation. De plus, une définition séparée des « modèles de calcul » et des « modèle d'exécution » était nécessaire. Le premier permet de décrire ce que fait un système (contrôles ou traitements de données). Le second permet de décrire comment le calcul est implémenté. On distinguera trois types d'implémentation : logicielle, matérielle et fonctionnelle.

La seconde contribution de ce travail est la séparation entre modèle de calcul et modèle d'exécution. Cette séparation est nécessaire pour interpréter les modélisations de systèmes composés de plusieurs modules hétérogènes. Chacun de ces modules est modélisé à son tour avec différents langages, graphes, équations, etc. Ceci conduit à des systèmes hiérarchiques complexes qu'il faut assembler afin d'obtenir un modèle global. L'assemblage est défini non simplement par la mise en correspondance des noms mais par un processus de composition complexe pouvant cacher des traitements et des protocoles complexes. Cette composition est déterminée par l'intermédiaire de certains éléments d'assemblage non pas par le calcul mais aussi par la manière

de communication. Il est donc nécessaire de définir le moyen de composition : c'est le concept d'interconnexion hétérogène.

La confusion des modèles de calcul et d'exécution a été un handicap pour tous les travaux antérieurs qui ont essayé d'étudier la composition de systèmes hétérogènes. Ils se sont heurtés à la diversité des modèles existants et ont souvent abouti à certaines hypothèses trop réductrices pour être utiles dans le cas des systèmes complexes. Cette distinction a été faite par le **recensement des modèles de calcul** et **des modèles d'exécution** existants les plus usités.

La troisième contribution est une conséquence des études précédentes. Les concepts extraits de ces deux études seront le point de départ pour la définition et la **généralisation d'un modèle de spécification et d'exécution global** des systèmes hétérogènes sur puce. Les différents éléments de ces modèles seront détaillés tout au long du flot de conception des systèmes embarqués sur puce. Ces modèles devront être généraux, modulaires et flexibles.

Chacune de ces contributions sera présentée plus en détail dans les paragraphes ci-après.

1.3.1 Etude des différents langages de modélisation

Deux catégories de langages de modélisation existent :

1. Les langages non exécutables : ce sont des langages avec lesquels seulement la description des systèmes est effectuée. La sémantique d'exécution n'est pas définie à travers ces langages. Pour obtenir un modèle exécutable, il est nécessaire de recourir à un autre langage. Parmi ces langages on relève le langage UML (Unified Modeling Language) [Uml02], IDL (Interface Description Language) [Idl05], ROOM (Real Object Oriented Modeling) [Roo94], etc.
2. Les langages exécutables : ce sont des langages qui permettent aussi bien la modélisation que l'exécution du fonctionnement spécifié. La sémantique d'exécution est définie grâce aux modèles d'exécution qu'ils offrent via les simulateurs, compilateurs, etc. Parmi ces langages, on relève SystemC [SyC02], VHDL [Vhd93], Verilog et Verilog-AMS [Pec05], C/C++ [Dup03], VHDL-AMS [Pec05], Java [Jav05], etc.

Ces langages définissent certains concepts essentiels pour la modélisation tels que la composition et la hiérarchisation. Le concept essentiel commun est la séparation entre le comportement et la communication. Par exemple, pour le langage SystemC, le comportement d'un système est défini par un ensemble de processus (concurrents ou sequentiels) alors que la communication est définie, dans ce langage par l'ensemble des interconnexions (canaux) entre les ports des différentes entités.

Toutefois, ces langages utilisent des notations et des terminologies spécifiques au domaine d'application (matériel, logiciel, etc.) dans lequel ils sont souvent employés. Cette première partie

de la thèse consiste en la définition des concepts de base communs utilisés pour la spécification des systèmes qui varient selon les langages. Ces concepts permettront la spécification des systèmes indépendamment des langages de spécification, de modélisation ou de programmation tout en respectant la séparation entre la communication et le comportement.

1.3.2 Modèles d'exécution des systèmes

Un modèle d'exécution est à même de traduire le comportement d'un système, mais aussi dans une certaine mesure la façon dont le système est implémenté. Par exemple, une machine d'états finis notée FSM (Finite State Machine), peut être réalisée logiciellement ou matériellement. On distinguera trois types d'exécution : logicielle, matérielle et fonctionnelle. Ces modèles peuvent être définis à travers certains langages de spécification ou de modélisation. Chaque modèle d'exécution présente des particularités et des caractéristiques qui lui sont propres (par exemple, l'implémentation associée, les niveaux d'abstraction et le type de la communication, etc.).

L'exécution des systèmes ne s'arrête pas au comportement mais aussi elle s'étend à l'interconnexion entre les différents sous-systèmes. Par exemple, l'interconnexion en VHDL est définie par l'ensemble des signaux entre les différents sous-systèmes et la propagation des données à travers ces signaux. Mais par contre, dans le langage C, l'interconnexion est définie par l'appel de fonctions et de procédures entre les sous-systèmes. Dans cette partie, les différents modèles sont présentés et détaillés. De plus, une étude comparative et une classification des différents modèles d'exécution sont réalisées.

1.3.3 Définition d'un modèle d'exécution des systèmes hétérogènes

La définition d'un modèle d'exécution global pour un système hétérogène n'est pas évidente. En effet, avoir un modèle global qui englobe tous les modèles adéquats des sous-systèmes et des modèles d'interconnexions nécessite de mettre en œuvre le principe d'interconnexion hétérogène de systèmes hétérogènes. Cela est dû à la manipulation de concepts et de propriétés différents. Il est plus intéressant de pouvoir réutiliser les concepts déjà existants.

Plusieurs travaux ont été effectués pour réaliser ce modèle d'exécution. Certains travaux se basent sur la définition d'un modèle unique et unifié comme « Coware » [Cow05] et « SpecSyn » [Gaj98]. Toutefois, ces modèles ne peuvent pas constituer un environnement capable de réunir les différents concepts des différents langages manipulés. Ils sont insuffisants pour définir un modèle d'exécution global pour les systèmes hétérogènes.

D'autres travaux définissent un modèle d'exécution par composition de différents modèles. Par exemple, les travaux de « Ptolemy » [Pto91] [Pto05] qui offre un environnement de composition de modèles d'exécution. Cependant, ce modèle est basé sur une représentation unifiée pour l'implémentation de modèles de calcul en se basant sur un ensemble de bibliothèques

qui fournissent les fonctions nécessaires pour la réalisation de ces modèles comme les travaux de MathWorks [Mat05] dans Simulink [Mat00] pour l'intégration de sous-systèmes numériques *via* ModelSim [Mod05] et Smash [Sma05].

Les travaux de « G.Nicolescu » [Nic03] constituent une approche intéressante dans le cadre de la composition de modèles d'exécution de différents sous-systèmes. Ils offrent une méthode de génération automatique de modèles d'exécution pour les systèmes hétérogènes. Cependant, ces travaux restent spécifiques à la composition de sous-systèmes décrits par un nombre restreint de niveaux d'abstraction. Dans cette thèse, les travaux de recherche se basent sur ceux de « G.Nicolescu » pour définir un modèle généralisé d'exécution, basé sur l'assemblage des différents modèles d'exécution des éléments qui le constituent.

1.4 Plan du document

Ce document est composé de quatre chapitres. Le chapitre 2 présente une étude conceptuelle pour la modélisation des systèmes hétérogènes embarqués. Dans le Chapitre 3 une autre étude conceptuelle est effectuée sur les modèles d'exécution des systèmes homogènes où nous proposons les différents modèles logiciels, matériels et fonctionnels pour l'exécution des systèmes. A partir de ces études, le chapitre 4 présente un modèle généralisé pour la spécification et la validation des systèmes hétérogènes embarqués. Pour finir, le dernier chapitre illustre l'utilisation des concepts définis pour trois applications complexes : un centre d'information pour les véhicules, un modem d'une chaîne audio 802.16 et un système de radio défini par le logiciel SDR ('Software Defined Radio').

Chapitre 2 Modélisation des systèmes hétérogènes mono puce : étude conceptuelle

SOMMAIRE

2.1 Introduction

Modéliser un système consiste à en faire une représentation simplifiée. La modélisation est la base de tout acte de conception et/ou d'analyse des structures et de comportement. La modélisation existe bien au-delà des systèmes électroniques. La simulation de la fission atomique utilise des modèles de l'atome alors que la construction d'une maison utilise un modèle d'architecture.

L'abstraction de la composition et le raffinement représentent les instruments fondamentaux de la modélisation. Tout raisonnement sur les systèmes, mais plus encore, toute manipulation d'un système nécessite une modélisation.

Ce chapitre se limite au domaine des systèmes électroniques dont la réalisation met en œuvre des parties matérielles, fonctionnelles et des parties logicielles.

Les approches pour la modélisation sont aussi nombreuses que les métiers nécessaires à la réalisation d'un système. Pour les spécialistes de la définition de nouveaux produits, un modèle doit permettre la saisie des aspects fonctionnels et non fonctionnels du système. Pour l'architecte, la modélisation permet de raisonner sur le découpage du système et le choix des technologies nécessaires (par exemple, logicielle ou matérielle) à la réalisation des différentes parties.

La conception des différents sous-systèmes peut nécessiter une modélisation à différents niveaux d'abstraction permettant de cacher plus ou moins de détails selon l'étape considérée. Différents modèles peuvent aussi être nécessaires pour les différents métiers de la conception (documentation, validation, analyse et raffinement). Pourtant, il existe des points communs à tous ces modèles et à tous ces champs d'application. Ce sont ceux que nous avons tenté de mettre en évidence en utilisant des concepts transversaux, issus de toutes les disciplines, et compréhensibles, nous l'espérons, par tous. De ce fait, le reproche que notre approche pourrait encourir est de manquer parfois de rigueur sémantique car certains mots prennent des sens différents selon le domaine dans lequel ils sont utilisés. Par exemple, le mot « connexion » utilisé par les concepteur du logiciel n'a pas le même sens que dans le langage commun, de même que celui de « communication » qui évoque des processus plus ou moins complexes et met en jeu des éléments différents selon qu'on fait de la simulation ou de la synthèse.

Même si les domaines de la modélisation sont nombreux et si l'interprétation en varie à l'infini, nombre d'éléments peuvent être soulignés. Les objets utilisés sont en réalité des variantes de quelques concepts fondamentaux.

Pour une compréhension plus aisée, et pour éviter les répétitions inutiles, nous avons, dans une première partie analysé ces concepts de base. La section 2 détaille les concepts de base utilisés pour tous les domaines de modélisation tels que les notions de système, module, interface, port, interconnexion et communication. Dans cette section, on introduit la nécessité de

faire la distinction entre les *modèles de calcul* et les *modèles d'exécution*. Le premier permet de décrire ce que fait un système en terme de composition hiérarchique d'éléments de calcul de base. Alors que le second permet de décrire comment le calcul est réalisé dans une technologie particulière. Dans ce chapitre, les modèles de calcul sont uniquement détaillés puisqu'ils sont dédiés à la spécification des systèmes. Les modèles d'exécution seront détaillés dans le chapitre suivant.

2.2 Concepts de base pour la spécification des systèmes hétérogènes

Ce paragraphe est dédié à l'analyse des concepts de base des systèmes hétérogènes monopuce. Cette analyse se propose de faire l'inventaire de l'ensemble de la terminologie nécessaire pour définir les systèmes hétérogènes. Nous commencerons par développer les définitions des éléments de base d'un système hétérogène. Cette étude sera suivie par la comparaison entre deux autres concepts étroitement liés : les modèles de calcul et les modèles d'exécution.

2.2.1 Système

On appellera « système » une entité formée par un ensemble de modules interconnectés de manière hiérarchique et qui communique avec l'environnement *via* une interface bien déterminée [Lam02].

2.2.2 Module

C'est une entité qui communique *via* une interface bien définie avec un environnement externe donné. Un module est composé par un contenu et une interface. L'interface du module est connectée au réseau d'interconnexion. Un module peut être élémentaire ou hiérarchique.

Un module élémentaire représente une feuille dans la hiérarchie qui englobe un comportement élémentaire (tâche, processus, fonction, etc.).

Un module hiérarchique est un ensemble de modules élémentaires ou hiérarchiques interconnectés. Le module hiérarchique peut être aussi utilisé par d'autres systèmes ou par le même système. *Un système* peut être considéré alors comme un module hiérarchique.

La représentation graphique du module est donnée par la figure suivante.

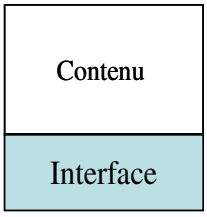
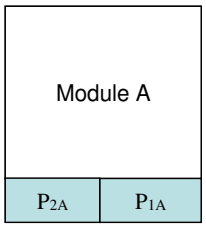


Figure 2-1 Représentation d'un module

La Figure 2-2 représente une description d'un module ayant un contenu et une interface en utilisant le langage SystemC [SyC02]. Le contenu est représenté par le processus « comportement » (noté ici par SC_THREAD (comportement)). L'interface est représentée explicitement par un ensemble de ports P1A, P2A. Ces ports sont des ports logiques élémentaires (voir 2.2.3). Dans cet exemple, deux modèles de ports sont représentés: les ports d'entrées notés « sc_in » et les ports de sorties notés « sc_out ».



(a) Représentation graphique du module A

```
SC_MODULE (A) {

/*Interface*/

            sc_out<double >   P2A;
            sc_in<int >       P1A;

/*Comportement */
            void comportement();
            SC_CTOR (A)
            {

            SC_THREAD (comportement);

            };
};
```

(b) Modélisation du module A avec le langage SystemC

Figure 2-2 Description d'un module A en SystemC (comportement et interface)

2.2.3 Interface

Une interface est un ensemble d'accès permettant aux modules d'interagir avec le reste du système et avec l'environnement externe.

La description des interfaces peut être plus ou moins détaillée. Selon le niveau d'abstraction, elle doit contenir l'information décrivant comment avoir accès aux composants ou comment ces composants peuvent accéder au reste du système. Une interface est vue différemment à travers les niveaux d'abstraction. Elle peut être soit des broches pour un circuit, soit une description de paramètres d'une fonction C ou un ensemble de procédures réalisées par les modules. Dans le dernier cas on parle d'API (Application Programming Interface).

Ainsi, dans cette étude, l'interface est définie par un ensemble de **ports** et de **services** selon la nature d'accès aux composants.

- Les services : ils sont définis par les APIs ou les méthodes d'appel de fonctions permettant l'accès au module. Ils sont présentés à travers les langages de haut niveau tel que C++ [Dup03], Java, etc. Nous les présentons par la suite par un losange.

- Les ports physiques : ils permettent de faire référence à un accès physique représentant par exemple l'accès à une porte logique (ex. porte « AND »). Les langages représentant une interface de ports physiques sont VHDL [Vhd93], SystemC [Hav02]. Nous les représentons par la suite par des octogones.
- Les ports logiques sont des entités qui fournissent et/ou demandent un ou plusieurs services et qui font partie de l'interface d'un module. Ils peuvent regrouper un ou plusieurs ports physiques et/ou des abstractions de ces ports. Le port logique peut être vu également comme une entité fonctionnelle présentant les ports d'un module décrits à un haut niveau d'abstraction [Sar05a]. Pour les ports logiques, des actions sont associées sous forme de services fournis par ceux-ci. Ainsi, un port logique est vu comme $\{P_I, A_J\ J=0..M, I= 0..N\}$, avec P_I désignant le port I, et A_J un service du port logique (par exemple un service d'écriture). Pour accéder aux ports, on définit un Point d'Accès au Port, noté PAP (Port Access Point). Pour accéder aux services, on définit des Points d'Accès Service noté SAP (Service Access point). La figure suivante spécifie le contenu d'un port logique.

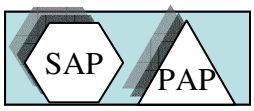


Figure 2-3 Port Logique

Pour simplifier, on ne présente les ports logiques que par des rectangles.

L'interface, ainsi définie, peut être présentée comme suit :

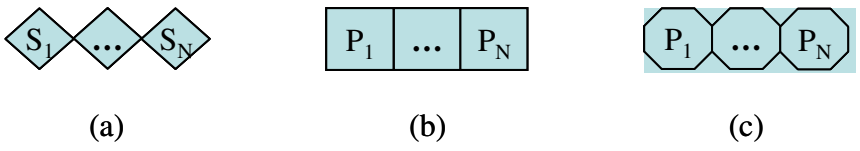
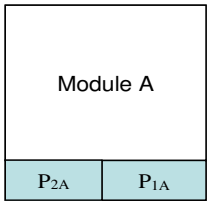


Figure 2-4 Interface : (a) ensemble de services, (b) ensemble de ports logiques, (c) ensemble de ports physiques

L'exemple de la Figure 2-5 décrit un module A en utilisant le langage VHDL non synthétisable.



(a) Représentation graphique du module A

```
/* Présentation de l'interface d'un module A en VHDL */
ENTITY A is
port(
        P1A : in integer;
        P2A : out real;)
end A;
```

(b) Modélisation de l'interface du module A avec le langage VHDL

Figure 2-5 Description de l'interface d'un module avec le langage VHDL

L'interface du module A est un ensemble de ports logiques élémentaires (pouvant être la représentation de broches d'un circuit). Elle est composée de deux ports P1A (port d'entrées) et P2A (un port de sortie). Les services associés aux ports logiques sont définis implicitement grâce à la description, dans la bibliothèque, de VHDL des ports correspondants. Généralement, les services définis sont de lectures et d'écritures.

Les ports et les services peuvent être simples (élémentaires) ou hiérarchiques (c'est-à-dire englobant plusieurs ports et/ou services ayant une même sémantique par exemple des ports physiques appartenant au même protocole de communication).

La description d'un système nécessite la déclaration des modules qui le compose avec leur différentes interfaces et différentes interactions. Considérons alors, un système ayant trois modules communicants *via* leur interface de communication. Cette interface est un ensemble de ports physiques. Les modules A et B communiquent *via* leurs ports respectifs P_{1B} et P_{1A}. Ces deux ports doivent spécifier un protocole de communication (par exemple Handshake). En effet, ce protocole modélise une régulation du flux de données en utilisant trois signaux transportés par des fils séparés. Ces signaux peuvent être (1) un signal servant à l'établissement de la connexion entre les deux modules (Rdy), (2) un signal transportant les données (Data) et (3) un signal servant pour l'acquittement (Ack). La spécification d'un tel protocole peut être faite alors en utilisant trois ports représentant les signaux respectifs. Toutefois, ces trois ports peuvent être regroupés dans les ports P_{1B} comme le montre la Figure 2-6. Ce port est dit alors port hiérarchique.

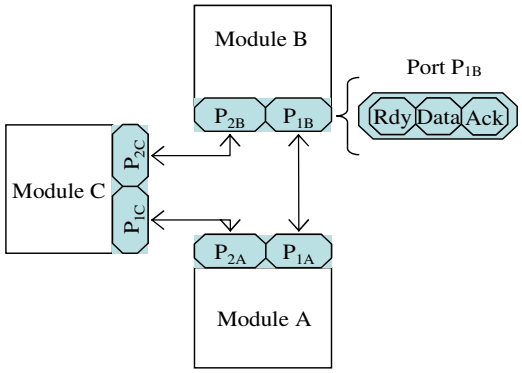


Figure 2-6 Système ayant un port hiérarchique (P_{1B})

La figure 2-7 présente un récapitulatif des différents types d'interface rencontrés lors de la modélisation d'un système.

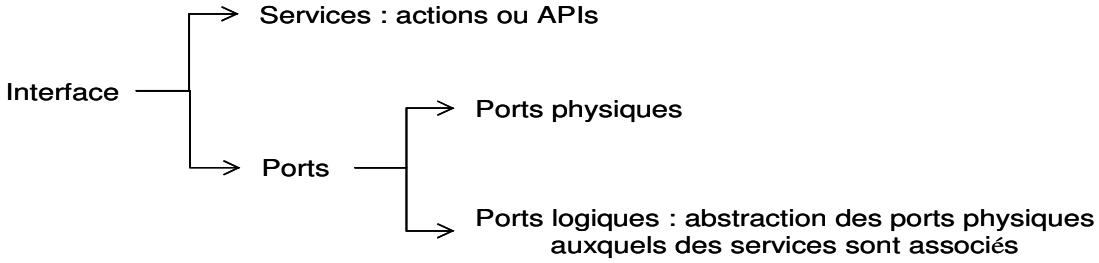


Figure 2-7 Récapitulatif de la définition d'une interface

Dans la suite, la communication et l'interconnexion seront séparées. Le terme « interconnexion » désignera la mise en correspondance de deux interfaces et « communication » désignera la procédure utilisée pour l'échange d'information entre les modules.

2.2.4 Interconnexion

L'interconnexion permet la mise en relation de plusieurs modules pour rendre leur fonctionnement interdépendant [Tsi03]. Dans le cas des systèmes composés, cette mise en relation se situe au niveau des interfaces donc au niveau des ports et des services.

L'interconnexion se fait à travers des moyens de raccordement qui peuvent être des réseaux (par exemple Æthereal de Philips, Octagon de STMicroelectronics, etc.), des canaux abstraits (par exemple canaux FIFO, canaux MPI, Bus ORB, etc.) ou physiques (par exemple fils physiques, etc.) selon le niveau d'abstraction. L'interconnexion assure le transport des données entre les modules. Elle peut englober certains détails de communication pouvant être transparents pour les modules qu'elle interconnecte. L'interconnexion est aussi désignée par le terme « canal de communication ».

Les concepts qui définissent cette interconnexion sont :

- Le média qui véhicule l'information (par exemple des fils physiques ou réseaux),
- Le type de données transmises (par exemple des données génériques ou données avec représentation fixe),
- Le comportement (la procédure utilisée pour acheminer les données à travers le média).

Ainsi une interconnexion abstraite (canal de communication abstrait) peut être raffinée en un module particulier. Cependant, la séparation entre les deux (module et canal) peut avoir un intérêt durant le flot de conception. Les méthodes de raffinement de la communication et du calcul utilisent généralement des techniques différentes.

L'exemple de la Figure 2-9 spécifie une description en SystemC des interconnexions point à point entre les modules A, B, et C de la Figure 2-8. Chaque module possède deux ports logiques. Le Module A possède une interface composée par les ports P1A (port d'entrée véhiculant des données de type « entier ») et P2A (port de sortie véhiculant des données de type « double »). Le module C possède une interface composée de P1C (port d'entrée véhiculant des données de type « double »), P2C (port de sortie véhiculant des données de type « double »). Le module B a une interface P1B (un port logique de sortie véhiculant des données de type « entier ») et P2B (port logique d'entrée véhiculant des données de type « double »). L'association des ports est donnée comme suit : P1A avec P1B, P2A avec P1C et P2B avec P2C. L'interconnexion est réalisée à travers des fils logiques représentés, dans ce langage de modélisation, par la notion de « sc_signal ».

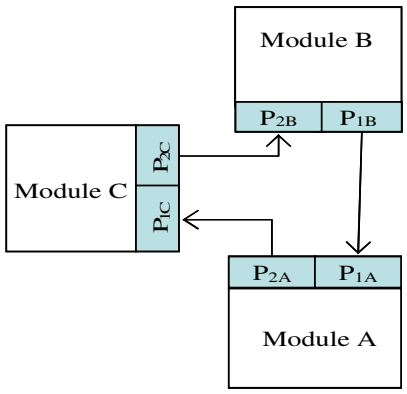


Figure 2-8 Interconnexion entre trois modules ayant des ports logiques comme interfaces de communication

```
sc_main(argc, argv)
{
/* déclaration des signaux*/
sc_signal<double> S1, S2;
sc_signal<int> S3;

/* Instanciation des modules */

A M_A("module A");
B M_B("module B");
C M_C("module C");

/* Mise en correspondance des ports P1A et P1B*/
M_A.P1A(S3);
M_B.P1B(S3);

/* Mise en correspondance des ports P2A et P1C*/

M_A.P2A(S2);
M_C.P1C(S2);

/* Mise en correspondance des ports P2B et P2C*/

M_A.P2B(S1);
M_B.P2C(S1);
```

Figure 2-9 Exemple SystemC d'interconnexion

Grâce à la notion de mise en correspondance on peut définir une nouvelle notion qui est la compatibilité d'interfaces.

Deux interfaces I1 et I2 sont dites compatibles si elles peuvent être interconnectées par une simple mise en correspondance. Il s'agit d'une généralisation des règles de connexions utilisées dans les langages de description de matériel. Par exemple, un port d'entrée et un port de sortie ayant la même structure (type et/ou représentation de données) peuvent être mis en correspondance par un simple fil physique ou logique dont le rôle est juste d'acheminer les informations entre les deux ports physiques. Pour étendre cette notion aux interfaces telles que définies plus haut, il est nécessaire d'étendre cette notion aux ports logiques, aux services et aux ports hiérarchiques.

Ainsi, la simple mise en correspondance de deux interfaces dans le cas général nécessite que les éléments constituants les interfaces aient la même hiérarchie et la même nature (services, logique ou physique), c'est-à-dire tous les ports et services élémentaires sont compatibles deux à deux. Dans le cas où les ports (logiques ou physiques) et services mis en correspondances sont hiérarchiques, l'interconnexion est aussi dite interconnexion hiérarchique.

Il est aussi important de noter que dans la suite, on fera l'hypothèse que les services définis pour la communication sont indépendants du module qui les utilise. C'est-à-dire que :

- le déroulement d'un service ne peut pas dépendre de l'état du module mais simplement de ses paramètres.

- Le raffinement d'un service peut être réalisé sans changer la description du module si le raffinement respecte les API utilisés par le module.

On distingue deux types d'interconnexions :

- **les interconnexions homogènes** permettant de mettre en correspondance des interfaces compatibles. Elles sont de trois types :
    - Interconnexion physique ou encore explicite : Une interconnexion physique est une mise en correspondance de ports physiques. Elle peut être définie comme les règles de connexions utilisées dans les langages de description de matériel (Figure 2-16 et Figure 2-17).
    - Interconnexion logique : elle met en correspondance des ports logiques. Elle présente des associations entre les services et entre les ports des interfaces respectives. Elle peut être considérée comme un regroupement d'interconnexion physique en une interconnexion plus abstraite ou un raffinement d'une interconnexion abstraite (Figure 2-15).
    - Interconnexion abstraite ou encore implicite : elle met en correspondance des services. Par exemple, un système peut être vue comme un ensemble de composants ayant des services (API) de communication permettant le dialogue entre eux sans préciser la nature des interconnexions (ex. appel de fonctions). L'interface des différents composants est vue, dans ce cas, comme des services (voir 2.2.3). La Figure 2-10 peut être une vision d'une telle description. Comme exemple, on considère un système décrit avec un des langages parallèles nommé MPI (Message Passing Interface [Gro96]). Ce système comporte deux composants un émetteur et un récepteur. L'interconnexion entre ces deux composants est cachée derrière les API de communication (dans l'exemple, les services sont définies par les API de communication « MPI_send » et « MPI_Recv »). Dans ce cas, l'interconnexion est vue comme un média logique ayant un comportement abstrait et de type de données compatibles (voir Figure 2-10 et Figure 2-11).

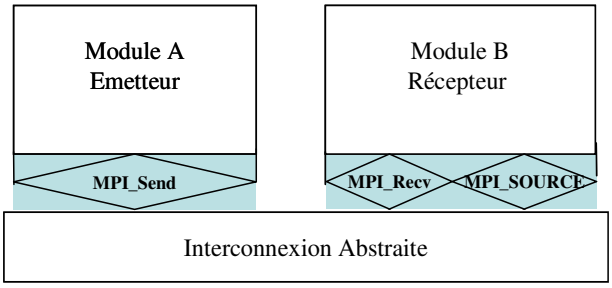


Figure 2-10 Abstraction de l'interconnexion

```
main(int argc, char *argv[])
{
  /* Déclaration des paramètres nécessaires pour le programmes
  const  int tag ;
  int id, ntasks, source_id, dest_id, err, i;
     MPI_Status status;
     int msg[2];
  /* instructions relatives à MPI, initialisation … */
/* Tester le nombre de composants */
if (ntasks != 2) {
    printf("You have to use 2 processors to run this program\n");
    MPI_Finalize();              /* Quit if there is only one processor
*/
    exit(0);
  }
  /* Module B : le récepteur */

  if (id == 0 || id ==2) {
    for (i=1; i<ntasks; i++)
      {
      err = MPI_Recv(msg, 2, MPI_INT, MPI_ANY_SOURCE, tag,
      MPI_COMM_WORLD, &status);          /* Recoit le message */
      source_id = status.MPI_SOURCE;      /* avoir le id de l'emetteur */
      printf("Received message %d %d from process %d\n", msg[0], msg[1],
source_id);
    }
  }
  /* Module A : l'émetteur */

  else if (id ==1)
  {
    msg[0] = id;      /* identifier le  message */
    msg[1] = ntasks;        /*identifie le nombre total des processus */
    dest_id = 0;      /* adresse de destination */
    err = MPI_Send(msg, 2, MPI_INT, dest_id, tag, MPI_COMM_WORLD);
  }

  err = MPI_Finalize();          /* Terminer MPI */
  if (id==0) printf("Ready\n");
  exit(0); }
```

Figure 2-11 Code MPI présentant une description d'une interconnexion abstraite

- **les interconnexions hétérogènes** permettant de mettre en correspondance des interfaces non compatibles. Le comportement d'une telle interconnexion doit permettre la conversion d'information entre les différents éléments (services, ports logiques ou physique) qu'elle connecte. Cette interconnexion présente une problématique de modélisation des systèmes et de leurs exécutions effectives.

La Figure 2-12 invoque une interconnexion hétérogène d'un système composé de deux modules A et B ayant des ports non compatibles. Le module A possède trois ports logiques (par exemple les ports P_{1A}, P_{2A} et P_{3A} sont des ports logiques véhiculant des données de type « entier ») alors que le module B possède deux ports physiques (par exemple P_{1B} et P_{2B} véhiculant des données de type « bit »). Le nombre et la nature des interfaces des deux modules sont différents.

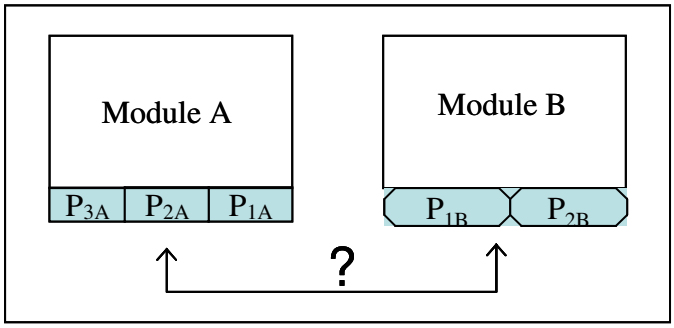


Figure 2-12 Système ayant une interconnexion hétérogène

L'interconnexion, ainsi définie, entre les différents modules peut se faire selon différents schémas d'interconnexion point à point ou multipoints. Dans tous les cas on représentera une interconnexion par un filet ou réseau de fils liés, il s'agit d'une généralisation de la représentation du « net » dans les langages de description du matériel.

- Interconnexion point à point : Une interconnexion point à point représente un lien direct entre exactement deux interfaces. Par exemple le lien va être une connexion par un signal entre deux ports physiques concernés. La Figure 2-13 illustre un schéma d'interconnexion point à point (par exemple les ports physiques élémentaires P_{1B}, P_{2B} du module B sont en lien direct avec les ports physiques élémentaires P_{1A} du module A et le port P_{2C} du module C respectivement).

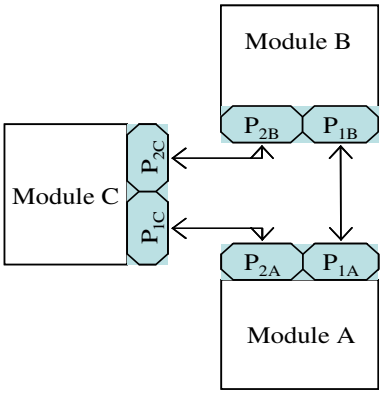


Figure 2-13 Schéma d'interconnexion point à point

- Interconnexion multipoints : Une interconnexion multipoint met en correspondance plusieurs interfaces en même temps. Dans certains cas, la connexion multipoint peut générer des conflits et peut donc nécessiter des fonctions d'arbitrage. Selon le niveau d'abstraction, la fonction d'arbitrage peut être simple telle que le choix d'une valeur parmi n. La fonction d'arbitrage peut être aussi plus complexe et peut mettre en œuvre des routages ou de la négociation de services (voir Figure 2-14).

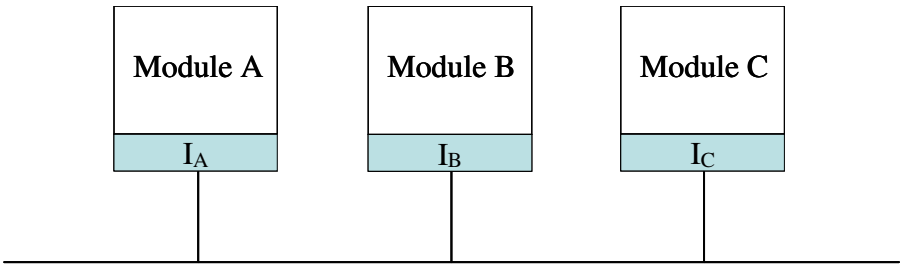


Figure 2-14 Connexion multipoint « interconnexion simple »

Un schéma de description de ce système, en SystemC, sera donné par la Figure 2-15.

```
sc_main(argc, argv)
{
/* déclaration de l'interconnexion de type FIFO */
sc_fifo Fifo ;

/* instanciation des modules */

A M_A("module A");
B M_B("module B");
C M_C("module C");

/* Mise en correspondance interface I1, I2 et I3*/
M_A.IA(Fifo);
M_B.IB(Fifo);
M_C.IC(Fifo) ;
```

Figure 2-15 Exemple en SystemC d'une interconnexion multipoint simple

Dans cet exemple, l'interconnexion est un canal FIFO décrit en haut niveau et que les interfaces correspondantes aux modules A, B et C sont de simples ports logiques notés IA, IB et IC respectivement.

- Dans le reste de ce document, dans les cas autres que la mise en correspondance directe, on considère que l'interconnexion cache un comportement complexe et peut être représentée par un module défini par un comportement et une interface. La Figure 2-16 illustre une interconnexion multipoint où l'interconnexion est vue comme un module particulier. L'interface du module interconnexion (I_{S1}, I_{S2}, I_{S3}) met en correspondance tous les ports des modules A, B et C et prend en charge l'acheminement des informations transférées entre les différents modules.

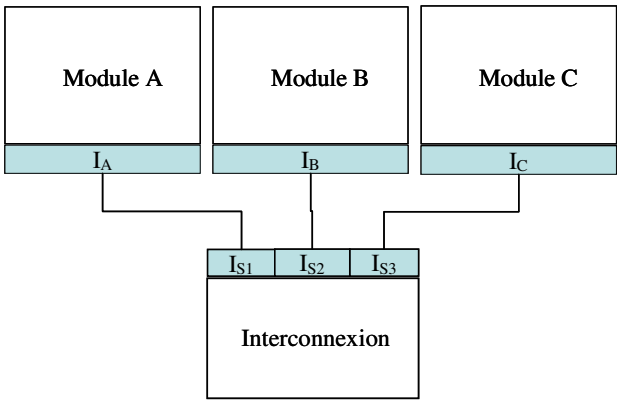


Figure 2-16 Schéma d'interconnexion multipoint « interconnexion complexe »

Un schéma de description de ce système en SystemC pour l'interconnexion multipoint complexe sera donné par la Figure 2-18. L'interconnexion est représentée dans ce cas par une instance d'une interconnexion de type bus AMBA (noté AHB). L'exemple comporte deux modules (A et B) et le module d'interconnexion AHB [Ahb99]. Les interfaces des modules A et B sont composées de dix ports physiques et huit ports physiques respectivement (voir Figure 2-17).

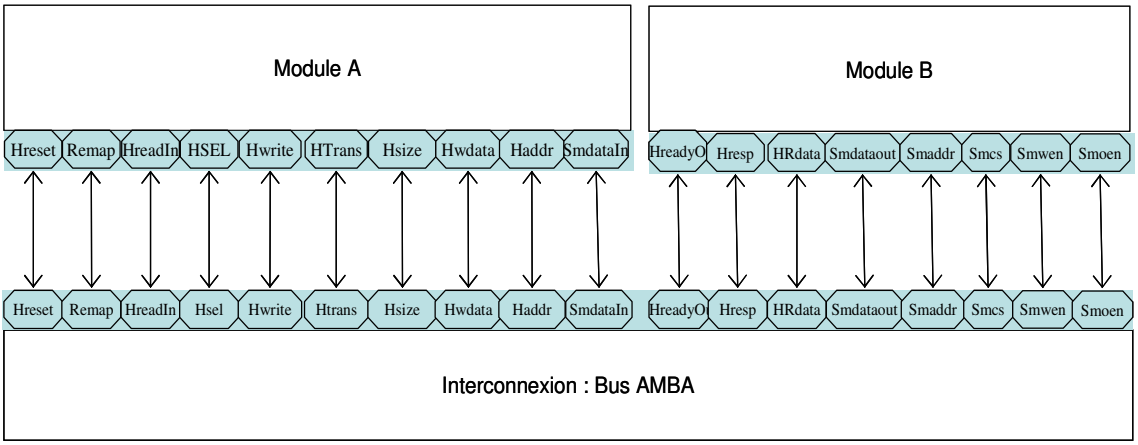


Figure 2-17 Interconnexion multipoint complexe avec le bus AMBA

```
// Déclaration des signaux de connexions
      sc_signal<bool>   RESETn, RMP, READYIn, SEL, WRITE, READYOut;
      sc_signal<sc_uint<2> >  TRANS, RESP;;
      sc_signal<sc_uint<3> >  SIZE;
      sc_signal<sc_uint<32> > WDATA, ADDR, DATAIn, RDATA, DATAOut;
      sc_signal<sc_uint<26> > SADDR;
      sc_signal<sc_uint<8> >  CS;
      sc_signal<bool>         WEn, OEn;
// Instanciation des modules
      A M_A("module A");
      B M_B("module B");
// Déclaration du module de l'interconnexion
      AHB smi("AHBSmi");
// Définir les relations entre les modules et l'interconnexion

      M_A.HRESETn(RESETn);
      M_A.REMAP(RMP);
      M_A.HREADYIn(READYIn);
      M_A.HSEL(SEL);
      M_A.HWRITE(WRITE);
      M_A.HTRANS(TRANS);
      M_A.HSIZE(SIZE);
      M_A.HWDATA(WDATA);
      M_A.HADDR(ADDR);
      M_A.SMDATAIn(DATAIn);

      M_B.HREADYOut(READYOut);
      M_B.HRESP(RESP);
      M_B.HRDATA(RDATA);
      M_B.SMDATAOut(DATAOut);
      M_B.SMADDR(SADDR);
      M_B.SMCS(CS);
      M_B.SMWEn(WEn);
      M_B.SMOEn(OEn);

// Connexions du module AHB avec le reste des modules
      // Avec le module A
      smi.HRESETn(RESETn);
      smi.REMAP(RMP);
      smi.HREADYIn(READYIn);
      smi.HSEL(SEL);
      smi.HWRITE(WRITE);
      smi.HTRANS(TRANS);
      smi.HSIZE(SIZE);
      smi.HWDATA(WDATA);
      smi.HADDR(ADDR);
      // Avec le module B
      smi.SMDATAIn(DATAIn);
      smi.HREADYOut(READYOut);
      smi.HRESP(RESP);
      smi.HRDATA(RDATA);
      smi.SMDATAOut(DATAOut);
      smi.SMADDR(SADDR);
      smi.SMCS(CS);
      smi.SMWEn(WEn);
      smi.SMOEn(OEn);
```

Figure 2-18 Exemple d'un système décrit en SystemC d'une interconnexion multipoint complexe avec le bus AMBA

2.2.5 Communication

La communication est l'échange ou le transfert des informations entre différents modules d'un système conformément à des conventions pré-établies.

On supposera que la communication se fait en utilisant les actions sur les ports. Ainsi pour un module, les actions de communication sur les ports constituent une API permettant l'accès au reste du système et de l'environnement. La définition de cet API constitue une décision d'architecture. Elle permet de séparer le comportement du module des services fournis par l'environnement du module. C'est ce qu'on appelle communément séparation de la communication et du calcul [Lee99].

2.2.6 Hiérarchie

La notion de la hiérarchie en informatique introduit souvent la notion d'héritage (exprimée essentiellement dans les modèles relationnels). Pour les systèmes électroniques cette définition rejoint celle de la composition.

2.2.7 Composition

La composition est une collection organisée de composants en interaction pour accomplir un comportement cohérent et commun [Tsi03].

Un système est dit composé s'il est constitué d'un ensemble de plusieurs modules interconnectés. Chaque module peut être un module élémentaire ou lui-même un module composé (hiérarchique) (voir 2.2.2).

Ainsi dans le cas qui nous intéresse, un système est une composition où les interconnexions servent à mettre en correspondance les ports des différents modules.

La Figure 2-19 et Figure 2-20 illustrent un exemple de module C composé de deux modules D et F. Le module C comporte une interface composée de ports logiques PC1 et PC2. L'interface du module D comporte le port logique PD. L'interface du module F comporte le port logique PF. Les ports PC1 et PC2 sont connectés aux ports PD et PF respectivement.

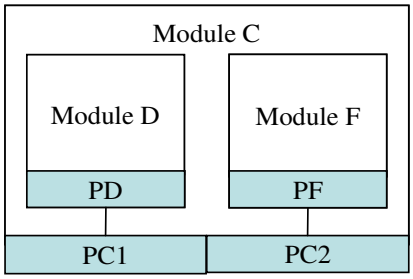


Figure 2-19 Représentation graphique d'un module composé

```
SC_MODULE (C)
{
sc_in<int> PC1;
sc_out<int> PC2;


/* déclaration des modules qui composent C*/
D *M_D;
F *M_F;

/* constructeur du module C
SC_CTOR ( C )
{
/* instanciation des modules*/

M_D= new D ("Module D");
M_F= new F ("Module F");

/* Mise en correspondance des ports PD et PF*/

M_D->PD(PC1);
M_F->PF(PC2);
};
}
```

Figure 2-20 Exemple illustrant la composition d'un module avec le langage SystemC

Pour une meilleure compréhension, les concepts de base sont illustrés dans la Figure 2-21 sur un exemple simple. Le système complet est représenté comme un module qui interagit avec l'extérieur *via* son interface modélisée par les ports logiques élémentaires P_{1S} et P_{2S}. Il est composé de deux modules A et B. Chaque module est décrit par un comportement composé d'opérations de calcul et de communication. Le module A est élémentaire, son comportement est décrit par deux tâches. Le module B est hiérarchique. Il contient deux sous modules (B1 et B2). Les différents modules du système communiquent à travers les ports logiques de leurs interfaces et sont liés par des interconnexions logiques (dans la figure, les modules A et B communiquent *via* deux interconnexions, à travers les ports P_{1A} et P_{1B} et les ports P_{2A} et P_{2B}).

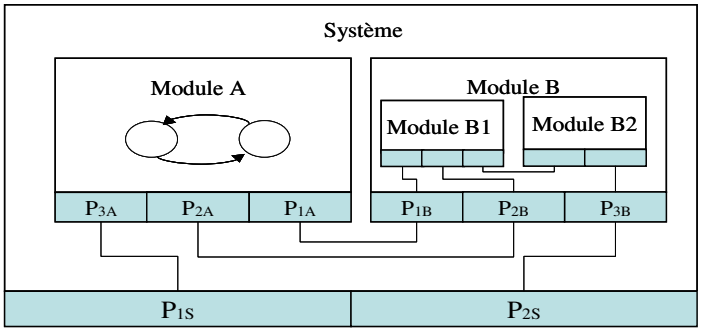


Figure 2-21 Spécification d'un système global

2.3 Modèle de calcul vs modèle d'exécution

L'un des problèmes principaux lors de l'étude des systèmes est de pouvoir séparer « ce que fait un système » de la « manière dont il le fait » (implémentation). Ces deux concepts seront appelés « modèle de calcul » et « modèle d'exécution ». Dans ce paragraphe, nous nous intéresserons à ces deux concepts de modélisation des systèmes.

Les modèles de calcul servent essentiellement à fournir une description ou une représentation des systèmes ; par exemple décrire le fonctionnement d'un filtre à travers sa description mathématique en utilisant les équations différentielles. Cependant, le modèle d'exécution représente une implémentation du modèle de calcul sur un environnement permettant de le réaliser. Pour ce même filtre, sa réalisation peut être matérielle en utilisant des registres, logicielle ou fonctionnelle (ceci sera détaillé dans le **Chapitre 3**).

Ces deux concepts ont fait l'objet de plusieurs études. Par exemple dans l'étude effectuée par [Sim98], le modèle de calcul représente un haut niveau d'abstraction plus haut que celui d'une architecture et d'un langage de programmation. Il est défini par trois concepts (1) les objets de base (données, opérations), (2) la sémantique de composition des opérations (procédural ou déclaratif) et (3) l'ordre d'exécution (dirigé par les données ou par le contrôle). Cette définition permet de présenter implicitement la manière avec laquelle le système réalise ces fonctionnalités [Sim98].

Une autre étude sur les modèles de calcul a été faite par [Lee99]. Elle définit le modèle de calcul comme un modèle formel. Ce modèle se base sur la définition des différents composants et la mise en évidence de leur interaction. Cette interaction représente le style de communication pouvant être établi entre les modules (asynchrone, synchrone, en temps continu, par suite d'événements discrets, etc.).

Il existe d'autres définitions d'autres modèles de calcul [Liu01] [Sgr00] qui rejoignent les définitions précédentes.

Toutes ces définitions se rejoignent sur un lien d'exécution du comportement (à travers le modèle de calcul) sans se soucier de l'implémentation finale du système (à travers le modèle d'exécution). Ce fossé entre le modèle de calcul et le modèle d'exécution constitue une source d'erreurs pour la réalisation finale du système et une limitation pour l'exploration d'architecture.

Il est alors adéquat d'introduire une définition de modèles de calcul et d'exécution qui permettent de réduire l'espace entre l'implémentation et la modélisation. Ceci a été possible en se basant sur l'étude séparée du calcul (comportement) et de l'interconnexion aussi bien au niveau modèle de calcul qu'au niveau de modèle d'exécution.

Dans les sections suivantes, nous nous focaliserons sur l'étude des concepts de modèle de calcul en présentant des classifications pour chaque type de modèle. Les modèles d'exécution seront traités dans le Chapitre 3.

2.4 Modélisation d'un système : modèle de calcul

Le modèle de calcul d'un composant présente la manière dont un résultat est obtenu indépendamment des données et du temps manipulés. Il nous permet de déterminer la relation entre les valeurs d'entrées et les valeurs de sorties. Ceci est vu de l'extérieur comme une description du comportement [Jan03]. Pour mieux expliquer cette définition, nous nous sommes basés sur la définition d'un calcul élémentaire.

2.4.1 Calcul élémentaire

On définit un *calcul élémentaire* comme étant un ensemble de règles pour la transformation des données en utilisant des opérations mathématiques. Ce calcul élémentaire est défini par :

- Les entrées,
- Les sorties,
- Les fonctions de transformation qui peuvent être des fonctions combinatoires (sans états) ou des fonctions séquentielles (avec états). Ces fonctions peuvent être simples ou complexes selon la granularité du calcul.

Un *modèle de calcul* est alors un ensemble de règles de composition pour des calculs élémentaires (à différentes granularités) afin d'effectuer l'ensemble des transformations sur les objets (données).

Les modèles de calcul ainsi définis permettent de décrire le fonctionnement d'un système en spécifiant un ensemble d'objets, un ensemble d'opérations sur ces objets et leur ordre d'occurrence. Les objets sont définis à travers l'ensemble de leurs entrées/sorties alors que les opérations sur ces objets sont l'ensemble des fonctions de transformations. Les relations entre ces objets peuvent être explicites (explicitement spécifiées par l'utilisateur à travers des opérations) ou implicites (inhérentes aux fonctions de transformations).

La description de ces systèmes repose essentiellement sur des modèles dits modèles de base. Ces modèles se répartissent selon la manière par laquelle ils définissent l'ordre d'occurrence des calculs élémentaires et les entrelacements entre les ordres des différents calculs:

- Le modèle de calcul *dirigé par les données* généralement représenté par des graphes de flux de données ou les équations différentielles. Dans ce modèle l'ordre d'occurrence des opérations dépend de la disponibilité des données.

- Le modèle de calcul *dirigé par le contrôle* généralement représenté par des graphes de flux de contrôle ou des machines d'états finis. L'ordre d'occurrence des opérations dans ces modèles dépend essentiellement de l'état actuel du système.

De plus, il existe une représentation mixte de ces deux modèles (contrôle et données). Pour cela ils sont représentés par des graphes dits graphes de flux de contrôle et flux de données (CDFG : Control Data Flow Graph) [CDF05].

Cependant, ces modèles deviennent insuffisants pour la représentation des systèmes composés. Ceci est dû à la complexité de ces systèmes et à l'emploi de différents modèles en même temps. En faisant abstraction du calcul dans un module élémentaire, et en ne présentant que la composition d'un système, on introduit les modèles hiérarchiques et en particulier le modèle distribué afin de décrire des systèmes complexes. Ce modèle sera présenté plus en détail dans la section 2.4.4.

2.4.2 Les modèles de calcul de base

Ce paragraphe décrit les différents types de modèles de calcul de base les plus couramment utilisés. Chaque modèle présente des caractéristiques spécifiques comme les notations employées pour l'expression des opérations et l'expression de l'ordre d'exécution comme l'expression de la concurrence et du parallélisme.

2.4.2.1 Le modèle de flux de données

Un modèle dirigé par les données ou modèle flux de données (en anglais Dataflow: DF) est un modèle représentant un calcul où le séquencement des opérations est défini par la disponibilité des données [Ger93] [Naj99].

Plusieurs modèles de flux de données existent. Les plus employés sont les graphes de flux de données (Data Flow Graph : DFG) et les équations différentielles.

La caractéristique essentielle d'un modèle de flux de données est l'absence d'état global du système. Un modèle flux de données est équivalent à un circuit combinatoire [Ger93] [Mad97].

2.4.2.2 Le graphe de flux de données

Le graphe de flux de données est un modèle générique permettant la représentation des dépendances des données. Il est composé de nœuds et d'arcs. Les nœuds correspondent à des opérations contrôlées par les données, les arcs représentent les valeurs de ces données.

Ainsi les objets définis par ce modèle sont l'ensemble des nœuds qui agissent sur des données d'entrées pour fournir des données de sorties. Les relations entre ces objets sont implicitement décrites par l'ensemble des arcs reliant les nœuds.

Dans ce type de graphe [Jag96], il existe plusieurs variantes d'activation des nœuds. On distingue alors deux types de flux de données :

- Ceux orientés par les données (en anglais data-driven dataflow model) : ils sont caractérisés par la règle d'activation dictant qu'un nœud est activé si et seulement si toutes les données d'entrées sont disponibles.
- Ceux orientés par la demande (en anglais demand-driven dataflow model) : dans ces modèles, l'activation d'un nœud est réalisée seulement si on a besoin du nœud pour terminer un certain calcul.

A l'aide de cette présentation en graphe, les dépendances entre les données sont explicitées et ainsi la détermination de l'ordre d'occurrence des opérations et celle du parallélisme sont triviales.

L'intérêt des DFG est qu'ils sont basés sur un modèle formel permettant des raisonnements mathématiques complexes. Ainsi, on peut définir des transformations formelles sur des graphes de flux de données. Plusieurs fonctions de traitement du signal peuvent être représentées par des fonctions de données.

2.4.2.3 Les équations différentielles

Les équations différentielles sont une notation mathématique permettant d'exprimer les mêmes calculs que les DFG sous forme d'équations. Les opérations sont constituées par les fonctions et les opérations d'égalité. L'ordre d'exécution est aussi défini implicitement par les dépendances de données.

2.4.3 Les modèles de flux de contrôle

Dans le modèle de flux de contrôle, une relation de précédence définit l'ordre d'exécution des opérations. L'exécution d'une opération est conditionnée par l'exécution de l'opération précédente. Les modèles les plus utilisés pour représenter le modèle flux de contrôle sont les machines d'états finis et les graphes de flux de contrôle.

L'une des caractéristiques essentielles des modèles de contrôle est la notion d'état global et de variables globales du système. Une autre spécificité est qu'à tout instant du calcul, le modèle de contrôle est caractérisé par son état courant et l'état des données qu'il manipule [Hol02].

2.4.3.1 Le modèle des machines d'états finis FSM

Le modèle des machines d'états finis nommé souvent « FSM » (Finite States Machine) est un graphe orienté, dont les nœuds représentent les états du système et dont les arcs représentent les transitions [Gir99].

Plusieurs modèles de FSM peuvent être définis selon la nature des états et des transitions. Un état peut renfermer un calcul complexe et même contenir des états internes. Aux transitions, on associe des conditions qui peuvent être spécifiées aussi comme un calcul. Les transitions peuvent aussi réaliser des calculs complexes en plus des conditions.

On distingue un état initial qui présente le point de départ du calcul. Le changement d'état dépend des fonctions des transitions associées aux arcs et de l'état où la dernière opération a été effectuée.

Le modèle FSM est généralement utilisé pour modéliser les parties contrôles des systèmes.

2.4.3.2 Le graphe de flux de contrôle

Le graphe de flux de contrôle (CFG de l'anglais Control Flow Graph) permet de décrire un séquencement de calcul des opérations *via* la représentation d'un graphe orienté. Ce graphe possède un ensemble de nœuds et un ensemble d'arcs.

Les nœuds de ce modèle représentent les opérations. Les arcs spécifient les relations de précédence entre ces opérations et définissent le séquencement du calcul. L'ordre d'occurrence d'un nœud dépend essentiellement du calcul du nœud précédent. Ces arcs peuvent être associés à des conditions qui valident l'occurrence de l'opération suivante.

2.4.4 Composition des modèles de calcul

Pour maîtriser la complexité, la conception des systèmes est généralement décrite en utilisant la composition des modules. Chaque module peut être spécifié comme une entité indépendante et utiliser un modèle de calcul spécifique. La tendance actuelle est de définir des conceptions des modules indépendamment de leur environnement d'implémentation afin de permettre la réutilisation des mêmes modules dans des contextes différents.

Les modèles hiérarchiques permettent la description d'un système composé. On distingue deux types de hiérarchies :

- La hiérarchie comportementale ou de spécification : cette hiérarchie ne représente qu'une facilité syntaxique. Un objet de ce modèle peut cacher un modèle de calcul de base comme décrit ci-dessus. Comme exemple, on peut citer les FSM hiérarchiques [Ard99] [Lut00] [Mar91], le modèle flux de données hiérarchiques [Mar91]. Il est même possible d'imaginer de combiner les modules flux de données et les modèles flux de contrôle dans une même hiérarchie. Rien n'empêche d'avoir des opérations représentées par des flux de données dans un graphe de flux de contrôle. La principale restriction de ces modèles hybrides est la difficulté de décrire de manière *efficace* un calcul distribué entre plusieurs modules. Ainsi, si on utilise un modèle DF pour composer des sous-modules CF on aura du mal à raisonner mathématiquement sur l'ensemble du système à cause des boucles possibles dans les nœuds de contrôle. Si on utilise un modèle de contrôle pour composer des modules DF, on aura des difficultés à exprimer du parallélisme entre les différents modules DF. Toutes les expériences de ce type relatées par la

littérature finissent par réduire le modèle de l'ensemble du système à une FSM de base afin de le traiter.

- La hiérarchie par interconnexion : cette hiérarchie définit la composition comme des interconnexions entre les différents modules pour constituer un système. Elle consiste à abstraire les composants du système et à ne spécifier que l'ensemble des interactions de chaque module avec son environnement. Dans la suite de ce paragraphe, nous ne nous intéresserons qu'à ces modèles.

Un modèle distribué est défini comme une composition de plusieurs modules interconnectés (voir Figure 2-21)

Un modèle distribué comporte deux types d'objets : les modules et les interconnexions. Chaque objet possède des points de connexions comme défini dans la section 2.2.4. La relation de composition de base est la mise en correspondance des ports d'objets. Dans un modèle distribué, chaque module est indépendant et ne communique avec l'extérieur que *via* son interface. Les interconnexions permettent donc de coordonner les calculs des différents modules. Cette coordination peut aussi être formulé à l'aide d'un modèle de calcul.

Ainsi, le modèle de calcul d'un système distribué est défini par la composition de deux types de modèles :

- Le modèle de calcul des composants
- Le modèle de calcul des interconnexions.

La composition est généralement définitecomme le produit des machines représentant les modules et les interconnexions. Cette définition engendre une explosion du nombre d'états et rend quasi impossible tout raisonnement formel sur les systèmes distribués complexes. C'est pour cette raison qu'on définit des modèles d'exécution afin de pouvoir les visualiser et effectuer les analyses requises.

2.5 Modèle de calcul de l'interconnexion

L'interconnexion a été définie comme une mise en correspondance entre les ports (voir2.2.4). La fonction d'une interconnexion est de réaliser un transfert d'information entre modules qui peut être simple (ex. lecture/écriture) ou complexe (ex. au moyen d'un protocole de type "hand-shaking" ou TCP).

L'interconnexion réalise trois types de fonctions pour permettre la coordination des modules parallèles. Les trois fonctions sont le transfert de données, la synchronisation et assurer une qualité de service (QoS de l'anglais Quality of Service) bien déterminée. Ces trois fonctions sont détaillées par la suite.

2.5.1 Le transfert de données

Le transfert de données est l'acheminement des données d'une entité à une autre. Il peut être fait de deux manières : par la mémoire partagée ou par l'envoi de messages.

2.5.1.1 La mémoire partagée

Le concept de mémoire partagée permet à plusieurs modules d'accéder à un même segment de mémoire. C'est un moyen rapide pour échanger des données entre processus. Aucune duplication de données n'est nécessaire, les données sont simplement placées à un endroit accessible par les différents processus.

La mémoire peut être accédée par plusieurs processus. Cependant, ce modèle peut nécessiter une synchronisation complexe pour assurer la cohérence de données (voir ci-dessous).

2.5.1.2 L'envoi de message

Dans ce cas, le transfert se fait par l'envoi de messages entre les processus en ayant chacun son espace mémoire privé. C'est un transfert plus lent que celui de la mémoire partagée mais capable d'éviter les problèmes de contention de mémoires.

Dans ce cas, la synchronisation est plus simple, par contre ce modèle génère des duplications de données et accroît la latence du système.

2.5.2 La synchronisation

La synchronisation permet la coordination entre les différents modules communicants afin d'éviter les conflits entraînant des pertes de données. La synchronisation peut se faire par un agent (une ressource) global tel que l'utilisation d'une horloge ou des ressources locales comme les sémaphores. Elle peut être synchrone ou asynchrone.

Il est à noter qu'un modèle de synchronisation peut être spécifique à une seule interconnexion. Ainsi un système contenant plusieurs interconnexions peut contenir plusieurs modèles de synchronisation différents.

2.5.2.1 Le modèle synchrone

Le terme synchrone est un terme dérivé du grec composé du terme « *syn* » qui signifie « avec », et du terme « *chronos* », qui signifie « temps ». C'est une description d'objets et d'événements qui sont coordonnés selon une abscisse temporelle.

Un modèle synchrone suppose l'existence d'un agent central synchronisant les modules communicants. Ce modèle est généralement caractérisé par un mode bloquant, c'est-à-dire que la communication impose une dépendance d'activité et des attentes entre les différents modules concernés [Syn05].

2.5.2.2 Le modèle asynchrone

Le terme asynchrone est un terme dérivé du grec composé du terme « a*syn* » qui signifie « sans », et du terme « *chronos* », qui signifie « temps ». C'est une description d'objets et d'événements, qui ne sont pas coordonnés dans le temps.

Un modèle asynchrone est défini par une communication qui n'influe pas sur l'exécution du comportement de certains modules. Par exemple un module peut éventuellement continuer son exécution même après l'envoi de messages (sans avoir la nécessité d'attendre un accusé de réception du récepteur). C'est un mode non bloquant, qui favorise la multi-tâche [Syn05].

2.5.3 La qualité de service

La qualité de service est une mesure, qui nous permet d'analyser les performances d'une interconnexion et d'en orienter le fonctionnement afin de garantir un certain nombre de contraintes inhérentes aux services proposés. Au niveau des interconnexions, elle est assurée par des mécanismes de contrôle qui peuvent être, par exemple, des techniques de routages et d'ordonnancement des différentes données transmises. Ces mécanismes peuvent être déterminés d'une manière statique ou dynamique [QoS99].

2.6 Modèle de calcul des systèmes hétérogènes

Définir un modèle de calcul d'un système hétérogène revient à définir l'ensemble des calculs et d'interconnexions. Cependant, jusqu'à présent il n'existe pas un modèle unique capable de traiter une telle description. Ceci est dû au fait que dans un système hétérogène on traite plusieurs types de modèles de calcul à la fois (élémentaires, hiérarchiques, etc.). Ainsi, la définition des modèles de calcul pour les systèmes hétérogènes nécessite une évaluation de toutes les opérations relatives à tout le système. Ceci étant fastidieux à obtenir (en raison de la manipulation de différents concepts et de différents langages), le choix d'un modèle unique pour ces systèmes n'est pas une solution optimale. La modélisation de tels systèmes vient alors de l'utilisation de la notion de modèle distribué (défini dans 2.4.4). Ce choix repose sur le fait que la définition d'un système hétérogène est une composition de modules différents interconnectés. Ainsi, on peut définir le modèle de calcul des systèmes hétérogènes comme étant un assemblage de différents modules ayant des modèles de calcul différents (voir 2.4.2) avec des interconnexions hétérogènes. Cependant, la sémantique du modèle ainsi présenté n'est définie qu'à travers les modèles d'exécution des différents éléments d'un système hétérogène.

La suite de la thèse fait l'étude des modèles d'exécution des systèmes hétérogènes à travers les modèles d'exécution des composants et ceux des interconnexions.

2.7 Conclusion

Dans ce chapitre, nous avons présenté une terminologie commune extraite de l'étude de différents langages de modélisation et de spécification. Cette terminologie couvre un certain nombre de concepts offerts par ces langages tels que le comportement, l'interface, l'interconnexion, la communication, la hiérarchie et la composition.

La deuxième partie était dédiée à l'importance de la séparation des notions de modèles d'exécution et de modèles de calcul pour parvenir à l'étude des modèles des systèmes hétérogènes. Dans cette partie, nous avons recensé les différents modèles de calcul des composants et des interconnexions. Finalement, nous avons mis l'accent sur la difficulté de modéliser un système hétérogène en présentant comme solution le fait de parvenir à étudier le modèle d'exécution.

Chapitre 3 Modèle d'exécution d'un système

SOMMAIRE

3.1 Introduction

Un système est un ensemble de sous-systèmes interconnectés. Chaque sous-système possède certaines caractéristiques (fonctionnalités et implémentations). L'étude des fonctionnalités du système à travers les modèles de calcul étudiés dans le chapitre 1 permet de définir certaines caractéristiques telles que le parallélisme, les types d'informations traitées, etc.

Cependant, ces caractéristiques restent insuffisantes pour déterminer l'exécution finale du système. En effet, l'implémentation en matériel ou en logiciel du sous-système influent sur ces caractéristiques. Par exemple, l'exécution parallèle d'un ensemble de calcul peut être implémenté directement en matériel sous la forme de circuits combinatoires alors qu'en logiciel, il sera nécessaire d'ajouter un système d'exploitation qui définira l'exécution parallèle des tâches logicielles à travers son ordonnanceur.

Pour étudier un système, il est nécessaire d'analyser son implémentation. L'implémentation définit le modèle d'exécution. Elle peut être logicielle, matérielle ou fonctionnelle. Ces modèles seront étudiés dans la section 3.2. Ces modèles peuvent être décrits à plusieurs niveaux d'abstraction. Les différents niveaux d'abstraction seront détaillés par la suite.

L'exécution des systèmes ne s'arrête pas au comportement, elle s'étend aussi à l'interconnexion entre les différents sous-systèmes. Par exemple, l'interconnexion en VHDL est définie par l'ensemble des signaux entre les différents sous-systèmes et la propagation des données à travers ces signaux. Par contre, dans le langage C, l'interconnexion est définie par l'appel de fonctions et de procédures entre les sous-systèmes. Cet appel sera interprété lors des phases de compilation et d'édition des liens pour effectuer le fonctionnement global. L'étude des exécutions des interconnexions sera détaillée dans la partie 3.3. Les implémentations des interconnexions sont définies à travers plusieurs niveaux d'abstraction. Certains langages d'exécution implémentant ces interconnexions sont présentés et classifiés dans la section 3.4.

3.2 Modèles d'exécution d'un système

Un modèle d'exécution décrit la manière avec laquelle un calcul est réalisé. Cette réalisation peut être de différentes natures : logicielle, matérielle, fonctionnelle ou mixte.

Un modèle d'exécution peut être vu comme une interprétation du modèle de calcul. Ainsi, à chaque modèle de calcul peut correspondre un ou plusieurs modèles d'exécution.

Un système peut être réalisé d'une manière homogène c'est-à-dire tout en matériel, logiciel, ou fonctionnel. Le modèle d'exécution est aussi caractérisé par un niveau d'abstraction qui définit la granularité des opérations de communication utilisées par le module. Il se trouve que l'unité de mesure de la granularité est différente selon le domaine de réalisation du module. La granularité dépend des techniques de raffinement utilisées pour le passage d'un niveau à un

autre. Ainsi pour le logiciel, on s'intéressera au niveau des opérations de calculs (instructions) et des opérations d'entrées/sorties. Les instructions permettent d'abstraire la machine sur laquelle sera exécutée le logiciel et les entrées/sorties (les interconnexions avec l'extérieur).

Ainsi, le raffinement du logiciel consiste à traduire les opérations de calcul en instructions plus fines et les opérations d'entrées/sorties en couches d'adaptation du logiciel à l'architecture. Pour le matériel, on s'intéressera à la représentation du temps (continu, cycle d'horloge, transaction, message, service) car le raffinement du matériel consiste à définir une machine temporelle pour exécuter la fonction. Pour les modules fonctionnels, l'abstraction concerne la représentation des données (types abstraits, bits) et le mode d'interaction entre les fonctions.

3.2.1 Modèle d'exécution des composants

Le modèle d'exécution des composants présente la réalisation effective de la fonctionnalité d'un module. Cette réalisation peut être de trois types : logicielle, matérielle ou fonctionnelle.

En se basant sur la séparation de la partie calcul et de la partie communication, on définit le modèle d'exécution d'un composant comme suit :

- Une partie qui réalise le calcul appelé exécution du calcul de base. L'exécution du calcul est la mise en œuvre du modèle de calcul par un moteur bien adapté.
- Une partie pour la communication. Elle est nommée « interface de communication ». Elle prend en charge la communication du composant (module) avec l'extérieur. Cette interface est définie à travers l'ensemble des services de communications (offerts ou requis). Pour chaque modèle d'exécution, on distingue différents types d'interfaces de communication.

La figure suivante illustre un modèle d'exécution d'un composant.

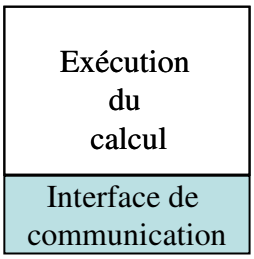


Figure 3-1 Modèle d'exécution d'un composant

Dans la suite, nous présentons les différentes mises en oeuvre des modèles d'exécution des composants ainsi que leurs différentes caractéristiques.

3.2.1.1 Modèle d'exécution logiciel

L'exécution du comportement, pour le logiciel, se fait par interprétation d'un ensemble d'instructions. Le modèle d'exécution logiciel fait appel à une plateforme matérielle qui interprète un programme (un ensemble d'instructions).

La spécification de la plateforme matérielle dépend de la granularité avec laquelle l'interprétation est accomplie. Cette granularité concerne l'abstraction du calcul (les opérations sur les données). On définit alors quatre niveaux de granularité : le niveau service (ou application), le niveau système d'exploitation ou « OS » (de l'anglais Operating System), le niveau architecture générique ou « HAL » (de l'anglais Hardware Abstraction Layer) et le niveau jeu d'instruction ou « ISA » (de l'anglais Instruction Set Architecture) [Tan99].

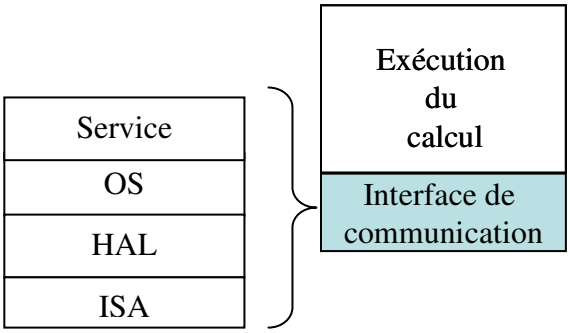


Figure 3-2 Niveaux d'abstraction de l'interface de communication d'un modèle d'exécution logiciel

Pour chaque niveau d'abstraction les détails d'implémentation sont plus ou moins explicites.

- Niveau service.

  Le niveau service (ou application) est défini comme étant une interprétation abstraite de la communication logicielle (soit entre les différentes applications soit entre le module et le reste du système). Le programmeur à ce niveau fait abstraction de la machine sur laquelle son programme va être exécuté puisqu'à ce niveau la spécification de la communication peut être indépendante de la structure des machines sur laquelle l'exécution du logiciel va s'effectuer réellement. Les détails d'implémentation de la communication tels que la synchronisation et le parallélisme peuvent être implicites dans les API offertes à ce niveau.

  La description du modèle peut être faite en utilisant des langages exécutables de modélisation ou de programmation. L'efficacité de l'exécution dépend alors de la puissance du compilateur ou de l'interpréteur associé. CORBA [Gei97], DCOM [Dco05] sont des exemples de description typiques de ce niveau.

- Niveau OS

A ce niveau l'API est limitée aux services fournis par le système d'exploitation donné. L'utilisation d'un système d'exploitation rend la synchronisation entre les tâches explicite. Les protocoles de communication entre les tâches et l'extérieur seront restreints par le système d'exploitation donné. Cependant, l'implémentation des pilotes (topologie des canaux, etc.) reste abstraite. Un exemple d'API de niveau OS est les API de « thread POSIX » [But97].

- Niveau HAL

L'interface de communication, du modèle d'exécution logiciel, au niveau HAL est vue comme un ensemble de services de communication permettant d'abstraire les accès physiques au matériel. L'API HAL contient des primitives pour abstraire les fonctions de base d'accès au matériel telles que l'initialisation (en anglais BOOT), le changement de contexte ou les entrées/sorties de base et accès aux périphériques. Comme exemple on peut citer les API HAL de RISC OS [HAL02].

- Niveau ISA

Le niveau ISA correspond à la spécification du logiciel en un langage assembleur relatif à un processeur cible. Il s'agit d'une spécification explicite de tous les détails de la communication (transfert de registre, interruption, etc.). En effet, à ce niveau, tous les détails du matériel sont fixés. L'implémentation des fonctionnalités du matériel est alors connue.

3.2.1.2 Modèle d'exécution matériel

Le modèle d'exécution matériel est un modèle où l'interprétation des fonctions de transformation n'est pas nécessaire pour réaliser une spécification donnée. C'est une transposition directe (en anglais mapping) d'une spécification sur une configuration matérielle.

L'exécution du comportement d'un module matériel correspond à la réalisation du calcul. C'est une exécution câblée c'est-à-dire, par exemple, si nous voulons exécuter l'opération d'addition (a+b), en matériel, on simule le circuit correspondant à l'addition.

Pour le matériel, la notion de niveaux d'abstraction est souvent basée sur l'abstraction temporelle [Bur03] [Con03].

On peut distinguer alors les différents niveaux temporels suivantes : (1) le niveau physique, (2) le niveau RTL (de l'anglais Register Transfer Level), et 2) le niveau TLM (de l'anglais Transaction Level Modeling) (voir ci-dessous).

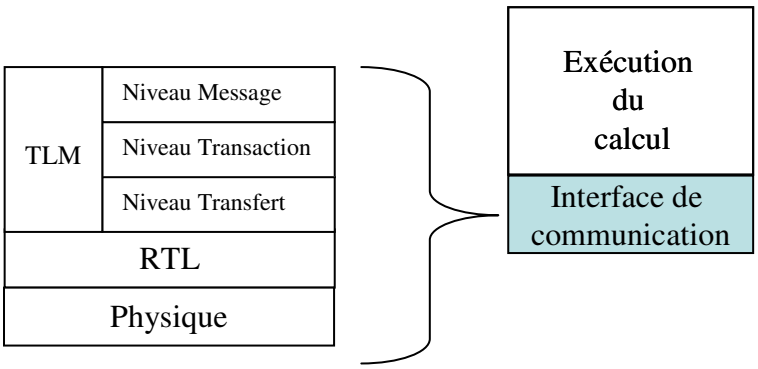


Figure 3-3 Niveaux d'abstraction de l'interface de communication du modèle d'exécution matériel.

Le temps peut être modélisé selon plusieurs niveaux d'abstraction. Ces niveaux sont :

- Le niveau physique : le temps, vu par la simulation au niveau électrique, est vu comme une valeur réelle. La modélisation se fait à base de transistors et le temps est considéré comme étant une unité continue indivisible.
- Le niveau « transfert au niveau registre » (Register Transfer Level noté « RTL ») : l'unité de temps est une unité entière (ex .cycle d'horloge). Cependant, on associe à ce niveau les délais de communication au niveau portes logiques et les données échangées sont définies au niveau bit. Ainsi, le temps de simulation est plus rapide que celui du niveau physique.
- Le niveau transaction (Transaction Level) : à ce niveau, le modèle est indépendant des détails de bas niveau tels que la taille des données sur le bus ou la nature des protocoles utilisés. Ceci est réalisé en faisant abstraction de certains paramètres tels que le type de données et le temps des échanges effectués (par exemple en utilisant une horloge logique). Selon ces paramètres, la décomposition en plusieurs sous-niveaux est réalisée : le niveau transfert, le niveau transaction et le niveau message. La notion de temps devient une notion d'ordre des transactions. Cet ordre peut être défini selon plusieurs paramètres :
    - Le niveau transfert (Transfer Layer) : l'unité de temps est toujours le cycle d'horloge mais certaines informations restent cachées par exemple, la description de l'organisation mémoire (« memory mapping ») pour un processeur. Les données transférées sont abstraites en une représentation plus abstraite qu'un vecteur de bits (par exemple « entier »).
    - Le niveau transaction (Transaction layer) : l'unité de temps correspond à une transaction du protocole de communication utilisé, l'abstraction concerne les détails des échanges entre, par exemple,

le bus de communication et la mémoire. On se base sur certains signaux de contrôle (relatifs aux protocoles utilisés) pour exprimer l'ordre des transactions. Une transaction peut correspondre à plusieurs cycles d'horloge. Par exemple, lecture d'un mot de taille quatre en un cycle ou en quatre cycles successifs.

- Le niveau message (Message layer) : il s'agit d'une abstraction de la structure de l'interconnexion entre les blocs matériels : un protocole abstrait de communication est utilisé. A ce niveau, on se base sur le principe de causalité pour la vérification du transfert. Ce principe se résume ainsi « on ne reçoit jamais un message non envoyé ».

3.2.1.3 Modèle d'exécution fonctionnel

Le modèle fonctionnel est un modèle indépendant de la réalisation finale du composant (matérielle ou logicielle). Le modèle fonctionnel peut être comparé à un modèle matériel simulé ou à un modèle logiciel exécuté en natif sur une station de travail. Dans un modèle fonctionnel ainsi défini, on ne s'intéresse qu'aux fonctionnalités du composant et qu'à son modèle de simulation.

Le modèle d'exécution fonctionnel correspond à une exécution virtuelle d'un système en utilisant les langages de modélisation exécutables et leurs environnements de simulation. L'exécution du comportement se fait grâce aux propriétés relatives au langage de modélisation du composant. Ces propriétés sont relatives à la syntaxe et à la sémantique du langage correspondant.

L'interface de communication est définie comme un ensemble de services permettant la communication entre différents modules. Ces services sont généralement exprimés par les primitives du langage de description utilisé.

Le modèle fonctionnel s'intéresse au traitement des données sans tenir compte de la réalisation finale (logicielle ou matérielle). Grâce à ce modèle, on ne montre que les dépendances et les relations entre les valeurs et les fonctions. Les niveaux d'abstraction d'un modèle fonctionnel dépendent du mode d'interactions entre les différentes fonctions. On définit alors quatre niveaux d'abstraction pour le modèle d'exécution fonctionnel.

- Niveau service : à ce niveau, le mode d'interaction entre le composant fonctionnel et l'extérieur s'effectue par des API de haut niveau. Ils définissent un ensemble de services requis et fournis de communication abstraite. Les services fournis sont l'ensemble des API du composant que les autres peuvent utiliser pour y accéder. Les services requis sont les appels de fonctions des autres composants. Les dépendances entre les valeurs et les fonctions s'observent à travers le mécanisme d'appel de ces

services. Ces appels ne sont pas dédiés à un composant prédéfini mais plutôt à un des composants capables de fournir ce service. Un des modèles d'exécution les plus utilisés pour ce niveau est le modèle CORBA [Gei97].

- Niveau message : à ce niveau le mode d'interaction entre les différents composants est plus raffiné qu'au niveau service. Il est défini aussi par un ensemble d'API définissant des services d'envoi et de réception de message. La dépendance des données est décrite explicitement par l'intermédiaire de paramètres des services requis ou fournis par le composant. Par exemple, en utilisant le langage MPI, un composant spécifie quels sont les destinataires qui devront recevoir le message. Les langages les plus utilisés qui permettent une telle description sont SDL (System Description Language) [Sdl87] et MPI (Message Passing Interface) [Gro96].
- Niveau transaction : une interaction est définie par un mode particulier (par exemple l'envoi d'un message nécessite deux autres actions : une requête pour permettre l'envoi et un acquittement précisant la terminaison de l'envoi). Ce mode dépend des protocoles de communication entre les composants en question. A ce niveau, l'interface est définie comme un ensemble de ports logiques (voir Chapitre 1). La dépendance des données est définie alors par ces protocoles. Les langages dans lesquels l'interface peut être décrite à ce niveau sont CSP [Sch99], SystemC comportemental [SyC02], VHDL comportemental [Vhd93], Cossap [Cos97] et StateCharts [Har87] [Lut00].
- Niveau RTL (Register Transfer Level) : à ce niveau, le mode d'interaction est ordonné par une unité temporelle définie par le langage utilisé. L'interface est définie comme un ensemble de ports physiques (chapitre 1). Les dépendances de données sont indiquées par l'émission ou la réception des signaux *via* les ports. Les données manipulées sont considérées à un niveau bas (bits). Les langages permettant la description d'un composant à ce niveau sont les langages assembleur, SystemC au niveau RTL, VHDL RTL [Vhd93], etc.

La figure suivante représente le modèle d'exécution fonctionnel.

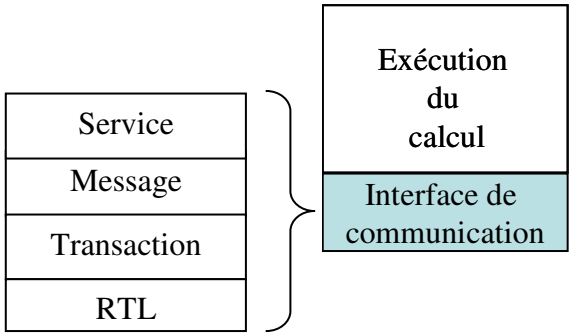


Figure 3-4 Niveaux d'abstraction de l'interface de communication d'un modèle d'exécution fonctionnel

3.3 Modèle d'exécution des interconnexions d'un système

Cette partie de notre étude concerne les systèmes distribués. Ces systèmes sont un ensemble de composants interconnectés. Le modèle d'exécution des interconnexions permet de définir la manière avec laquelle la communication est interprétée (exécutée). C'est la manière dont on peut réaliser les trois fonctions du modèle de calcul (voir chapitre 2), à savoir le transfert des données, la gestion du conflit réalisée par la synchronisation entre les différents modules et le contrôle de la qualité de service par l'utilisation de différentes politiques d'ordonnancement.

Le modèle d'exécution d'une interconnexion ainsi défini peut être l'interprétation d'un simple fil physique ne possédant pas de comportement et une interface se résumant aux extrémités du fil ou pouvant être plus complexe (par exemple un bus) ayant un comportement complexe et une interface particulière.

Les trois fonctionnalités décrites dans le Chapitre 2 peuvent aussi être vues à différents niveaux d'abstraction. Ces niveaux sont déterminés à partir de certains critères. Pour le transfert de données, le critère est le type du média ainsi que les données transportées. A ce stade, on note quatre niveaux d'abstraction : le niveau service, le niveau transaction, le niveau transfert et le niveau physique. Pour la synchronisation, selon le modèle du temps utilisé on a différents types de synchronisation (l'utilisation d'une instance d'horloge ou le principe de causalité). Pour la fonction de contrôle, les niveaux sont étroitement liés au niveau du transfert de données. Car selon le média et les données utilisés, on peut avoir un contrôle simple ou complexe. Ainsi, les niveaux du contrôle seront le niveau service, le niveau transaction, le niveau transfert et le niveau physique.

Les trois fonctionnalités sont étroitement liées ce qui réduit les combinaisons des niveaux d'abstraction entre elles.

Dans la suite, on détaillera pour chacune des fonctions les niveaux d'abstraction.

3.3.1 Les niveaux d'abstraction de transfert de données

L'abstraction du transfert des données se base sur deux critères : le type de média de transmission et le type des données [Nic01]. On distingue alors quatre niveaux d'abstraction :

3.3.1.1 Le niveau service

L'interconnexion au niveau service est dite interconnexion abstraite (voir chapitre 1). Le média de communication est défini à travers le mode d'interaction entre les différents composants. Pour ce niveau, le mode d'interaction est réalisé par des appels de services (fonctions). Le média est défini en tant que bus abstrait pour la communication (par exemple le bus ORB (Object Request Broker) de CORBA [Cor04]). Les types de données transportées

(transmises) ne sont pas fixés. Ils sont définis en tant que types abstraits (par exemple le type « CORBA_INT » qui pourra être l'entier de C++ ou l'entier de VHDL).

3.3.1.2 Le niveau transaction

Le média de l'interconnexion est un réseau logique. Un réseau logique est défini comme une abstraction d'une interconnexion physique (ensemble de fils ou bus physique). C'est un regroupement logique de connexions physiques. Par exemple, une interconnexion de type FIFO est une abstraction de trois connexions de fils physiques (deux signaux de contrôle et un de données). Le mode d'interaction est défini comme une transaction lorsque l'échange des données est assuré *via* le média. Les types de données transmises sont scalaires ou complexes (structure de données).

3.3.1.3 Le niveau transfert

A ce niveau, le média correspond à un bus physique dont certains détails ont été abstraits. Par exemple un bus AMBA, décrit au niveau transfert, ne modélise pas le temps de propagation physique des données mais une transaction qui peut correspondre à plusieurs cycles. Le type des données transmises peut être des bits ou vecteurs de bits.

3.3.1.4 Le niveau physique

Le média est défini comme un ensemble de *fils* et *des bus physiques*. A ce niveau toutes les données sont des bits ayant une représentation fixe. Le média de communication est défini via l'ensemble de fils physiques. Au niveau physique, la granularité est le temps discret (entier multiple d'une unité de base) ou même le temps continu.

3.3.2 Les modes de synchronisation

La synchronisation au niveau de l'interconnexion définit comment l'échange des données est assuré. On définit la synchronisation selon deux modes de temps physique ou logique.

La synchronisation peut être définie :

- En se basant sur une unité temporelle définie dans le modèle utilisé, c'est-à-dire à chaque changement de cette unité (ex. front montant ou descendant d'une horloge), les transferts des données sont assurés entre les composants. Dans ce cas on parle d'une synchronisation à temps physique.
- En se basant sur l'envoi et la réception des données, auquel cas, une horloge logique est utilisée. C'est une horloge définie par le « principe de causalité ». Cependant dans ce cas, on parle de mode synchrone lorsque l'émetteur et le récepteur doivent se synchroniser pour l'envoi ou l'émission des données (par exemple une communication par « sockets » [Soc05]). Dans ce cas, il s'agit de mode synchrone temps logique. Dans les autres cas, l'émetteur et le récepteur peuvent ne pas être synchronisés. L'un peut émettre sans que l'autre soit en mesure de recevoir le

message. Dans ce cas, on parle de transfert asynchrone. Par exemple, la communication par des FIFO non bloquantes.

3.3.3 Les niveaux d'abstraction du contrôle des interconnexions

Le contrôle est lié au comportement (ou protocole). Il inclut des fonctionnalités d'ordonnancement et d'arbitrage (certaines fonctions de résolution). L'ordonnancement est un ensemble de décisions ayant pour but d'établir un ordre d'application des actions, avant leur exécution [Dic05]. L'arbitrage est une opération consistant à faire un choix pour répondre à des demandes conflictuelles lors de l'utilisation d'une ressource par plusieurs systèmes. Les niveaux d'abstraction du contrôle sont déterminés à travers la modélisation de ces fonctionnalités.

3.3.3.1 Le niveau service

Le contrôle des interconnexions au niveau service est transparent pour l'utilisateur. Cependant, pour assurer un fonctionnement cohérent et correct, les fonctionnalités d'arbitrage et d'ordonnancement sont complexes. En effet, pour permettre cette transparence à l'utilisateur la modélisation de l'interconnexion doit tenir compte de l'acheminement de l'information entre les composants avec des dépendances de données non spécifiques. Par exemple, un composant CORBA requiert un service S sans connaître le fournisseur de ce service. Le composant CORBA demande à travers l'interconnexion « ORB » le service aux autres composants du système. L'ORB prend en charge alors l'acheminement de la requête vers le composant, qui la satisfait et retourne le résultat au composant fournissant S [Gei97]. A ce niveau, l'ordonnancement est la recherche du composant fournissant le service et bien sûr l'arbitrage de l'accès aux différents composants dans le cas d'un transfert multiple.

3.3.3.2 Le niveau transaction

Le contrôle des interconnexions au niveau transaction est défini par l'ordonnancement des différents acheminements des données (émission et réception) entre les composants. A ce niveau, la dépendance des données entre les composants est fixée par le protocole que le média implémente (par exemple, FIFO bloquante). L'ordonnancement, dans ce cas, a pour but d'assurer les transactions définies par le protocole en question. Par exemple, pour la FIFO bloquante, l'accès est permis à tous les composants qui lui sont interconnectés tant qu'elle n'est pas vide.

3.3.3.3 Le niveau transfert

Le contrôle des interconnexions au niveau transfert correspond à l'ordonnancement du transfert des données selon le mode de fonctionnement du bus d'interconnexion par l'intermédiaire de contrôleurs de bus (par exemple un contrôleur de bus AMBA [Ahb99]).

3.3.3.4 Le niveau physique

Le contrôle des interconnexions au niveau physique est assuré par des fonctions de résolution de signaux physiques. A ce niveau, toutes les interconnexions et tous les acheminements sont définis par une topologie physique.

3.3.4 Récapitulatif

Le tableau suivant représente les différentes caractéristiques de l'interconnexion à travers les différents niveaux d'abstraction.

Niveau d'abstraction	Média de comm.	Comportement	Type de Données
Niveau service	Média abstrait (Ex.ORB)	Appels de fonctions	# non interprété
Niveau transaction	Canaux logiques (Ex.FIFO)	protocoles abstraits	Scalaires/Structures de données
Niveau transfert	Canaux explicites (Ex. Bus)	protocoles fixés	Bits ou vecteurs de bits
Niveau physique	Fils Physiques	Transfert de signaux	Signaux

Tableau 1 Les caractéristiques de base de l'interconnexion à travers les nivaux d'abstraction

L'étude de certains langages de modélisation et de spécification nous a permis de réaliser une classification selon la modélisation de ces langages de l'interconnexion à travers les différents niveaux d'abstraction étudiés ci-dessus. Les langages de modélisation définissent les interconnexions comme une partie de leur sémantique d'exécution. Certains langages regroupent une définition unique de l'interconnexion, par exemple, simplement un appel de fonctions distantes pour CORBA.

Cependant, certains langages permettent de définir plusieurs niveaux d'abstraction pour l'interconnexion, par exemple la plupart des langages de modélisation matérielle tels que SystemC et VHDL. SystemC, par exemple, peut modéliser une interconnexion au niveau physique tel qu'un fil physique (à l'aide de la primitive sc_signal) ou une interconnexion au niveau transaction telle que l'utilisation d'une interconnexion de type FIFO (grâce à la primitive sc_fifo). Ces caractéristiques peuvent être définies grâce aux modèles d'exécution des interconnexions sur lesquels se basent les langages. Ces modèles seront traités dans la section 3.4.1.

Les langages de modélisation matérielle (Hardware Description Language, noté HDL) se basent sur un modèle pouvant balayer l'espace de réalisation à travers tous les niveaux

d'abstraction. Pour la modélisation de cette caractéristique (possibilité d'avoir plusieurs réalisations à travers les différents niveaux d'abstraction), dans la Figure 3-5, une ellipse est représentée pour décrire le balayage des HDL à travers les différents niveaux d'abstraction.

La réalisation des interconnexions est vue selon trois axes correspondant aux trois fonctionnalités : la synchronisation, le contrôle des interconnexions et le transfert des données. La figure suivante présente les différents niveaux d'abstraction pour les différentes fonctionnalités et un essai de distribution de quelques langages dans l'espace des réalisations.

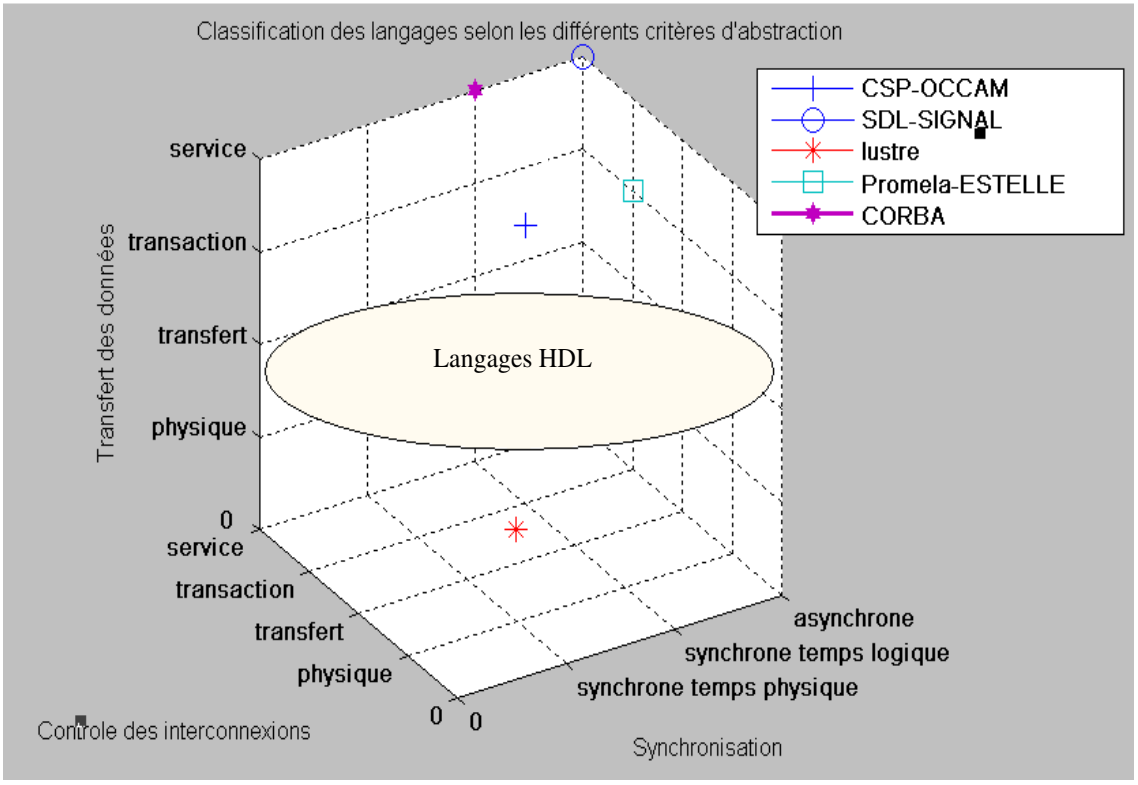


Figure 3-5 Classification de certains langages selon les niveaux d'abstraction

Cependant, et derrière chaque langage, un modèle d'exécution des interconnexions est spécifié. Dans la partie suivante, on s'intéressera à ces modèles et on associera pour chaque langage le modèle correspondant.

3.4 Classification des modèles d'exécution des interconnexions existants

La description des interconnexions est réalisée à travers des modèles dits modèles d'exécution. Ces modèles peuvent être présentés à différents niveaux d'abstraction selon la description plus ou moins explicite de certains détails (propriétés) de communication (canal, protocoles, etc.). Ces modèles peuvent être classifiés, en plus, par la description de la synchronisation (synchrone, ou asynchrone) pour la réalisation de la communication.

Dans la section suivante, un recensement des modèles d'exécution des interconnexions les plus utilisés est présenté.

3.4.1 Modèles d'exécution des interconnexions existants

Plusieurs modèles d'exécution existent pour la réalisation des interconnexions. Ce paragraphe décrit quelques modèles (généralement les plus utilisés). On relève les modèles suivants :

- Modèles d'invocation à distance (ID).
- Modèles de passage de messages synchrones (SMP).
- Modèles de passage de messages asynchrones (AMP).
- Modèles synchrones.
- Modèles à événement discret (DE).

3.4.1.1 Modèle par invocation à distance : ID

Le modèle de communication par invocation à distance se base sur l'appel des services distants. Il existe plusieurs types de modèles par invocation à distance. On peut citer les modèles client serveur, RPC (Remote Procedure Call), etc.

Le modèle client serveur est un modèle qui décrit la relation entre deux programmes décrits par des langages différents. Un des programmes est nommé client; c'est le programme qui demande un service à un autre programme. L'autre programme est nommé serveur ; c'est le programme qui fourni le service au client. Ce modèle fournit un moyen commode pour l'interconnexion des programmes distribués.

Le modèle RPC a été défini par Birell et Nelson [Rpc05]. Il représente un protocole qui demande l'exécution d'une procédure distante. Par exemple, on peut citer le ONC RPC[1] et CORBA comme protocoles basés sur des RPC.

On peut trouver deux types de RPC : synchrone et asynchrone.

- Modèle RPC synchrone (c'est dans le cas du modèle client serveur) : dans ce modèle le client est bloqué en attente d'une réponse du serveur (comme un appel de procédure locale). Ce modèle est facile à comprendre. De plus, il permet la détection des erreurs facilement, d'autant plus qu'il n'est pas nécessaire de stocker l'information.
- Modèle RPC asynchrone (ce cas est identifié dans le modèle producteur consommateur) : dans ce modèle le client n'est pas bloqué, mais il existe un test continu sur la réponse du serveur. Ce modèle nécessite l'utilisation d'un élément de mémorisation pour le stockage de la réponse relative à la requête. Cependant, Il nécessite que la communication soit fiable et que les canaux soient sans pertes. Un système qui utilise ce type de modèle est X-11 ou même l'impression d'un fichier.

[1] Open Network Computing Remote Procedure

3.4.1.2 Modèles de passage de messages synchrones : (SMP : Synchronous Message Passing)

Le modèle SMP se base sur la synchronisation entre les composants à travers l'envoi des messages. Dans ce modèle, la synchronisation, les différentes conversions (de formats, de données, etc.) et la gestion de contrôle sont explicites. La communication se fait alors de façon atomique. Les composants qui ne sont que des processus communiquent par des rendez-vous ce qui permettra de reconnaître un état global du système. Il est utilisé surtout pour les systèmes où il y a beaucoup de ressources partagées [Kai02].

Cependant il représente un couplage très fort entre les composants ce qui rend le maintien le déterminisme très difficile.

Il existe plusieurs langages qui peuvent le modéliser comme CSP [Sch99] et Occam [Lee99].

3.4.1.3 Modèles de passage de message asynchrone : (AMP : Asynchronous Message Passing)

Le modèle AMP invoque l'utilisation des réseaux de processus PN (Process Network) qui permet la représentation des comportements à flux de données (Data Flow) [Lee01] [Lee99].

La communication dans ce modèle se fait par l'envoi des messages à travers des canaux de communication. Il n'existe pas de dépendances temporelles entre les composants, ce qui ne permet pas un acquittement implicite entre les processus qui représentent les composants [Kai02].

L'implantation d'un tel modèle est aussi facile pour le matériel que pour le logiciel. Ceci est dû à la liberté du choix du renvoi des ordres. Il est efficace pour les systèmes de traitement de signaux. Les langages ayant la possibilité de le modéliser sont SDL [Sdl87] Promela [Hol90], Estelle [Est05].

3.4.1.4 Modèles synchrones

Les systèmes synchrones s'appuient sur les notions d'horloge globale, de diffusion instantanée d'informations, de parallélisme déterministe et de préemption pour fournir un modèle de programmation cohérent et adapté.

Ce modèle suppose que tous les modules sont cadencés par la même horloge (physique et/ou logique). Chaque signal peut être accompagné par un signal d'horloge (explicitement ou conceptuellement) mais cette horloge possède un sens relatif par rapport aux autres horloges des autres signaux, ce qui nous permettra d'établir un ordre total du temps.

A chaque instant, les systèmes synchrones exécutent une transition avec les signaux mémorisés. Et pour un état donné, deux phases sont exécutées; la première consiste en la mémorisation des signaux mémorisés et la deuxième est l'exécution d'une transition pour tous les processus avec les signaux mémorisés. L'interaction entre les composants, qui ne sont que les relations entre les entrées et les sorties, se fait alors à travers les données alignées par un

front d'horloge. Ce modèle est prévu surtout pour les applications concurrentes à contrôle logique complexe [Lee99]. La communication entre les différents modules se fait à travers des valeurs des données qui sont alignées par un front d'horloge.

Les langages ayant la possibilité de les modéliser sont Esterel et SyncCharts [And98].

3.4.1.5 Modèles à événement discret (DE : Discret Event)

Le modèle à événements discret est un modèle de système pour lequel le temps et les composantes du vecteur d'états sont des variables discrètes. Il représente un modèle où le temps est totalement ordonné. Il permet la représentation explicite du temps. C'est pourquoi il est adéquat aussi bien à la simulation qu'à l'implémentation [Lee99].

La communication se fait par des événements localisés sur une même ligne temporelle de manière discrète à l'aide de primitives de lecture et d'écriture sur des signaux partagés. Les composants, lors de leur exécution consomment les événements d'entrées pour produire d'autres en sortie. Ces derniers généralement respectent la règle de causalité, c'est-à-dire, ils ne doivent pas être générés avant les événements d'entrées.

Le modèle à événements discrets est adopté surtout pour la spécification matérielle du fait qu'il est excellent pour exprimer la concurrence entre des composants matériels.

Les langages de modélisation de ce modèle sont VHDL, SystemC, etc.

3.4.1.6 Classification des modèles d'exécution de l'interconnexion

Dans ce paragraphe, les modèles d'exécution décrits ci-dessus sont classifiés selon les niveaux d'abstraction, décrits dans la section 3.3.1. Cette classification est résumée dans le Tableau 2.

Synchronisation / **Niveaux d'abstraction**	**Synchrone**	**Asynchrone**
Niveau Service	Invocation à distance synchrone	Invocation à distance asynchrone
Niveau Transaction	Passage de message synchrone	Passage de message asynchrone
Niveau Transfert/Physique	Synchrone	Evènement discret

Tableau 2 Classification des modèles d'exécution pour l'interconnexion selon les niveaux d'abstraction

Cette classification reste limitée. En effet, vu le nombre important des modèles utilisés pour la communication, il était difficile de tout étudier. Donc, nous nous sommes restreints à ces modèles.

3.5 Conclusion

Dans ce chapitre, nous avons présenté les différents modèles d'exécution des systèmes. Ces modèles peuvent être logiciel, matériel et fonctionnel. Chacun de ces modèles peut être décrit à plusieurs niveaux d'abstraction.

Nous avons présenté également les modèles d'exécution des interconnexions des systèmes. Ce sont des réalisations de la mise en correspondance des interfaces de communication des différents composants. Ces modèles sont définis aussi à travers plusieurs niveaux d'abstraction. Une classification des langages de modélisation a été réalisée en fonction des niveaux d'abstraction de l'interconnexion.

Une autre classification est réalisée pour la classification des modèles d'exécution existants des interconnexions selon le mode de synchronisation d'échange de données entre les composants.

Toutefois, la définition d'un modèle d'exécution global pour un système hétérogène n'est pas évidente. En effet, avoir un modèle global qui englobe tous les modèles adéquats des sous-systèmes et des modèles d'interconnexions nécessite de mettre en œuvre le principe d'interconnexion hétérogène de systèmes hétérogènes. Cela est dû à la manipulation de concepts et de propriétés différents. Il est plus intéressant de pouvoir réutiliser les concepts déjà existants. Ce principe sera détaillé dans le chapitre suivant.

Chapitre 4 Modèle d'exécution global des systèmes hétérogènes

SOMMAIRE

4.1 Introduction

Le modèle d'exécution d'un système permet de définir aussi bien sa réalisation que sa validation. Il permet l'interprétation du comportement et de l'interconnexion comme vu dans le chapitre précédent. L'interprétation peut être logicielle, matérielle ou fonctionnelle. La validation du système logiciel, matériel ou fonctionnel est effectuée par l'intermédiaire de simulateurs des langages, des émulateurs, des prototypes selon le niveau de description utilisé. La validation globale d'un système est nécessaire afin de vérifier ses différentes fonctionnalités tout au long du flot de conception. Toutefois, l'hétérogénéité des composants des systèmes embarqués rend la validation globale très difficile (50-80% du temps de conception [Kea99]). Cela nécessite **un modèle d'exécution global** des systèmes hétérogènes qui puisse accommoder différents protocoles de communication, différents niveaux d'abstraction ou langages de modélisation.

Plusieurs travaux portent sur la définition de 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 où elles maîtrisent seulement l'adaptation de certains niveaux d'abstraction et/ou protocoles de communication. De plus, la confusion entre les modèles de calcul et d'exécution a été un handicap pour certains des travaux antérieurs qui ont essayé d'étudier la composition des systèmes hétérogènes. Ils se sont heurtés à la diversité des modèles existants et ont souvent abouti à des hypothèses trop réductrices pour être utiles dans le cas des systèmes complexes. Certains de ces modèles seront présentés dans la section 4.2.1.

Très souvent aussi, les modèles d'exécution et de validation actuels (par ex : simulateurs, émulateurs, compilateurs, etc.) sont validés et devenus standards. Dans ces cas, il serait dommage de ne pas les réutiliser.

L'objectif de ce travail est de généraliser le concept de modèle global d'exécution par composition pour les systèmes hétérogènes embarqués. Ce modèle pourra être utilisé tout au long du flot de conception de ces systèmes. L'intérêt de ce modèle est de supporter les interconnexions hétérogènes et de réutiliser l'existant. Le concept d'interconnexion hétérogène a été généralisé et défini grâce à la séparation entre modèle de calcul et modèle d'exécution.

Ce chapitre est organisé en quatre parties. Dans la première partie, le modèle générique d'exécution des systèmes hétérogènes par composition est défini ainsi que ses différents éléments. Certains travaux existants pour la génération des modèles exécutables seront également présentés, dans cette partie. La deuxième partie est consacrée à la présentation du modèle de spécification pour la génération automatique des modèles d'exécution. C'est le modèle à base de composants virtuels. Certains travaux ont utilisé ce concept. Ils seront détaillés

dans la troisième partie. La dernière partie est consacrée à la présentation d'un flot automatique pour la génération des modèles d'exécution globaux des systèmes hétérogènes. Dans cette partie, une architecture des différents éléments de modèles d'exécution est présentée.

4.2 Définition d'un modèle généralisé d'exécution global des systèmes hétérogènes

Dans cette section nous présenterons dans la première partie les travaux existants pour la définition de modèle d'exécution global des systèmes hétérogènes. Puis, nous donnerons la définition de notre modèle global et généralisé d'exécution de ces systèmes.

4.2.1 Etat de l'art sur la définition des modèles d'exécution globaux des systèmes hétérogènes

Dans la littérature, certains travaux ont essayé de définir un modèle d'exécution global pour les systèmes hétérogènes. Deux définitions sont distinguées [Cos99].

- Modèle d'exécution monomoteur : il est déterminé par des transformations des modèles de composants et ceux de l'interconnexion pour opérer sur un modèle unique (par exemple utiliser un langage exécutable unique) [Cal96] [Pas99].
- Modèle distribué : il est défini par l'exécution conjointe des différents composants et leurs interconnexions. C'est un modèle par composition des modèles existants [Cos99].

Dans notre cas, nous nous basons, pour la définition généralisée de notre modèle d'exécution, sur le modèle distribué. En effet, (1) il convient à la définition adoptée dans cette thèse pour les systèmes hétérogènes (comme étant des systèmes distribués) et (2) il permet la réutilisation de l'existant : pour les composants et les interconnexions du système, un modèle d'exécution approprié (standard) peut être employé.

Des études se sont basées sur ce modèle et ont essayé de définir un modèle généralisé d'exécution global pour les systèmes hétérogènes. Cependant ces définitions restent insuffisantes du fait que certaines études se limitent à l'étude des adaptations de certains concepts [Laj99] où le manque de généricité de certains modèles (modèle purement matériel, software ou fonctionnel) [SyC02] [Vhd93].

E.Lee [Lee99] est celui qui a fait l'étude la plus complète concernant la composition des systèmes hétérogènes. Cependant, dans cette étude, la confusion des modèles de calcul et celui d'exécution l'ont amené à choisir un modèle de composition basé sur un schéma de communication unique acceptable pour la conception d'une classe restreinte de systèmes.

Les travaux de G.Nicolescu [Nic03] présentent un modèle d'exécution distribué pour la validation des systèmes hétérogènes. Cependant, ce modèle se base sur la définition empirique (expérimentale) de ces différents éléments : modules, interfaces et bus de cosimulation. Les

parties traitées essentiellement sont la définition et la génération automatique de l'architecture des interfaces de cosimulation. Certains concepts nécessitent des définitions plus approfondies et généralisées.

Les travaux de recherche, dans cette thèse, complètent ceux de G.Nicolescu en définissant un modèle généralisé d'exécution basé sur l'assemblage des différents modèles d'exécution des différents éléments qui le constituent.

4.2.2 Définition du modèle généralisé d'exécution de systèmes hétérogènes

Nous définissons le modèle d'exécution généralisé des systèmes hétérogènes comme étant un assemblage de différents modèles d'exécution des différents éléments qui le constituent (composants et interconnexions). Pour chaque élément, on retient les spécificités et les caractéristiques des modèles appropriés (standard) pour chaque élément

Cependant, l'assemblage des différents modèles nécessitera la définition du concept de l'interconnexion hétérogène entre les différents modèles existants. La figure suivante présente la problématique de l'assemblage.

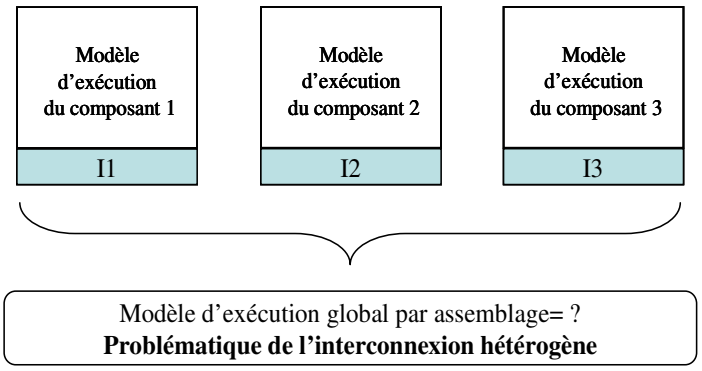


Figure 4-1 Problématique du modèle d'exécution global d'un système hétérogène

L'interconnexion hétérogène est définie par une mise en correspondance entre les différentes interfaces de communication des modèles d'exécution des composants.

Le modèle d'exécution global d'un système hétérogène est défini alors par :

- Les modèles d'exécution des composants vus dans le chapitre précédent.
- L'interprétation de l'interconnexion hétérogène : cette interprétation est définie par un ensemble de règles de composition du système global et l'ensemble des adaptations nécessaires entre les différents modèles pour réaliser leur communication. Elle est définie par deux éléments : les adaptateurs et l'environnement d'exécution.

Le modèle d'exécution global des systèmes hétérogènes est donné par la figure suivante.

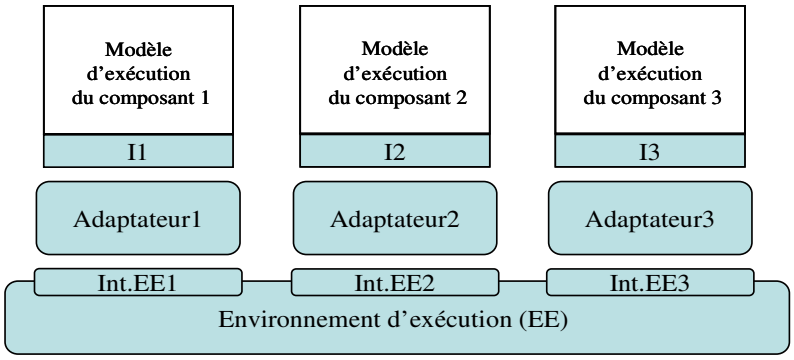


Figure 4-2 Modèle d'exécution global des systèmes hétérogènes

La généralité de ce modèle réside dans l'indépendance des définitions de ces éléments par rapport aux langages de modélisation et modèles d'exécution des composants. De plus, ils sont indépendants des niveaux d'abstraction dans lesquels les interfaces de communication des composants sont décrites.

Les différents éléments de ce modèle sont détaillés dans la section suivante en mettant l'accent sur leurs caractéristiques de généralité.

4.2.3 Eléments du modèle d'exécution des systèmes hétérogènes.

4.2.3.1 L'environnement d'exécution

L'environnement d'exécution est défini comme l'entité permettant l'interaction entre les différents modèles d'exécution des composants. Il est défini comme l'ensemble de règles de fonctionnement global du système et la sémantique des connexions.

L'environnement d'exécution définit une interface d'interaction avec chaque modèle d'exécution notée « Int.EEi », (i=1..3), dans la Figure 4-2. Cette interface est de même type que l'interface décrite dans le chapitre 1. Elle peut être un ensemble de services, de ports logiques ou physiques.

Cette définition peut s'appliquer à tous les modèles d'interaction entre les systèmes. En effet, l'environnement d'exécution peut être vu comme un bus de cosimulation [Nic03], un réseau de communication (par exemple OCCN [Occ05], ou même des fils physiques) ou même une abstraction d'un sous-système logiciel au niveau HAL [You05]. Pour chacun de ces environnements d'exécution, les interfaces peuvent être décrites par nos définitions.

4.2.3.2 Les adaptateurs

Les adaptateurs sont des entités passives (pas de comportement) ou actives (ayant un comportement) dont le contenu doit être capable d'adapter les différentes interactions entre l'environnement d'exécution et les composants d'une part et les composants entre eux d'autres part. Le rôle de ces adaptateurs est donc :

- L'adaptation de différents modèles d'exécution des composants (simulateurs) à l'environnement d'exécution pour garantir la transmission des informations entre eux.
- L'adaptation des différentes interconnexions (média, type des données et protocole de communication), voir chapitre 1, des composants pour permettre l'exécution des systèmes englobant des interconnexions hétérogènes.

Les éléments constituants un adaptateur sont définis par :

- L'interface interne relative au composant.
- L'interface externe relative à l'environnement d'exécution.
- Les éléments d'adaptation entre ces deux interfaces.

La généralité des adaptateurs pour notre modèle réside dans son architecture. Elle sera détaillée dans la section 4.5.3. Nous avons défini une architecture qui peut être utilisée quels que soient les modèles à adapter (logiciel, matériel ou fonctionnel).

Toutefois, la définition de ces éléments d'une manière *ad hoc* ne nous permet pas un gain pour réduire le temps et le coût de conception des systèmes hétérogènes. Il est intéressant de définir une méthodologie de génération automatique de tels modèles. Pour aboutir à cet objectif, nous avons défini un modèle de spécification des systèmes hétérogènes permettant de représenter des interconnexions hétérogènes. Ce modèle est le modèle à base de composants virtuels. Ce modèle sera détaillé dans la section 4.3. Il sera le point d'entrée d'un flot de génération de modèles d'exécution décrit dans la section 4.5.3.

4.3 Modèles à base de composants virtuels

Pour simplifier, les modèles d'exécution dans la suite sont appelés composants (un composant opère sur un modèle d'exécution).

Pour abstraire les différentes propriétés manipulées et faciliter la génération automatique d'un modèle d'exécution, on définit un modèle dit modèle à base de composants virtuels. C'est un modèle qui consiste à envelopper les différents composants d'un système dans une enveloppe dite *interface abstraite* connectée avec un environnement d'exécution. La figure suivante est une représentation du modèle à base de composants virtuels.

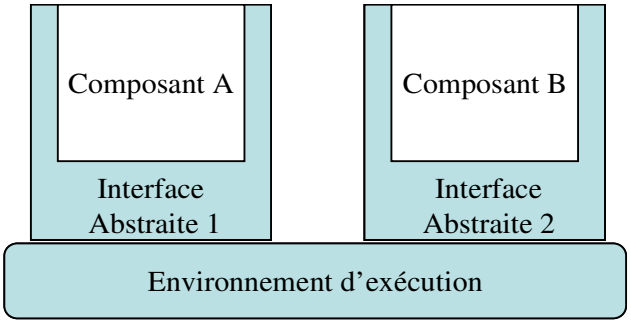


Figure 4-3 Modèle à base de composants virtuels

Ce concept est un concept populaire pour la communauté logicielle. On peut citer par exemple, les modèles CCM (CORBA Component Model) [Gei97], etc. Il a été également adopté par la communauté matérielle. On peut citer comme exemple Coware, SystemC, etc.

La puissance de ce modèle réside dans le pouvoir d'abstraire l'hétérogénéité des composants. Il est considéré, comme un support pour l'hétérogénéité (différents niveaux d'abstraction, différents composants, différents langages) des composants en cachant certains détails (d'architecture par exemple) et en permettant de retarder certaines décisions à travers l'utilisation de modèles génériques. De plus, l'utilité de ce modèle est visible lors de la génération du modèle d'exécution (voir section suivante) en permettant une exploration d'architecture à travers la génération d'un ou plusieurs schémas d'adaptation. Ceci grâce à la possibilité d'automatisation de la génération automatique des modèles d'exécution correspondants.

L'interface abstraite est définie par un ensemble de deux interfaces : l'interface interne et l'interface externe (voir Figure 4-4). L'interface interne est relative au composant. Elle peut être décrite à n'importe quel niveau d'abstraction vu dans le chapitre 2. L'interface externe est relative à celle de l'environnement d'exécution. Elle peut être décrite aussi à différents niveaux d'abstraction selon le modèle considéré de cet environnement (bus de cosimulation, fils physique, etc.). La Figure 4-4 montre une interface abstraite regroupant une interface interne, définie en tant qu'ensemble de services (API), et une interface externe, définie en tant qu'ensemble de ports physiques.

Le modèle à base de composants virtuels, tel qu'il est défini, est indépendant de toutes les implémentations et capable de modéliser des interconnexions hétérogènes.

Il permet de modéliser les interconnexions hétérogènes des différents modèles par la mise en correspondance des interfaces internes et les interfaces externes. Par exemple associer une interconnexion logique (voir chapitre1) entre S1 et P1.

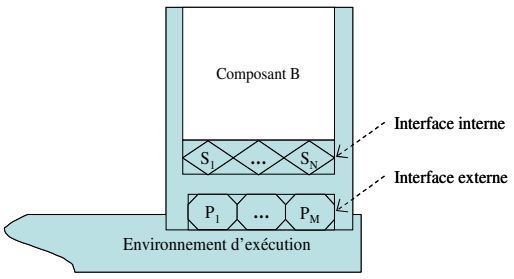


Figure 4-4 Interface Abstraite

Ce modèle est une représentation abstraite du modèle d'exécution global. L'interprétation des interconnexions hétérogènes permettra d'analyser les mises en correspondance entre les

interfaces. Rappel, l'interprétation de ces interconnexions permettra la génération des adaptateurs.

4.4 Rétrospective : comparaison des autres modèles d'exécution à base de composants virtuels

Plusieurs études étaient consacrées à la définition des modèles d'exécution à base de modèles à composants virtuels grâce à son efficacité à abstraire l'hétérogénéité des composants.

L'objectif de ce paragraphe est de présenter une rétrospective des différents modèles d'exécution existants qui sont à la base du modèle à composants virtuels. Parmi ces modèles, on relève le modèle CORBA, le modèle DCOM, le modèle SystemC, ROOM et Matlab/Simulink.

4.4.1 Modèle à base de composants CORBA

Le modèle CORBA est une présentation d'une architecture qui est apparue en 1990 [Cor04]. Elle permet la communication d'un ensemble d'applications dans un environnement hétérogène (langages, systèmes d'exploitations, etc.)

C'est une norme d'architecture distribuée ouverte. On peut l'assimiler à un bus fonctionnel qui offre plusieurs services à travers des interfaces. Parmi ces services, on relève les services de nommage, cycle de vie, événement, etc.

Les composants CORBA présentent des implémentations différentes (par exemple Java, C++, etc.). Ils peuvent s'exécuter sur des environnements différents (différents systèmes d'exploitation, différents processeurs, etc.). Ces composants communiquent avec l'extérieur *via* une interface définie comme un ensemble de services fournis ou requis par les composants.

Dans la norme CORBA, les différents services des composants ne peuvent exister qu'à travers l'enveloppe définissant une interface abstraite. Pour la description de cette interface, CORBA définit le langage de définition des interfaces IDL (Interface Description language). Ce langage permet ainsi la séparation entre la communication et l'implémentation.

Le bus d'objet de CORBA noté ORB (Object Request Broker) représente le conduit par lequel les requêtes sur les objets transitent. Il n'est pas accessible directement par le programmeur d'application CORBA. L'ORB gère les transferts entre les composants CORBA. Il représente une interprétation particulière des interconnexions au niveau service. Il est équivalent à l'environnement d'exécution dans le modèle à base de composants virtuels.
Le modèle à base de composants virtuels CORBA est donné alors par la Figure 4-5.

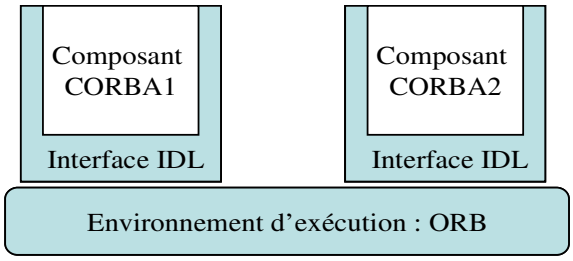


Figure 4-5 Modèle à base de composants virtuels CORBA

La figure 4-5 illustre un système intégrant deux composants implémentés par deux langages différent (Java et C++). Un des composants requiert un service de calcul de multiplication de matrices implémenté par l'autre composant. Pour décrire le modèle à base de composants virtuels de ce système, l'interface IDL doit être spécifiée. Un exemple de description de l'interface IDL est donné par la Figure 4-6.

```
interface Product_Service
{
  string Greeting(in string greetstr);
  short product_matrix(Matrix Mat1, Matrix Mat2);
  oneway void Shutdown();
};
```

Figure 4-6 Description d'une interface CORBA par IDL

Pour le modèle d'exécution global d'un système hétérogène décrit avec des composants CORBA différents, le modèle CORBA permet la génération des adaptateurs spécifiques. Ces adaptateurs sont chargés de la gestion des références aux objets offrant les services demandés.

Les noms des adaptateurs dépendent du rôle du composant (offrant des services ou demandant des services). Ils sont appelés souche et squelette respectivement.

L'environnement d'exécution ORB définit une interface permettant son interaction avec les composants CORBA. Cette interface est un ensemble d'API ou services. La Figure 4-7 illustre le modèle d'exécution global CORBA du système décrit ci-dessus. Elle présente un exemple de services internes et de services externes.

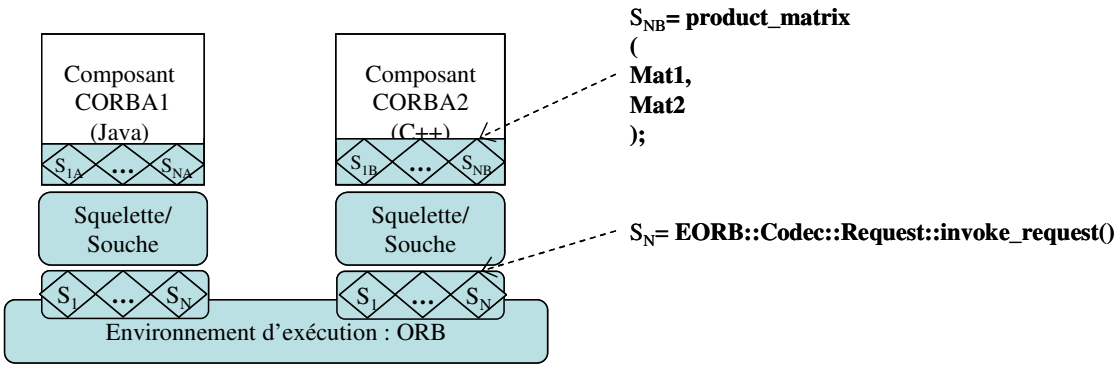


Figure 4-7 Modèle d'exécution CORBA avec une description de quelques services internes et externes

Les API offerts par l'ORB seront traités dans le Chapitre 5 à travers une application. La mise en correspondance entre les différents services internes et externes se fait par un mécanisme automatique de CORBA lors de la génération des adaptateurs.

4.4.2 DCOM

Le modèle DCOM représente une alternative de CORBA proposée par MicroSoft [Dco05]. Toutefois CORBA ne représente aucune limitation sur l'implémentation des différents éléments du modèle alors que DCOM définit, en plus de la norme, une implémentation spécifique. Ceci restreint l'utilisation du modèle et ne laisse aucune flexibilité.

C'est un modèle qui représente l'environnement d'exécution comme un bus fonctionnel permettant la communication des composants logiciels directement à travers un réseau.

4.4.3 SystemC

SystemC est une extension de C++ pour la description des systèmes électroniques essentiellement. Il représente une bibliothèque enrichie avec laquelle on peut réaliser la modélisation et la simulation des systèmes.

L'utilisation du langage SystemC est basée sur la séparation entre le comportement et la communication. La communication est déterminée à travers une interface de communication. L'interface est décrite à travers un ensemble de ports. Ces ports peuvent être de différents types : des ports d'entrées (noté sc_in), des ports de sorties (notés sc_out) et des ports d'entrées/sorties (notés sc_inout). Il fournit également un port spécifique pour la modélisation d'une horloge physique (noté sc_in_clk). Ces ports peuvent représenter des ports logiques ou des ports physiques.

Pour la modélisation des composants (modules) SystemC utilise la notion de SC_MODULE. Un module peut être hiérarchique contenant d'autres modules ou élémentaire contenant un comportement actif ou passif, ceci grâce aux différentes primitives de modélisation du comportement (« SC_METHOD », « SC_THREAD » ou « SC_CTHREAD »).

Pour la communication entre les différents modules, SystemC propose une modélisation de la communication permettant l'échange de données ou des signaux entre les différents modules. Ceci se réalise à travers la notion de « SC_CHANNEL » auquel on associe une interface de communication « SC_INTERFACE ». La description des médias de communication en SystemC est liée à la notion de « SC_CHANNEL ». Ce canal peut être décrit à différents niveaux d'abstraction. Il peut être une modélisation d'un canal abstrait comme une FIFO haut niveau ou de signaux bas niveau. Cette dernière modélisation se fait en utilisant les primitives du langage « sc_signal ».

SystemC offre un environnement d'exécution basé sur l'exécution de son simulateur des différents comportements associés aux modules et aux canaux. Deux phases d'exécution sont nécessaires : une phase d'initialisation et une phase d'exécution. Pendant la phase d'initialisation, il y a l'instanciation de tous les modules et l'établissement des interconnexions entre eux. Elle est effectuée lors de la phase d'élaboration. Pour la phase d'exécution, SystemC exécute toutes les tâches jusqu'à des points bloquants (par exemple des instructions de « wait »). Ensuite une mise à jour de tous les signaux est effectuée. Puis, une mise à jour de la liste des tâches prêtes à être exécutées est réalisée. Grâce à cette liste, SystemC détecte s'il y a des tâches prêtes ou non. Si oui, le temps de simulation est avancé seulement de delta cycle. Dans le cas contraire, l'horloge de SystemC est avancée et le cycle d'exécution reprend.

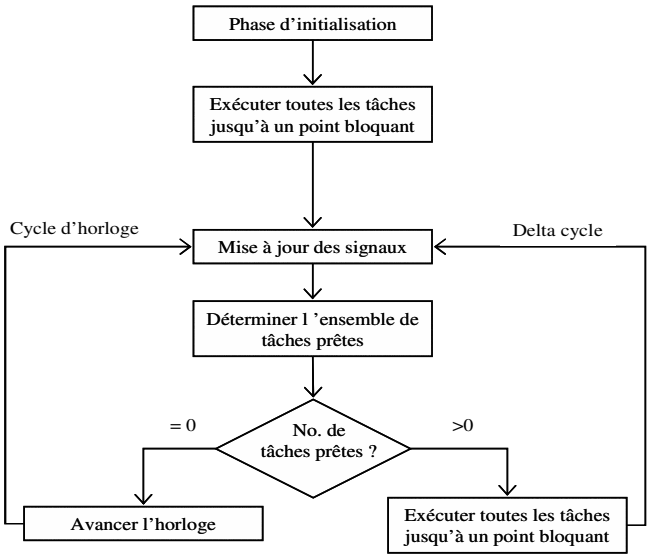


Figure 4-8 Ordonnanceur SystemC

La synchronisation de ce modèle peut être vue comme (1) une synchronisation par temps logique si l'équivalent d'une horloge physique n'est pas utilisé ou (2) une synchronisation par temps physique dans le cas contraire (voir la section 3.3.2).

Dans le modèle à base de composants virtuels de SystemC, l'interface de communication de l'environnement d'exécution est définie de manière implicite à travers la mise en correspondance des interfaces des modules avec les canaux. La Figure 4-9(a) montre le modèle d'exécution à base de composants virtuels de deux modules SystemC interconnectés. Alors que la Figure 4-9 (b) spécifie le modèle d'exécution de ce système. Dans ce cas, les deux modules opèrent sur un même modèle d'exécution (matériel niveau RTL, voir chapitre 2) et leurs interfaces respectives sont décrites au même niveau d'abstraction. Ainsi, il n'est pas nécessaire de générer des adaptateurs.

La description de ces modules et de deux interconnexions est donnée par la Figure 4.10 (a). Les interfaces des deux modules A et B sont formées par trois ports physiques chacune. L'interconnexion se fait par une modélisation d'un canal simple en utilisant la notion de signal comme le montre la Figure 4.10 (b). L'environnement d'exécution va permettre alors l'échange des données entre les différents modules à travers les signaux.

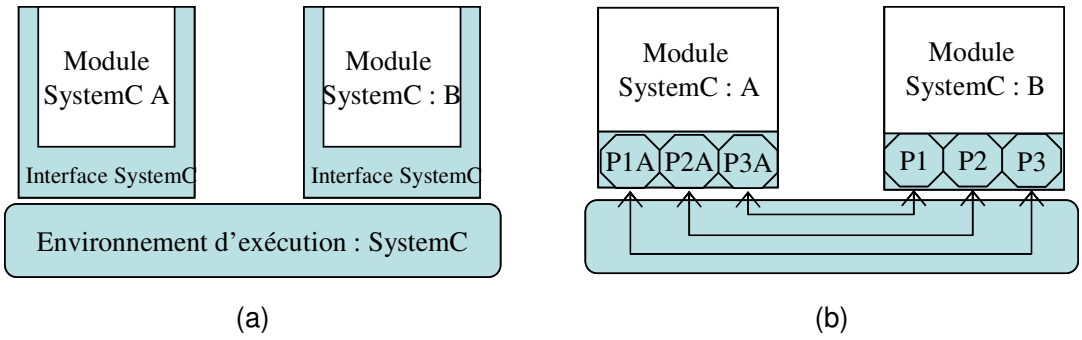


Figure 4-9 Modèle d'exécution basé sur SystemC

```
/*Description du module B*/
SC_MODULE (B)
{
/*Description de l'interface du module B*/
sc_in<bool> P1;
sc_in<bool> P2;
sc_out<bool> P3;
/*reste de la description*/
}
```

(a)

```
/* Description de la mise en correspondances des interface entre les
modules avec la spécification de l'environnement d'exécution : les
signaux*/
…{ …
sc_signal<bool> S1, S2, S3;
A M_A("Module A");                B M_B("Module B");
M_A.P1A(S1);                      M_B. P1(S1);
M_A.P2A(S2);                      M_B. P2(S2);
M_A.P3A(S3);                      M_B. P3(S3);
…}
```

(b)

Figure 4-10 Interconnexion entre deux modules SystemC avec des signaux

Il est à noter que SystemC nous offre la possibilité de modéliser des modules à différents niveaux d'abstraction. Toutefois, il n'existe pas de génération automatique des adaptateurs. C'est au concepteur de l'ajouter à la main.

4.4.4 ROOM « Real-Time Object Oriented Modeling »

C'est une méthodologie orientée objet pour les systèmes temps réel. Elle a été développée par « Bell Northern ». Elle est basée sur le principe d'utiliser le même modèle à travers toutes les phases du processus de développement. C'est un essai de formalisation de concepts à travers le flux de conception [Sel94].

Le modèle ROOM se base sur la définition des acteurs (en comparaison avec notre approche c'est le module) qui communiquent entre eux par envoi de messages selon des protocoles bien déterminés. Les acteurs peuvent être hiérarchiques et peuvent avoir un comportement élémentaire décrit essentiellement en utilisant le « ROOM CHART ». C'est un modèle ressemblant à celui développé par HAREL [Har87] : ce sont des machines d'états finis étendues. Les acteurs communiquent à travers une interface. Cette interface est un ensemble de ports et un ensemble de messages défini par le protocole associé à ces ports.

Il existe plusieurs types de ports dans ROOM :

- Ports de relais « RELAY PORT » : ils permettent de partager la classe de l'interface avec la classe du conteneur. Ces ports appartiennent à l'interface d'un acteur. Cependant les messages qui arrivent dans ce type de ports ne sont pas vus par le comportement du composant.
- Ports conjugués : ils manipulent les messages de sorties et les messages d'entrées d'un même protocole.
- Ports externes d'extrémité : ils appartiennent à l'interface d'un acteur. Ils sont connectés directement au comportement de l'acteur.
- Ports internes d'extrémité : c'est un port qui est utilisé pour connecter le comportement à un port interne de l'acteur. Ils ne font pas partie de l'interface de l'acteur, ils ne sont pas visibles de l'extérieur.

Les « ports de relais » et les « ports externes d'extrémité » font partie de l'interface d'un acteur. Cependant la distinction entre eux n'est pas discernable de l'extérieur [Roo05].

La représentation d'un acteur en ROOM est un ensemble de classes. La définition des différents ports est faite à travers la classe « Protocol ». Pour chaque port on précise le protocole et la direction des données. On peut considérer alors l'interface de ROOM comme étant des ports logiques vus dans le Chapitre 2**.** A travers les protocoles, des services peuvent être définis et associés à des ports hauts niveaux. Pour chaque interface d'acteurs, on définit le protocole associé. Avec cette approche, on ne peut connecter que des **interfaces ayant des protocoles**

compatibles. C'est un concept qui ne supporte pas l'hétérogénéité des protocoles de communication.

La Figure 4-11 présente un composant ROOM avec la description de son interface. En résumé, les ports (par exemple P1) font référence aux protocoles (par exemple Protocole P1) associés eux même à des classes (par exemple Class P1). Le port est défini alors comme un port d'entrée qui fait le transfert de données de types data1.

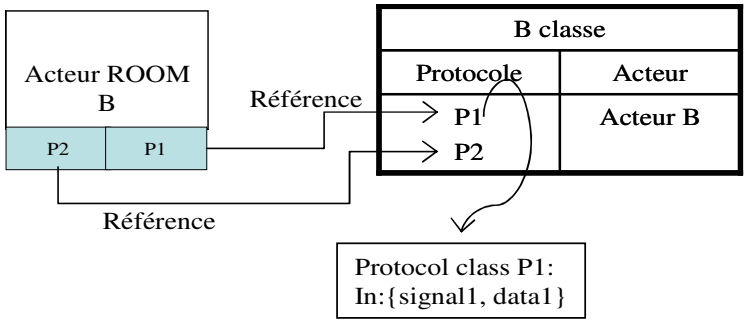


Figure 4-11 Description d'un acteur et son interface en ROOM

Le modèle à base de composants virtuels est défini ici par cette notion d'encapsulation de classes dans des acteurs qui communiquent via l'interface définie ci-dessus.

L'interaction entre les différents composants ROOM est assurée par le mode d'envoi et de réception des messages entre les acteurs. On peut alors définir le modèle d'exécution à base de composants virtuels (voir Figure 4-12 (a)). Les adaptateurs, au niveau du modèle d'exécution basé sur ROOM, ne sont pas définis du fait que ROOM ne peut connecter que des interfaces compatibles. L'environnement d'exécution comporte l'exécution des tâches et l'ordonnancement des différents messages envoyés et/ou reçus (Figure 4-12 (b)).

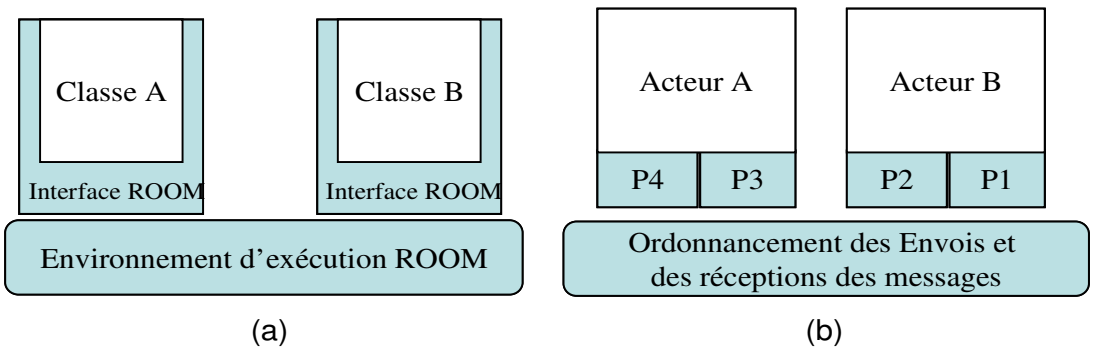


Figure 4-12 Modèles basés sur ROOM (a) Modèle à base de composants virtuels ROOM, (b) Modèle d'exécution d'un système basé sur ROOM

4.4.5 Matlab/Simulink

Matlab/simulink est un langage de modélisation et de simulation des systèmes manipulant dominés par les données. Il permet aussi l'analyse des systèmes dynamiques évoluant au cours du temps (les données et les états du systèmes dépendent du temps de manière continue), ceci en offrant une bibliothèque d'objets sous forme de blocs standards paramétrables (tels que des multiplieurs, des additionneurs, des intégrateurs, etc.). Le langage Matlab/Simulink offre plus une interface graphique permettant une manipulation facile de ces objets [Mat05].

Il permet la séparation entre l'interface de communication et le comportement. L'utilisateur observe et manipule seulement l'interface entre les blocs. Ils représentent des modules dans notre terminologie.

L'interface est un ensemble de ports auxquels on définit la nature des données et l'interconnexion avec les autres blocs. Cette interconnexion est faite d'une manière graphique sous forme de traits. Ces traits modélisent des signaux représentant les valeurs de sorties d'un bloc donné et/ou des signaux à temps continu ou discret. Ces valeurs possèdent des types prédéfinis et scalaires. Les types scalaires utilisés sont « entiers », « réels » et « booléens ». La communication entre les différents ports est similaire à un envoi de message synchrone (Rendez-Vous). Les blocs émetteur et récepteur se synchronisent pour effectuer un échange de données [Sim05].

La définition du média de communication en Matlab/Simulink est assimilée à un canal de type FIFO haut niveau. La modélisation de ces canaux peut être simple ou complexe. Si on veut exprimer des médias spécifiques tel qu'un bus ou un réseau spécifique, il faut les décrire explicitement à l'aide d'autres modules (blocs) élémentaires.

Le modèle d'exécution de Matlab/Simulink suppose que les blocs sont de même nature et au même niveau d'abstraction. Si deux blocs ne possèdent pas le même niveau d'abstraction, la conception des adaptateurs doit être réalisée manuellement.

L'environnement d'exécution dépend de la nature du solutionneur utilisé dans simulink. Cet environnement permet l'échange des différentes valeurs de sorties entre les différents blocs. Deux phases d'exécution sont nécessaires pour réaliser cet échange : phase d'initialisation et la phase d'exécution. La phase d'initialisation est faite pour tous les blocs en même temps et leur exécution suit un algorithme dépendant du solutionneur utilisé. Elle comprend les actions suivantes :

- Evaluation des expressions des paramètres des blocs.
- Nivellement de la hiérarchie des modèles.
- Détermination de l'ordre de la mise à jour des blocs (phase de tri).
- Détermination des attributs des signaux et vérification de la concordance entre signaux.

- Détermination du temps d'échantillonnage pour les blocs n'ayant pas cette spécification.
- Allocation et initialisation de la mémoire utilisée pour sauvegarder les états et les sorties de chaque bloc.

La phase d'exécution comprend les actions suivantes

- Le calcul successif des états et des sorties du système à des intervalles du début de la simulation à sa fin.
- Le calcul du temps d'avancement dépendant du solutionneur.

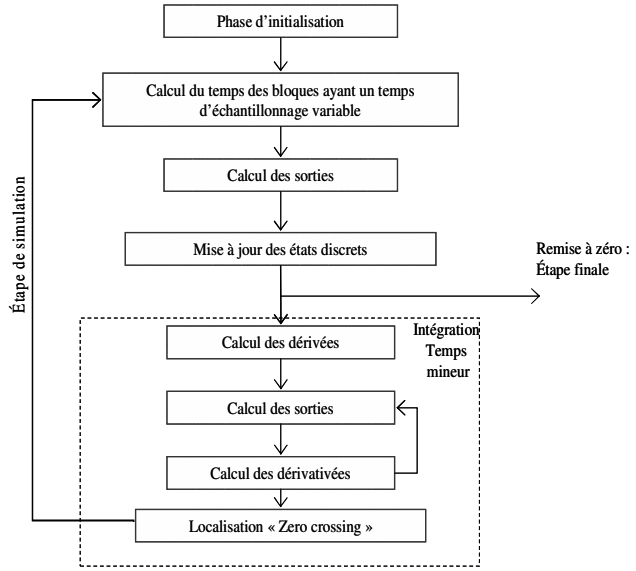


Figure 4-13 Ordonnanceur Simulink

L'Ordonnanceur de Simulink est donné par la Figure 4-13.

En simulink, on peut définir des modules utilisateurs par l'intermédiaire des blocs prédéfinis : les S_fonctions. Ces modules permettent de modéliser l'interface comme un ensemble de ports décrits par la figure 4-14.

```
#define IN_PORT_0_NAME      P1
#define INPUT_0_WIDTH       1
#define INPUT_DIMS_0_COL    1
#define INPUT_0_DTYPE       real_T
#define INPUT_0_COMPLEX     COMPLEX_NO
#define IN_0_FRAME_BASED    FRAME_NO
#define IN_0_DIMS           1-D
#define INPUT_0_FEEDTHROUGH 1
```

Figure 4-14 Interface décrite en Simulink

La description de ces interfaces comporte le nom du port, sa direction (port d'entrée ou de sortie), la taille et le type des données pouvant être véhiculés à travers ce port. Les ports décrits en Simulink sont des ports logiques (chapitre 2). On associe des actions de lecture et d'écriture (voir chapitre 5).

4.5 Vers la génération automatique des modèles d'exécution généraux des systèmes hétérogènes

Dans cette section, nous décrivons une implémentation de notre approche pour la génération des modèles d'exécution des systèmes hétérogènes.

Cette implémentation se base sur un modèle à base de composants virtuels nommé COLIF. Ce modèle sera décrit dans la section 4.5.1. Ce modèle est le point de départ pour la génération des différents modèles d'exécution. Cette génération se base sur des bibliothèques représentant l'architecture des adaptateurs (voir 4.5.3). Cette architecture possède une représentation particulière permettant ainsi la généralisation de notre modèle d'exécution.

4.5.1 Modèle à base de composants virtuels

Dans cette partie, nous définissons l'implémentation du modèle à base de composants virtuels utilisé dans notre approche. Ce modèle est le résultat d'un travail de groupe. Ma contribution dans ce modèle est l'ajout de certains concepts étudiés pour sa généralisation (par exemple, la définition des interfaces des modules, les interconnexions hétérogènes, voir chapitre 1).

Le modèle développé représente un outil génie logiciel. Il est nommé COLIF [Ces01], [Gau01]. Il utilise le concept présenté dans la partie 4.3. Il est décrit par le langage XML (Extensible Markup Language) [Xml05]. XML est un métalangage standardisé par W3C facilitant l'intégration des outils et l'échange des informations entre différents environnements de synthèse grâce à des grammaires spécifiques. COLIF définit aussi une grammaire de description pour les systèmes hétérogènes. Cette grammaire est donnée dans [Gau01].

Le diagramme représentant les concepts COLIF est présenté dans la figure suivante [Xi05].

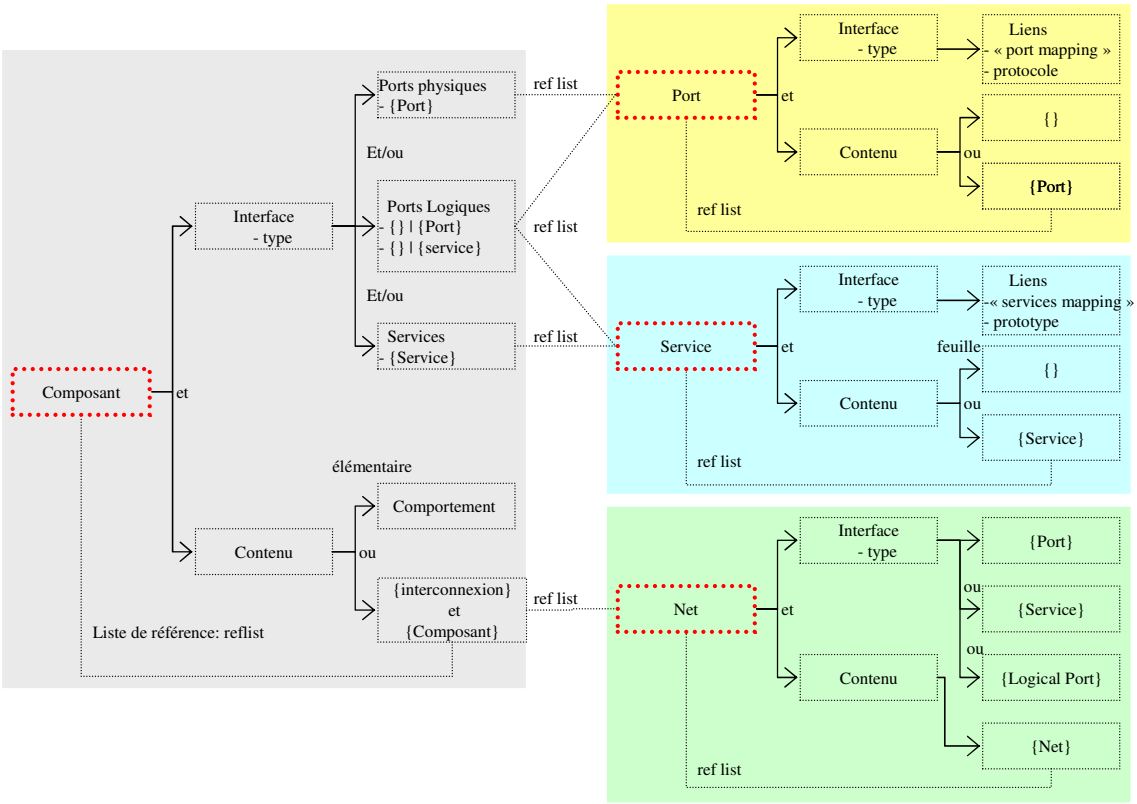


Figure 4-15 Diagramme de concepts COLIF

COLIF présente trois concepts de base : les composants, les interfaces et les interconnexions. Ces concepts étaient détaillés dans le chapitre 1.

Grâce à ces concepts, nous pouvons présenter notre système sous forme d'un modèle à base de composants virtuels :

- les modèles d'exécution des sous-systèmes (ayant un contenu et une interface interne) en tant que composant simple COLIF.
- l'interface abstraite via la notion de modules abstraits (composant particulier) ayant deux types d'interfaces (ensemble de ports et services) internes et externes. Cette description est réalisée grâce aux liens entre les différentes interfaces des composants et des interconnexions.
- les interconnexions (présentées ici par le « net ») définissent l'environnement d'exécution. Elle est définie par son interface et son contenu. Ces interconnexions permettent de définir les interconnexions hétérogènes entre les interfaces internes et les interfaces externes.

4.5.2 Génération automatique des modèles d'exécution globaux des systèmes hétérogènes

Le flot de génération des modèles d'exécution globaux des systèmes hétérogènes se base sur une présentation du système global par l'intermédiaire du modèle à base de composants virtuels COLIF.

La génération du modèle consiste, comme défini dans la section 4.2.2, en la définition des adaptateurs et de l'environnement d'exécution. La définition des adaptateurs dépend du modèle d'exécution des composants (logiciel, matériel ou fonctionnel), de leurs niveaux d'abstraction, etc. Ils sont déterminés à partir de bibliothèques définissant les éléments des adaptateurs. Par exemple, pour un système ayant un composant logiciel et un composant matériel. On distingue deux types d'adaptateurs, un adaptateur du coté logiciel, dans lequel on définit un système d'exploitation par exemple, et un adaptateur matériel définissant l'adaptation entre le bloc matériel et le reste du système. Ces adaptateurs sont définis à partir de deux bibliothèques différentes.

La figure suivante présente le flot de génération des adaptateurs :

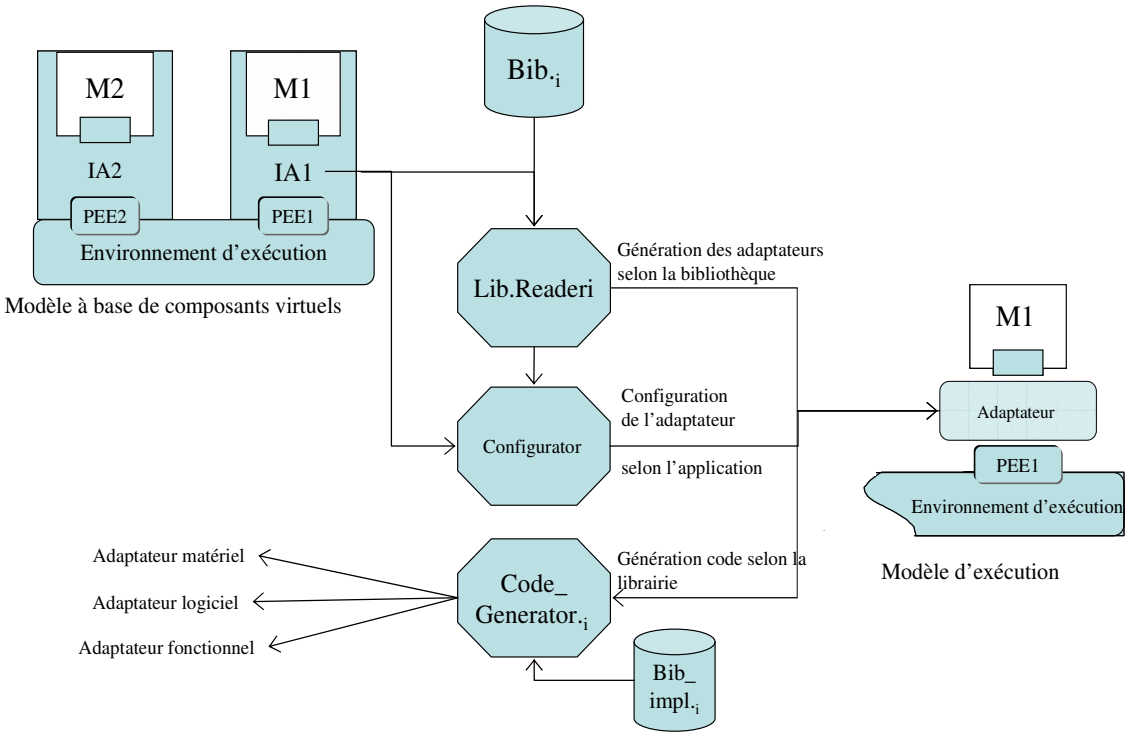


Figure 4-16 Flot de génération des modèles d'exécution pour les systèmes hétérogènes

Ce flot présente trois étapes :

- Etape 1 : le parcours de la bibliothèque pour déterminer l'architecture de l'adaptateur noté dans la figure par « Lib.Reader$_i$ ».

- Etape 2 : la configuration des adaptateurs selon l'application et les caractéristiques des modules. Cette étape est notée dans la figure par « Configurator ».
- Etape 3 : la génération du code selon les bibliothèques d'implémentation des adaptateurs, notée dans la figure par « Code generator$_i$ ».

A partir du modèle à base de composants virtuels, et selon des bibliothèques spécifiques pour les composants (logiciels, matériels et fonctionnels), l'étape 1 consiste à déterminer l'architecture de l'adaptateur selon un graphe dit graphe de dépendance de service noté GDS (Graph Dependency Services). Ce graphe sera expliqué dans la section 4.5.3.1.2. Cette architecture est uniforme pour tous les adaptateurs. Cependant, les éléments définissant cette architecture dépendent de la nature des adaptateurs. Ces architectures sont détaillées dans [Sar05a], [Bou05] et [Gra05].

La deuxième étape est la configuration des éléments des bibliothèques selon le système utilisé. Par exemple, dimensionner les éléments en utilisant le type des données transférées entre les composants de notre système.

La troisième étape porte sur la génération du code selon les implémentations des différents éléments (dans les bibliothèques d'implémentation de chaque type). Ainsi, des adaptateurs logiciels dans le cas d'adaptation de composants logiciels ou des adaptateurs matériels pour des composants matériels ou des adaptateurs fonctionnels pour des composants fonctionnels seront générés.

Ces différentes étapes sont définies pour chaque adaptateur dans [Sar05a], [Gau01] et [Gra05].

4.5.3 Modèle général d'adaptateur du modèle d'exécution global des systèmes hétérogènes : modèle basé sur les services

Dans cette partie du chapitre, nous proposons un modèle d'adaptateur de communication basé sur des services. Ce modèle est basé aussi sur l'assemblage de composants. Il a été défini en collaboration avec [Sar05a] [Sar05b]. Cette tâche était un travail de groupe qui a abouti aux définitions présentées dans la section 4.5.3.1. Dans cette partie, nous détaillons l'apport d'une telle architecture pour la généralisation de notre modèle d'exécution.

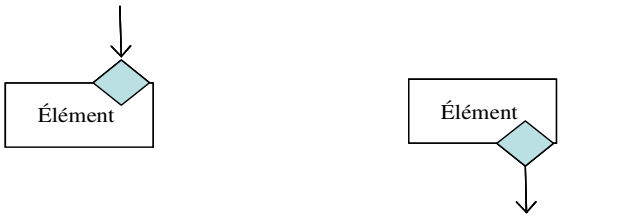
Le modèle basé sur les services n'est pas un nouveau concept. « Zitterbart et al » a proposé un tel modèle pour implémenter le logiciel embarqué pour des applications de télécommunication [Zit93]. L'outil ASOG du groupe SLS [Gau01] génère des interfaces logicielles en utilisant un modèle basé sur des services. Pourtant, ces travaux sont plus adaptés au domaine logiciel qu'au modèle matériel. Ils ne prennent pas en compte certains concepts du domaine matériel, (comme des ports physiques par exemple). Nous proposons alors un modèle complémentaire à ces travaux afin de rendre notre architecture générale et par conséquent le devient aussi notre modèle d'exécution global des systèmes hétérogènes.

Pour mieux expliquer le modèle basé sur les services que nous proposons, nous présenterons d'abord les composants de base du modèle, définissant les éléments de la bibliothèque. Par la suite, nous présentons l'assemblage de ces éléments pour définir l'architecture de l'adaptateur.

4.5.3.1 Composants du Modèle de l'adaptateur

Le modèle de l'adaptateur est défini comme un assemblage de modules. Chaque module est défini par un comportement et une interface. L'interface peut être un service, un port physique ou un port logique (voir chapitre 1). Un module peut avoir des interfaces de natures différentes, c'est-à-dire un port physique et un service. Chaque module de l'interface appelé *élément*, est une unité abstraite qui fournit et/ou demande un ou plusieurs services. Les services, donc, sont implémentés par ces éléments d'interface. Ils sont déterminés au niveau de l'interface. Pour les interfaces définies comme un ensemble de ports physiques, les services sont déterminés par un point d'accès aux ports noté PAP (Point Access Port).

Ceux-ci peuvent être des entités logicielles et/ou fonctionnelles (comme des classes C++). L'assemblage de ces éléments se fait par la mise en correspondance de leurs interfaces. La mise en correspondance entre les interfaces est définie par un graphe de dépendance de services (voir 4.5.3.1.2). Pour plus de clarté, un élément de l'adaptateur avec son interface est présenté comme suit :



(a) Élément fournissant un service (b) Elément requérant un service

Figure 4-17 Eléments d'une interface d'adaptation

Les éléments accédant directement aux ports physiques présentent des accès spécifiques. Ces accès sont effectués par l'intermédiaire de point d'accès aux ports notés « PAP ». Ils sont présentés par des triangles (voir Figure 2-3). La figure suivante présente un tel élément.

(b) Elément fournissant un service à un port physique à travers le « PAP » (a) Elément requérant un service d'un port physique « PAP »

Figure 4-18 Eléments du graphe accédant à des ports physiques

Pour les ports logiques des interfaces des modules du système, deux sortes d'accès peuvent se faire soit par le biais du point d'accès services (SAP) soit par le biais du point d'accès du port (PAP) pour demander ou accéder à des services voir Figure 4-19.

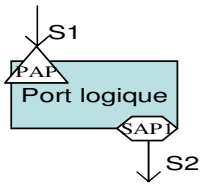


Figure 4-19 Accès aux services d'un port logique

4.5.3.1.1 Composition des Adaptateurs de Communication

La composition des adaptateurs de communication se fait en utilisant la mise en correspondance des interfaces des éléments. Ces relations peuvent être modélisées par un graphe dit graphe de dépendance de services (GDS). Un adaptateur de communication est donc constitué de l'interface, relative à un des modules du système, et de l'interface, relative à l'environnement d'exécution, reliées par un GDS.

4.5.3.1.2 Graphe de Dépendance de Services

La Figure 4-20 présente un GDS. Un noeud du graphe de dépendance de services peut être un port logique, un service ou un élément d'interface. Un arc dans le GDS définit la relation de dépendance entre un port logique ou un élément d'interface et un service. Une relation de dépendance existe quand un port logique ou un élément d'interface requiert ou fournit un service particulier à travers les services ou les ports. Dans l'exemple représenté sur la Figure 4-20, les ports PM1 et PEE1 sont des ports logiques regroupant d'autres ports logiques (P1, P2 et P3). Alors, PM1 demande le service S1 (par le biais d'un SAP), S1 est fourni par l'élément E1 qui requiert le service S5. S5 est fourni par l'élément E2, qui demande le service S6 (par le biais d'un PAP) qui est fourni par le port PEE1.

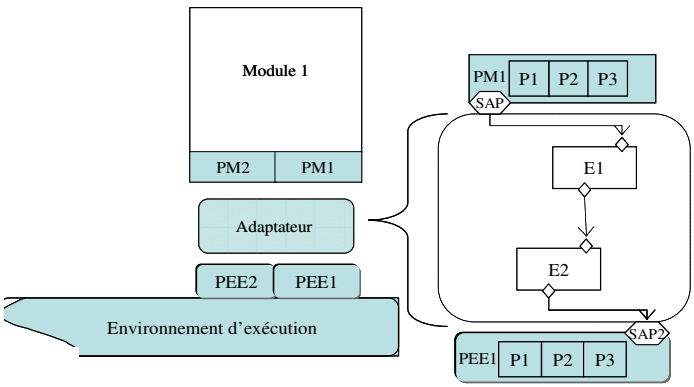


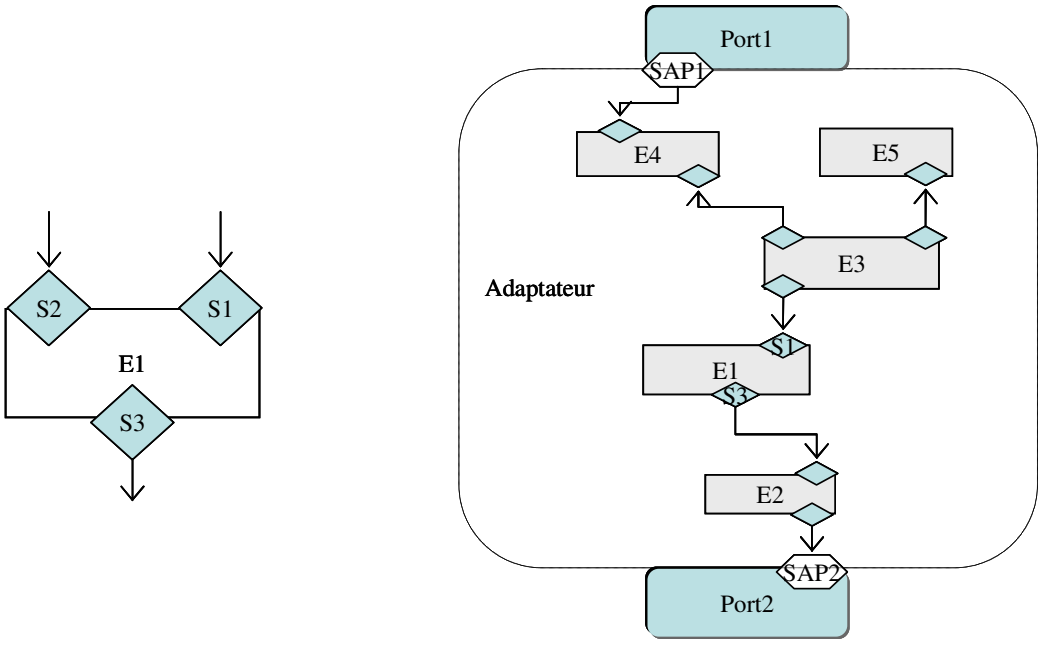
Figure 4-20 Graphe de Dépendance de Services

Il faut remarquer qu'un GDS représente, en effet, la décomposition fonctionnelle de la fonction globale d'adaptation de communication. Les services dans un GDS représentent les fonctions nécessaires pour effectuer une telle adaptation, tandis que les éléments d'interface et ports logiques représentent les composants de base qui vont implémenter ces services.

La granularité fine des services facilite l'implémentation des adaptateurs de communication plus flexibles. Additionner/supprimer une fonctionnalité d'un adaptateur, par exemple, peut impliquer seulement l'addition/suppression d'un service. L'idée d'avoir le service comme composant de base facilite aussi sa réutilisation, puisqu'une même fonctionnalité peut être nécessaire à plusieurs adaptateurs de communication.

L'implémentation finale d'un adaptateur de communication fournit seulement les services demandés dans le GDS. En effet, si un élément fournit plusieurs services, l'implémentation de cet élément concernera seulement le service requis ou fourni seulement par l'adaptateur en question. Par exemple, si l'élément E1 fournit les services S1 et S2 et requiert S3. Lors de l'assemblage des éléments de l'adaptateur, si seulement S1 et S3 de E1 sont nécessaires (voir la Figure 4-21) alors lors de la phase de configuration de l'adaptateur vu dans 4.5.2, seulement le service S1 et S3 seront implémentés.

Ainsi, le modèle aide la génération des adaptateurs de communication plus performants, car ceux-ci n'implémentent que les services strictement nécessaires pour la fonction d'adaptation.



(a) Elément E1 avec ces différents services

(B) Sélection de certains services de l'Elément E1

Figure 4-21 Sélection des services nécessaires pour l'adaptation

Enfin, le modèle n'impose aucune contrainte par rapport à l'ordre d'exécution des services dans un adaptateur de communication. Si par exemple un élément requiert deux ou plusieurs services, ces services peuvent être exécutés en parallèle dans l'adaptateur. Ainsi, la construction des adaptateurs plus performants est réalisée.

4.6 Conclusion

Dans ce chapitre, nous avons défini un modèle global et général d'exécution des systèmes hétérogènes. Une première partie a été consacrée à l'étude des travaux existants en montrant leur insuffisance à définir un modèle d'exécution global et général. La deuxième partie est dédiée à la définition d'un modèle d'exécution global généralisé. Ce modèle se base sur la définition d'un environnement d'exécution et des adaptateurs. Ces éléments permettent d'interpréter les interconnexions hétérogènes. Afin d'abstraire ces interconnexions, nous avons défini un modèle à base de composants virtuels. Ce modèle est une généralisation de ce qui existe déjà. La dernière partie est consacrée à l'étude d'une architecture des adaptateurs.

Chapitre 5 Applications et résultats expérimentaux

SOMMAIRE

5.1 Introduction

Les concepts mis en oeuvre dans les chapitres précédents pour la spécification et la génération du modèle d'exécution pour les systèmes hétérogènes, ont été appliqués pour quelques applications complexes dont notamment trois systèmes complexes - un centre d'information pour les véhicules, un modem d'une chaîne audio 802.16 et un système de radio définie par logiciel.

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

5.2 Modélisation d'un centre d'information de véhicule

Dans cette partie du chapitre, nous étudierons la modélisation d'un centre d'information de véhicule. Ce système se composera de modules hétérogènes modélisés par différents simulateurs. Il nécessitera la définition d'un modèle d'exécution global pour sa validation.

5.2.1 Présentation générale de l'application du centre d'information de véhicule

Dans notre cas d'étude, on envisage de réaliser un centre d'information automobile. C'est un système qui est dédié à l'amélioration du confort et surtout la sécurité du conducteur et des passagers. Ce système aura une reconnaissance vocale des commandes du conducteur et synthèse vocale pour délivrer des informations sur les conditions de circulation (trafic, météo). Ce système sera relié à des caméras de surveillance de l'état de fatigue du conducteur, à des radars anti-collision, des caméras de guidage et à de nombreux autres capteurs permettant de contrôler l'état des divers éléments du véhicule, comme la pression et la température des pneus. Le but de ce système sera réellement de prévenir tout accident en alertant le conducteur au plus tôt.

Le système en question est composé, comme le montre la Figure 5-1, de processeurs pour le traitement des différentes données provenant de plusieurs capteurs. L'interconnexion entre les différentes unités de ce système doit être minimale (puisqu'il faut le mettre sur une puce). C'est alors l'utilisation du mode de communication par fibres optiques ou RF. De plus, et afin de minimiser la consommation d'énergie, ce système comporte des capteurs autonomes en utilisant des sous-systèmes nommés MEMS (Micro Electro Mechanical System).

Un tel système de «centre d'information» est particulièrement complexe à concevoir d'autant plus que le secteur automobile est soumis à deux contraintes : le rapport prix/ performance et les conditions hostiles de fonctionnement : la température peut varier de –40°C en Scandinavie à +55°C dans le désert, les vibrations sont particulièrement néfastes aux soudures et enfin, lors du démarrage, des pics de tensions très élevés peuvent endommager les

composants. C'est pourquoi il est nécessaire de pousser l'intégration au maximum pour réduire à la fois les coûts et les problèmes de connexion entre circuits.

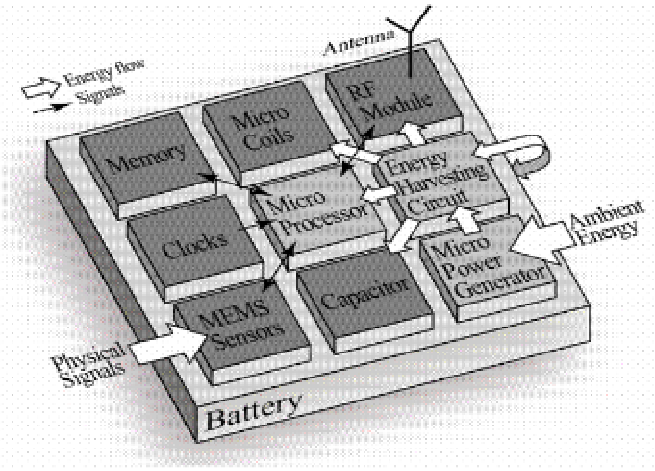


Figure 5-1 Les différents composants du centre d'information du véhicule

La complexité et l'hétérogénéité de tels systèmes rendent impossible la construction d'un prototype virtuel, comportant des éléments de natures différentes à l'échelle de plusieurs centaines de millions, en raison (a) de l'absence d'outils de validation adaptés d'une part, et (b) de l'absence d'aide à la conception d'autre part.

Le **but global** est, tout en permettant un travail coopératif entre plusieurs équipes, de trouver un modèle d'exécution global de la description hétérogène de ce système.

Dans le cadre de cette application, nous nous intéressons à l'intégration de deux composants, décrits par deux langages de modélisation différents, les microprocesseurs et les systèmes autonomes. Le but est d'avoir un système qui s'auto-alimente. Le principe de l'auto alimentation repose sur la récolte de l'énergie environnementale (vibrations, chaleur, lumière…), la conversion de cette énergie en puissance électrique, le stockage de l'énergie, l'extraction et la transmission des différents signaux. Ceci est réalisé grâce à un micro générateur de puissance et une batterie. Ces deux parties seront expliquées dans la section 5.2.1.1.

5.2.1.1 Les microsystèmes de génération de puissance

Le modèle global des microsystèmes de génération de puissance est donné dans [Zen05b]. Il se compose par :

- **Le Micro générateur de puissance** : le micro générateur de puissance (µPG) se compose d'un élément piézoélectrique qui convertit l'énergie environnementale (vibrations) en énergie électrique [Zen05a].

- **EHC** (Energy Harvesting Circuit) : il contient le circuit de récolte de la puissance et un circuit de contrôle de charge et de décharge de la batterie. Ces éléments permettent d'assurer le transfert continu de la puissance optimale et de maximiser la puissance stockée dans la batterie.
- **La batterie** est un composant permettant le stockage de l'énergie. Le but de son utilisation est de remédier à l'utilisation d'une connexion électrique directe.

5.2.1.2 Les microprocesseurs

Les microprocesseurs comportent différents traitements (traitement d'images, traitement de signal, traitement d'énergie, etc.). Ces microprocesseurs sont reliés au microsystème autonome pour réduire ou relancer leur activité selon l'état de l'énergie.

L'application tournant sur le microprocesseur est un algorithme de calcul de consommation globale du reste du circuit. Suivant une comparaison entre la consommation calculée et les états de charge et de décharge de la batterie, le microprocesseur (application) envoie une commande pour la charge ou la décharge de la batterie au circuit de contrôle de l'EHC.

Le microsystème de génération de puissance réagit par rapport à ces données pour charger ou décharger la batterie. Une valeur est reçue par le processeur représentant l'état actuel de la batterie notée « SOBat ». Les données envoyées au microsystème de génération sont notées « commande de charge » notée « Cmd_C ».

La modélisation de ce système est réalisée en utilisant plusieurs langages de spécification. Chaque composant de ce système est défini par son modèle d'exécution (voir section 5.2.1.3). Le modèle d'exécution global est alors déterminé par l'assemblage de ceux des composants, les adaptateurs et l'environnement d'exécution (voir Chapitre 4). Ce modèle sera présenté en détail dans la section 5.2.1.5.

5.2.1.3 Modèles d'exécution des composants du centre d'information du véhicule

Le choix du langage de modélisation est important pour la spécification des composants. Ce choix dépend des différentes compétences des différentes équipes de modélisation et se voit comme une première orientation vers l'implémentation réelle. Grâce au langage, on pourra déterminer le modèle d'exécution des composants (logiciel, matériel ou fonctionnel).

Dans le sous-système étudié, sur le microprocesseur, une application logicielle va être exécutée, ce qui favorise l'utilisation du langage SystemC (voir 5.2.1.4).

La modélisation du microsystème de génération de puissance est réalisée en utilisant le langage de modélisation Matlab/Simulink (voir 5.2.1.4.2).

A ce stade de cette étude, on s'intéresse seulement à une modélisation fonctionnelle de tous les modules sans ce soucier de l'implémentation matérielle ou logicielle, ce qui implique

que les différents modèles d'exécution sont des modèles fonctionnels. Pour chaque modèle, une étude sur les ordonnanceurs respectifs des simulateurs a été nécessaire du fait qu'il y aura une communication entre plusieurs simulateurs.

5.2.1.4 Modèle d'exécution du microprocesseur

Le modèle d'exécution du microprocesseur est un modèle fonctionnel décrit avec SystemC. L'exécution du comportement est la réalisation de l'application de calcul de consommation globale du reste du circuit. L'interface de communication avec le microsystème de génération de puissance est définie au niveau message. A ce niveau, on ne spécifie pas encore le protocole de communication entre les deux sous-systèmes. Les données sont de type prédéfini (réel).

Cette interface comporte un ensemble de deux ports logiques permettant la communication avec l'extérieur. Ces ports sont notés « PCmd_C » et « PSOBat ». Des actions sont associées à ces deux ports : action d'écriture sur le ports « PCmd_C » et action de lecture sur le port « PSOBat ». L'action d'écriture correspond à l'envoi des données du composant vers l'extérieur et la lecture correspond à la réception des données de l'extérieur vers le composant.

Le modèle d'exécution du microprocesseur est présenté par la figure suivante.

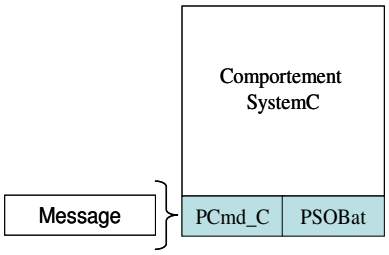


Figure 5-2 Modèle d'exécution fonctionnel du microprocesseur

5.2.1.4.1.1 Choix du langage SystemC

Le modèle fonctionnel de ce composant est décrit en langage SystemC. Ce langage est une extension de C++ pour la description des systèmes électroniques. Il représente une bibliothèque enrichie avec laquelle on peut réaliser la modélisation et la simulation des systèmes électroniques. L'utilisation du langage SystemC est basée sur les concepts qu'il peut fournir et l'ensemble des API facilitant la modélisation de tels systèmes. Ces concepts sont décrits ci-dessous :

- La hiérarchie : afin d'aborder la complexité des systèmes et de permettre une modélisation simple et claire, SystemC nous offre le concept de la hiérarchie. Ce concept est fourni grâce à la déclaration d'un composant comme un SC_MODULE. Ce composant peut à son tour contenir d'autres SC_MODULE.

- La concurrence : SystemC nous offre grâce à son simulateur intégré la possibilité d'exécuter en parallèle plusieurs modules (tâches, ou composants). Ceci est exprimé par l'utilisation des notions de SC_THREAD, SC_METHOD et SC_CTHREAD.
- La communication : pour que les différents modules puissent communiquer, SystemC propose une modélisation de la communication permettant l'échange de données ou des signaux entre les différents modules. Ceci est défini à travers la notion de SC_CHANNEL et la nouvelle notion d'interface de communication.
- La description des media de communication en SystemC est liée à la notion de SC_CHANNEL. Ce canal peut être décrit à différents niveaux d'abstractions. Il peut être une modélisation d'un canal abstrait comme une FIFO haut niveau ou de signaux bas niveaux. Avec ce canal, on précise les différents types de données. La synchronisation : comme l'environnement d'exécution de SystemC englobe de la concurrence, il est nécessaire de coordonner l'exécution des différents modules. Cette synchronisation est faite grâce aux différentes variantes de l'instruction Wait offerte par ce langage.

5.2.1.4.1.2 Description de l'interface de communication

L'interface de communication pour ce composant fonctionnel est décrite au niveau message. Le langage SystemC nous permet cette description à ce niveau grâce à ces définitions de ports en utilisant « sc_in » pour les ports d'entrées et « sc_out » pour les ports de sorties sans spécifier le protocole de communication entre les deux sous-systèmes.

La description de l'application du microprocesseur sera un module (SC_MODULE) ayant pour interface deux ports : un de sortie et un d'entrée. Ce module est élémentaire, il ne contient pas d'autres modules. Le listing de la Figure 5-3 nous montre la modélisation de notre composant avec son interface. L'interaction avec l'extérieur se fait à travers les ports « PCmd_C » et » PSOBat ». Le type de données échangées dans ce cas d'exemple est le même, c'est un type scalaire défini par le langage de type réel, noté « double ». Ce type est l'équivalent au type algorithmique « réel». Le comportement est précis au niveau cycle. On l'observe à travers la liste de sensibilité notée par « sensitive » dans le constructeur du composant.

```
SC_MODULE (Microprocessor) {

/*interface*/

      sc_out<double >   PCmd_C;
      sc_in<double >   PSOBat;
// comportement
      void calculate_Consume ();
      SC_CTOR(calcul)
      {

      SC_THREAD(calculate_Consume)  ;
      sensitive<< clk;
      };
};
```

Figure 5-3 Modélisation du microprocesseur en SystemC

Les actions sur ces ports sont des actions de lecture ou d'écriture. Elles sont offertes par un ensemble d'API. En SystemC, les API d'écriture ou de lecture sont décelées derrière une surcharge de l'opérateur « = ». Par exemple pour envoyer une valeur vers l'extérieur il suffit d'écrire « PCmd_C = val » et pour lire « val = PSOBat ».

5.2.1.4.2 Modèle d'exécution du microsystème de génération de puissance

Le modèle d'exécution du modèle généré est un modèle fonctionnel. Il est décrit avec le langage de modélisation Matlab/Simulink. Le modèle d'exécution de ce composant est donné par la figure suivante.

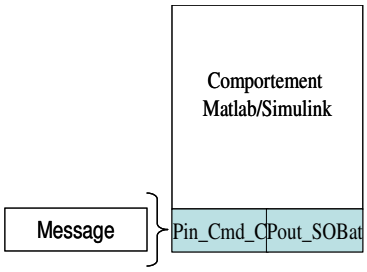


Figure 5-4 Modèle d'exécution fonctionnel du capteur autonome.

D'un point de vue modélisation sous SIMULINK/MATLAB, le microsystème peut être partitionné en blocs ou sous-systèmes. Ces sous-systèmes sont modélisés avec les éléments élémentaires de la librairie de Simulink pour l'analyse comportementale du microsystème complet. Cependant, le microsystème de génération de puissance communique avec l'extérieur via son interface de communication. Elle comporte deux ports auxquels on applique également des actions de lecture et d'écriture. Cette interface est détaillée dans la section 5.2.1.4.2.2.

Dans la suite, nous définirons le choix de langage de modélisation et nous détaillerons l'interface de communication.

5.2.1.4.2.1 Choix du langage Simulink

Matlab/simulink est un langage de modélisation et de simulation des systèmes manipulant d'une manière intense des données. Il permet aussi l'analyse des systèmes dynamiques évoluant au cours du temps (les données et les états du système dépendent du temps), ceci en offrant une bibliothèque d'objets sous forme de blocs standards paramétrables (tels que des multiplieurs, des additionneurs, des intégrateurs, etc.). L'environnement Matlab/Simulink offre en plus une interface graphique permettant une manipulation facile de ces objets.

Il offre aussi des concepts qui permettent une modélisation simple et rapide des systèmes. Ces concepts sont :

- La hiérarchie : un système est formé par un ensemble de blocs pouvant contenir d'autres blocs. Un bloc feuille (noté B_f) est défini comme étant la réunion des entrées, des sorties et de ses états internes. B_f = ({entrées} U {sorties} U {états}).
- La concurrence : la concurrence d'exécution entre les blocs est exprimée implicitement. L'exécution des blocs est réalisée dès que les données d'entrées sont disponibles pour générer des données de sorties. Tous les blocs ayant leurs données disponibles en même temps seront exécutés en parallèle.
- La communication entre les blocs Simulink est exprimée graphiquement par des liens (traits) entre eux. La communication se fait à l'aide de la modélisation des signaux comme étant les valeurs de sorties. Ces valeurs possèdent des types prédéfinis et scalaires. Les types scalaires utilisés sont « entiers », « réels » et « booléens ». La communication entre les différents ports est similaire à un envoi de message synchrone (Rendez-Vous). Les blocs émetteur et récepteur se synchronisent pour effectuer un échange de données.
- Le média de communication : il est modélisé par un trait connectant deux blocs. Le média de communication entre les blocs est un canal qui peut être simple ou complexe. L'expression des médias spécifiques tels qu'un bus ou un réseau spécifique, se fait d'une manière explicite avec des blocs primaires.
- Synchronisation : la synchronisation entre les blocs est implicite.

5.2.1.4.2.2 Description de l'interface de communication

L'interface de communication du modèle du microsystème de génération de puissance représente les ports avec lesquels le modèle décrit en Simulink communique avec l'extérieur. Cette interface est formée par deux ports nommés « Pin_Cmd_C » et, « Pout_SOBat ». La description de cette interface est au niveau message. A ce niveau, les données sont de types scalaires. Elles ne sont pas au niveau cycle et ne sont pas guidées par un protocole donné.

La définition de cette interface par le langage Matlab/simulink est donnée par une déclaration des ports d'entrées/sorties. Ci-dessous un exemple de cette déclaration des deux ports d'entrées du système.

```
#define IN_PORT_0_NAME      Pin_Cmd_C
#define INPUT_0_WIDTH       1
#define INPUT_DIMS_0_COL    1
#define INPUT_0_DTYPE       real_T
#define INPUT_0_COMPLEX     COMPLEX_NO
#define IN_0_FRAME_BASED    FRAME_NO
#define IN_0_DIMS           1-D
#define INPUT_0_FEEDTHROUGH 1
#define OUT_PORT_1_NAME     Pout_SOBat
#define INPUT_1_WIDTH       1
#define INPUT_DIMS_1_COL    1
#define INPUT_1_DTYPE       real_T
#define INPUT_1_COMPLEX     COMPLEX_NO
#define IN_1_FRAME_BASED    FRAME_NO
#define IN_1_DIMS           1-D
#define INPUT_1_FEEDTHROUGH 1
```

Figure 5-5 Déclaration des ports d'entrées du modèle Simulink

Cette déclaration comporte la définition du nom du port logique (par exemple « IN_PORT_0_NAME » dans la Figure 5-5), le type (INPUT_X_DTYPE), la taille et la nature des données que l'on peut transmettre ou recevoir à travers ces ports.

Cependant, cette déclaration est insuffisante pour la définition de cette interface. Il est nécessaire d'utiliser une API d'instanciation de ces ports d'entrées/sorties. Cette API est « ssSetNumInputPorts(S, NUM_INPUTS) » pour les ports d'entrées. Pour les ports de sorties on utilise « ssSetNumOutputPorts(S, NUM_OUTPUTS) ».

D'autres API sont utilisées pour la manipulation de ces ports (comme l'écriture ou la lecture sur ces ports). Certaines APIs sont décrites dans le listing de la Figure 5-6.

```
- ssGetInputPortSignal(S,0); /* accéder aux données sur des ports d'entrées.
                                Ces données sont de type différents de « réel »*/
- ssGetInputPortRealSignal(S,0); /* pour des entrées de types réels*/
- ssGetOutputPortSignal(S,0);/* écrire des données sur des ports de sorties,
                                ces données sont de types différent de « réel »*/
- ssGetOutputPortRealSignal(S,0);/* pour des sorties de types réels*/
```

Figure 5-6 API Simulink pour la manipulation des ports

Dans le cas de notre application, on considère que les données envoyées sont de types « réels », ce qui impose l'utilisation des API relatives au type « réel ».

5.2.1.5 Vers la génération du modèle d'exécution global du sous-système du centre d'information d'un véhicule

Dans cette partie, nous présenterons le modèle d'exécution global du sous-système ainsi que le modèle à base de composants virtuels correspondant.

5.2.1.5.1 Modèle d'exécution global du sous-système de centre d'information d'un véhicule

Le modèle d'exécution global des sous-systèmes est une composition des modèles des composants décrits ci-dessus. Ce modèle est défini par deux éléments : l'environnement d'exécution et les adaptateurs. Ces deux éléments sont nécessaires pour la réalisation de la communication entre les différents modèles.

- **L'environnement d'exécution** : l'environnement d'exécution est l'interprétation de l'interconnexion. L'interconnexion entre ces deux modèles est la mise en correspondance des différents ports de différentes interfaces. La définition de l'interconnexion en tant que média, comportement et type de données est donnée comme suit :
    - Média : dans ce modèle, la communication est entre deux simulateurs différents (SystemC et Simulink). Pour ce type de communication, le média est défini en tant que canal hiérarchique. Il comporte deux modes de communication : un mode avec chaque composant. Pour le composant SystemC, le média est défini comme de simples signaux SystemC (voir 5.2.1.4.1.1), alors qu'avec le composant Simulink, le canal est défini par les IPC (Inter Process Communication). Il est de type mémoire partagée.
    - Comportement : le comportement du média est défini par la synchronisation de l'échange des données entre les différents simulateurs par des sémaphores.
    - Type de données : le type de données échangé est le type scalaire réel.
- **Les adaptateurs** : les adaptateurs sont la réalisation de l'interface abstraite (voir 5.2.1.5.2). Ils permettent l'adaptation entre les modèles d'exécution des composants et l'environnement d'exécution voir Figure 5-7.
    - Modèle SystemC : comme l'environnement d'exécution est décrit en SystemC et que les ports externes sont de type équivalent à celui des ports internes du composant SystemC, alors il n'est pas nécessaire de générer un adaptateur.
    - Modèle Simulink : il a été nécessaire de générer un adaptateur entre l'environnement de communication et le modèle en Simulink. Il existe une

incompatibilité entre les différents ports internes (ports simulink) et externes (ports IPC). La structure des adaptateurs est donnée selon le graphe de dépendance de données (voir chapitre 3). La réalisation de cet adaptateur est expliquée dans la section 5.2.1.5.1.1.

Le modèle d'exécution est donné par la figure ci-dessous.

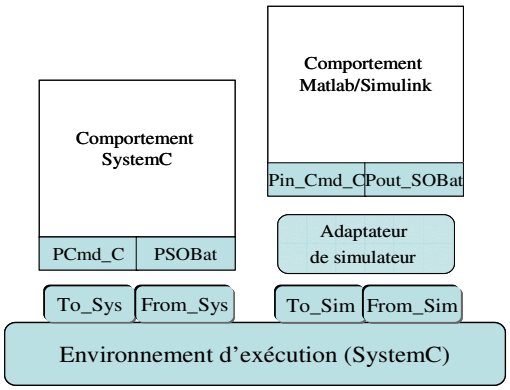


Figure 5-7 Modèle d'exécution global du sous-système du centre d'information d'un véhicule

5.2.1.5.1.1 Adaptateur entre le modèle Simulink et l'environnement d'exécution

Pour la génération de l'adaptateur, on utilise le graphe de dépendance de services défini dans le chapitre 3. Ce graphe consiste à définir une dépendance entre des modules offrants des services d'adaptation *via* leurs interfaces.

Pour la génération de l'adaptateur entre Simulink et SystemC, on utilise aussi des potentialités offertes par le modèle Matlab/Simulink pour la communication avec l'extérieur via les S_fonctions [Mat00].

La figure 5-8 illustre le graphe de dépendance de service entre les ports « Pin_Cmd_C » et « To_Sim ». Ces deux ports requièrent et fournissent des services à travers leurs « SAP » respectifs. Pour le port « To_Sim », les services sont des services de lecture et d'écriture dans la mémoire partagée. Pour le port « Pin_Cmd_C », les services sont implicites aussi, ce sont le service d'initialisation des paramètres et le service de calcul des sorties et des états internes.

L'adaptateur, entre ces deux ports, comprend alors des éléments offrant des services relatifs aux S_fonctions tels que l'initialisation, et le service de calcul des sorties et des états internes. Le service d'initialisation nécessite un service interne de Simulink : « Solver Simulink » qui est en relation avec l'Ordonnanceur Simulink. D'autres services sont requis pour l'accès aux données envoyées par SystemC *via* la mémoire partagée. Ces services sont : l'attachement à la mémoire partagée, la lecture/écriture de la mémoire partagée et la synchronisation.

Le service de calcul des sorties et des états internes est nécessaire pour le traitement des données envoyées par SystemC (par exemple, conversion de type de données et adaptation de protocole si nécessaire).

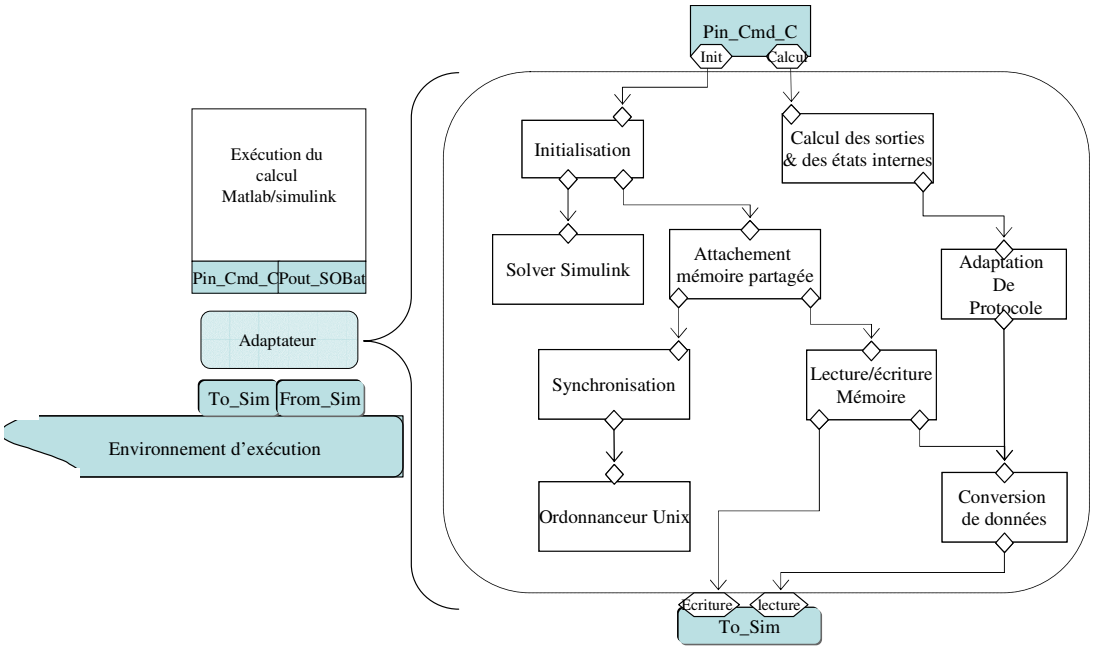


Figure 5-8 Adaptateur entre Simulink et SystemC

5.2.1.5.2 Le modèle à base de composants virtuels du sous-système de centre d'information d'un véhicule

Le modèle à base de composants virtuels de cette application est défini comme étant l'assemblage de deux modèles d'exécution existant enveloppés dans des interfaces abstraites communiquant via un environnement d'exécution bien déterminé. L'interface abstraite, comme décrite dans le chapitre 3, est un ensemble de ports internes et de ports externes. Les ports internes sont relatifs aux ports des différents composants (voir 5.2.1.3). Dans cet exemple, deux interfaces abstraites sont définies, une interface pour le composant du microprocesseur et une pour le microsystème de génération de puissance.

L'interface abstraite du microprocesseur comporte une interface interne et une interface externe. L'interface interne est constituée par les ports logiques « PCmd_C » et, « PSOBat ». Comme il a été défini dans 5.2.1.5, l'environnement d'exécution est décrit en SystemC. L'interface externe relative à cet environnement d'exécution est constituée de deux ports logiques aussi nommés « To_Sys » et « From_Sys ». Ces ports sont des ports de même nature et véhiculent le même type de données que celles de l'interface interne.

L'interface du microsystème de génération de puissance définit comme interface interne l'ensemble de ports notés « Pin_Cmd_C » et, « Pout_SOBat ». Ce sont des ports logiques

Simulink. L'interface externe relative à l'environnement d'exécution, comme défini dans 5.2.1.5.1, est un ensemble de ports spécifiques pour la communication entre langages différents. Ce sont des ports logiques aussi de type IPC (Inter Process Communication). En effet, les IPC représentent un moyen de communication standard pour la communication entre processus Unix différents. Ce dernier point, dans notre cas, est utile puisque les différents simulateurs seront traités comme des processus unix. Parmi les IPC, on cite les mémoires partagées avec le jeu des sémaphores pour leur synchronisation, les tubes, les signaux etc.

La figure suivante montre le modèle à composants virtuels de l'application.

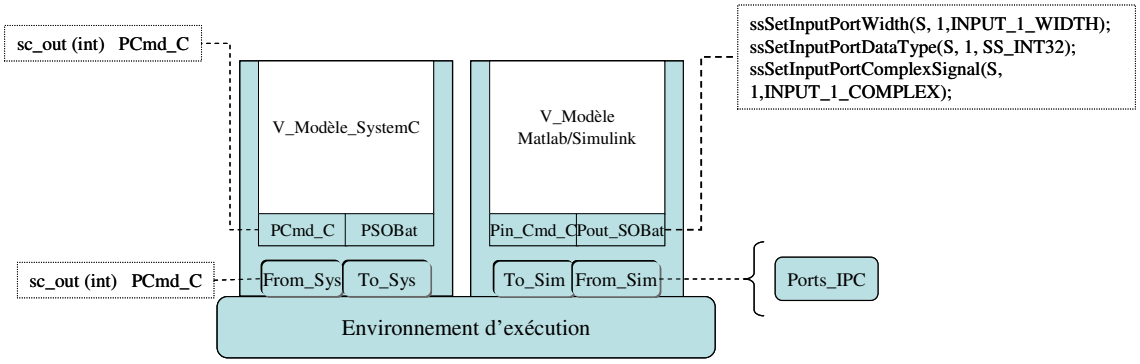


Figure 5-9 Modèle à composants virtuels de l'application

L'association entre les ports internes et les ports externes est effectuée à ce niveau. Le tableau suivant résume cette mise en correspondance.

Composant Virtuel	**Mise en correspondance** (port interne **Θ** port externe)
V_Modèle SystemC	PCmd_C **Θ** From_Sys
	PSOBat **Θ** To_Sys
V_Modèle Matlab/Simulink	Pin_Cmd_C **Θ** To_Sim
	Pout_SOBat **Θ** From_Sim

Tableau 3 Mise en correspondance entre les ports internes et les ports externes du sous-système du centre d'information d'un véhicule

5.2.2 Résultats et perspectives

Dans cet exemple, nous avons défini et réalisé un modèle d'exécution global pour un sous-système du centre d'information de véhicule (avec une description multi langages de ces différents composants). Le modèle généré est conforme aux concepts définis dans le chapitre 3.

Pour ce modèle, nous avons pu générer des adaptateurs SystemC_Simulink sous forme d'un graphe de dépendance de services. De plus, une définition de l'environnement d'exécution entre simulateurs différents a bien été déterminée comme une interprétation d'une interconnexion définie par un média, un comportement et les types de données transmises.

Les perspectives pour cette application est de terminer la modélisation du centre d'information d'un véhicule en ajoutant les modèles des autres composants (capteurs et autres applications logicielles, etc.).

5.3 Modélisation d'une chaîne de traitement radio

5.3.1 Présentation générale de l'application de la chaîne de traitement radio : la chaîne audio 802.16

L'application choisie appartient au domaine de la Radio Logicielle. Une Architecture de type « Radio Logicielle » est définie comme un système logiciel distribué et embarqué, sur lequel s'exécutent en temps réel divers applicatifs associés au codage et/ou décodage du protocole radio (GSM, GPRS, UMTS) [Fou05]. Elle se caractérise par de fortes contraintes d'implantation (temps réel, consommation, …) et se traduit généralement par une architecture hétérogène basée sur des ressources de calcul telles que : Processeur d'usage général (GPP), processeur(s) de traitement de signal (DSP), voire structures reconfigurables.

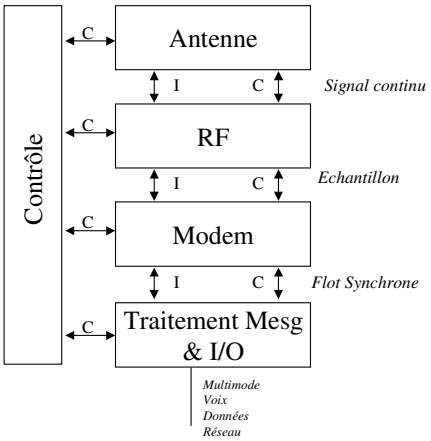


Figure 5-10 Chaîne de traitement radio

Dans le cadre du déploiement de nouveaux systèmes de transmissions haut débit sans fil de nouvelles normes se sont développées. L'application choisie utilise la norme 802.16. La norme 802.16 spécifie l'interface, y compris la couche d'accès (MAC) et la couche physique (PHY), d'accès point-à-multipoint des systèmes BWA, (de l'anglais Broadband Wireless Access), fournissant des services multiples. La Figure 5-10 présente les principaux composants de la chaîne de traitement radio.

Dans le cadre de cette application, les travaux de modélisation se feront sur le composant Modem qui regroupe l'ensemble des traitements associés à la modulation et à la démodulation du protocole radio, tant du point de vue traitement de données que du contrôle. Il s'agit plus spécifiquement des traitements situés entre la numérisation des échantillons acquis (I,Q) en bande de base et les bits démodulés servant à la constitution des messages.

Le **but de l'étude** est de permettre de définir une architecture du modem en partant d'une modélisation de haut niveau et en raffinant ses différents composants jusqu'à l'implémentation finale, en tenant compte, bien sûr, de certaines contraintes d'environnement (consommation, temps réel, etc.) et de contraintes d'implémentation matérielle (hétérogénéité : GPP, DSP, reconfigurable, etc.) et logicielle (hétérogénéité des systèmes d'exploitation, par exemple: Linux pour les GPP, DSPBios pour les DSP, etc.).

Une première modélisation à haut niveau (fonctionnelle) de l'application Modem est réalisée et servira de modèle de départ. Dans un second temps un raffinement de certains composants sera effectué, ainsi que des validations, afin de définir une meilleure architecture.

Définir un modèle d'exécution global est nécessaire pour vérifier le bon fonctionnement à chaque étape du raffinement et de valider les choix d'implémentation par comparaison avec les résultats de références [Fou05].

5.3.2 Présentation de l'application Modem

L'application Modem est donnée par la Figure 5-11. Elle présente un modèle de référence de la chaîne émission/réception. Elle se compose des composants suivants :

- Le composant « Data Source » génère une succession de bits aléatoires par bursts (pour la modélisation des sons). C'est un composant élémentaire.
- Le composant « Transmitter », transforme les bits reçus du composant « Data Source » en un signal équivalent à celui émis à l'antenne. Ce module est un module hiérarchique. Sa composition est développée dans la section 5.3.2.1.
- Le composant « Channel », modélise les effets du canal hertzien sur les données transmises : bruit gaussien, multi trajet, évanouissements de Rayleigh. Il reçoit des données du « Transmitter » pour les envoyer au « Receiver ». C'est un composant élémentaire.
- Le composant « Receiver » reçoit les données envoyées par le canal et décrypte le signal pour transformer le signal analogique en bits. C'est un composant hiérarchique et sa composition est expliquée dans la section 5.3.2.2.
- Les composants « Sink » et « Compare » sont des composants élémentaires de test. Leur rôle est de comparer les deux signaux d'émission et de réception pour valider le bon transfert.

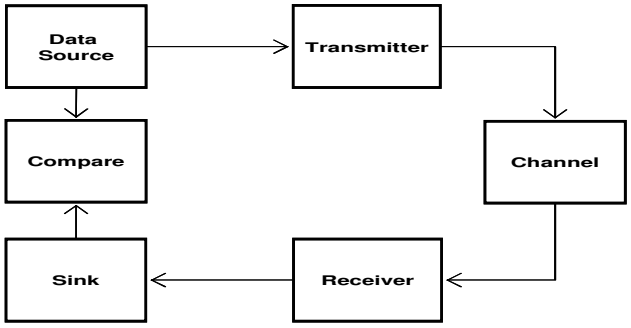


Figure 5-11 Composants de l'application Modem

5.3.2.1 Structure de l'émetteur

La partie émission, notée par « Transmitter », est composée des modules élémentaires suivants :

- Le composant **« Coder »** introduit une redondance dans les données afin de pouvoir retrouver facilement les données émises même en cas de perturbation sur le canal radio. Dans le cas du modem 802.16, pour chaque bit émis, le codeur produit deux bits calculés avec deux polynômes binaires bien choisis. Le module reçoit des matrices de vecteurs, de dimensionnement 1 x nb de bits émis, et produit en sortie des matrices de dimension 2 x nb de bits émis.
- Le composant « **Π »**, mélange les données selon une méthode bien précise de façon à diminuer le nombre de symboles identiques successifs, ce qui réduit l'énergie nécessaire à la transmission radio.
- Le composant **Mapping** transforme les bits en «symboles», valeurs complexes calculées d'après un schéma appelé constellation.
- Le composant « **IFFT »**, de transformée de Fourier inverse, réalise la conversion entre le domaine des fréquences et le domaine temporel nécessaire à la sortie du signal dans le monde extérieur.

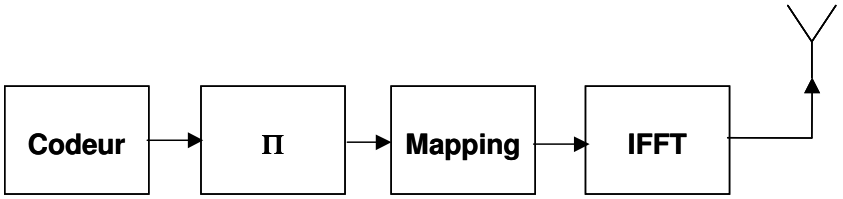


Figure 5-12 Structure de l'émetteur

5.3.2.2 Structure du récepteur

Le composant de réception, noté « Receiver » précédemment, est décrit par la figure suivante :

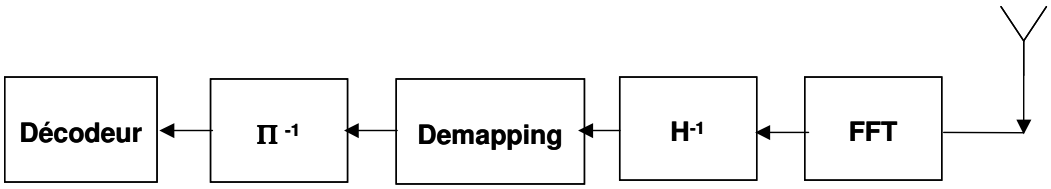


Figure 5-13 Structure du Récepteur

- Il présente une structure en partie symétrique de l'émetteur. Il est constitué des modules élémentaires suivants.
- Le composant « **FFT »** réalise la transformée de Fourier. Il réalise des opérations annulant les traitements correspondants effectués par l'émetteur.
- Les composants « $\mathbf{\Pi^{-1}}$ » (présentant le bloc de suppression des pilotes) et le composant « **FFT »**, réalisent les opérations annulant les traitements correspondants effectués par l'émetteur.
- Le composant « $\mathbf{H^{-1}}$ **»,** égaliseur de canal, fait l'opération «d'inversion de canal» pour annuler certains effets du canal hertzien à l'aide de mesures du profil fréquentiel du canal.
- Le module **« Demapping »** qui a pour fonction de retrouver dans la constellation à quel bit correspond le symbole reçu. L'opération de « Demapping » fait l'une des particularités du récepteur, car au lieu de fournir des bits en sortie correspondant aux symboles reçus, il calcule une probabilité de recevoir une valeur de bit.
- Le « **Décodeur »** est le dernier module avant la sortie du récepteur, il lui revient donc de prendre la décision finale sur la valeur des bits reçus. Cette décision est réalisée par un algorithme de «Viterbi».

5.3.3 Vers la génération d'un modèle d'exécution de la chaîne 802.16 en SystemC

Le but de cette application est de générer un modèle d'exécution global par assemblage de composants, de la chaîne audio, décrits à différents niveaux d'abstraction en utilisant le même langage de spécification, SystemC. Tous les composants sont fonctionnels et décrits à un très haut niveau d'abstraction (voir chapitre 3). Un des composants (modules) sera raffiné vers un niveau d'abstraction plus bas. Le module en question est la « FFT » de « Receiver ». Le

raffinement de la « FFT » permettra une exploration d'architecture pour le choix de son implémentation matérielle ou logicielle.

La génération du modèle d'exécution global de cette application est réalisée en deux étapes :

- La première étape : l'implémentation des différents composants est fonctionnelle et la communication est au niveau transaction, 3.2.1.3. Cette implémentation est réalisée en se basant sur une description avec le processus de Khan. Ce processus sera décrit dans la section 5.3.3.1. Le long de cette étape, nous explicitons le modèle d'exécution global correspondant ainsi que le modèle à base de composants virtuels. Ces modèles seront détaillés dans la section 5.3.3.2.1.1.
- La deuxième étape consiste en la réalisation du modèle d'exécution après raffinement de la « FFT » uniquement. Pendant cette étape, nous définissons le modèle d'exécution global ainsi que celui à base de composants virtuels (dans la section 5.3.3.3).

5.3.3.1 Spécification fonctionnelle de l'application

La spécification système à un niveau fonctionnel doit comporter les contraintes fonctionnelles et non fonctionnelles (telles que les contraintes temporelles, tolérances aux pannes, performances, etc.) imposées à celui-ci. Ces contraintes sont utilisées lors de l'exploration architecturale pour évaluer la pertinence et la qualité d'une réalisation par rapport au problème. Alors que les contraintes fonctionnelles quant à elles décrivent les dépendances entre tâches et les conditions d'exécution de celles-ci, ces contraintes s'expriment à travers un formalisme modélisant l'application.

Pour notre application, le modèle choisi est de type KPN, Kahn Process Network. En effet, le modèle de Kahn permet de représenter efficacement des algorithmes de traitement du signal. Ce modèle convient parfaitement alors à notre application de traitement de son.

Le KPN découpe l'algorithme en *processus concurrents* communiquant par l'intermédiaire de *files d'attentes* notées FIFO. Les « FIFO » transportent des unités d'informations élémentaires appelées jetons. Chaque processus est indépendant et ne peut interférer avec un autre (Figure 5-14).

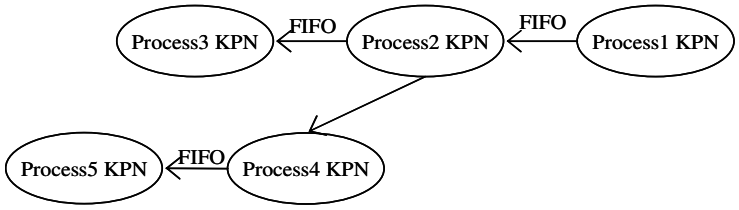


Figure 5-14 Modèle KPN

L'implémentation de notre application doit être équivalente au modèle de Khan. Pour cela, SystemC s'avère être un langage adéquat permettant la description de ce modèle. Ainsi, les processus du modèle KPN sont remplacés par des modules SystemC (sc_module). Chaque module peut être élémentaire ou hiérarchique. Les modules élémentaires possèdent un contenu noté par « SC_THREAD » et une interface qui permet la communication avec l'extérieur. Cette interface, prédéfinie dans le chapitre 1, est un ensemble de ports. Dans notre cas, ce sont des ports SystemC notés « sc_port ».

L'interconnexion entre les différents composants est la mise en correspondance des ports des différents modules (voir 2.2.4). Elle est définie par le média, le comportement et le type de données.

- Média : le média est un canal décrit, à haut niveau d'abstraction, noté « sc_fifo » prédéfini en SystemC. Normalement, les FIFO sont unidirectionnelles et non bornées pour le KPN mais pour des contraintes d'implémentation, la profondeur des FIFO est limitée.
- Comportement : le comportement de ces canaux modélise le transfert d'un flot de données circulant dans chaque FIFO déterminé uniquement par les données en entrées. Ce flot de données est contrôlé par des méthodes de lecture/écriture bloquantes.
- Type de données : les données transmises dans les FIFO sont des matrices de complexes, modélisant l'onde devant être transporter de l'émetteur vers le récepteur.

5.3.3.2 Modèles d'exécution global de la chaîne 802.16

Dans cette partie, les différents modèles d'exécution de la chaîne 802.16 sont définis. Le modèle d'exécution global tel qu'il était défini dans le chapitre 3 est vu comme un assemblage des différents modèles d'exécution des composants. C'est un modèle dans lequel on pourra réaliser la communication entre les différents composants considérés. Il doit être capable de prendre en compte les différents concepts manipulés par chaque élément du système.

Dans un premier temps, le modèle d'exécution global de l'application est défini à partir d'une description en SystemC des différents modules au même niveau d'abstraction (transaction) (voir chapitre3) et même protocoles de communication (FIFO). Le deuxième modèle comporte le module « FFT » décrit en SystemC au niveau RTL, alors que les autres modules sont restés inchangés. Ces deux études seront détaillées dans les sections suivantes.

5.3.3.2.1 Modèle d'exécution global de la chaîne 802.16 décrit en SystemC avec une description au même niveau d'abstraction des différents composants

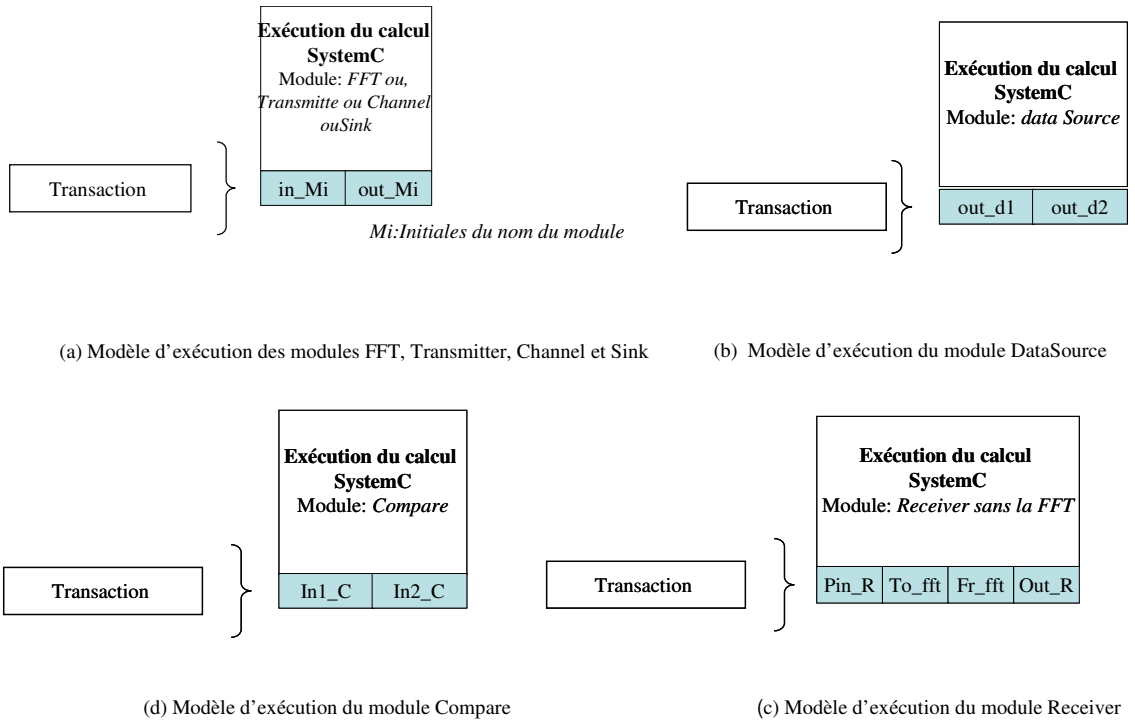
Dans cette partie, nous décrivons le modèle d'exécution global de la chaîne par assemblage de ceux des composants. Les modèles correspondant aux composants sont détaillés

dans la section 5.3.3.2.1.1 **.** L'assemblage de ces modèles est décrit dans la section 5.3.3.2.1.2. Dans cette partie, nous nous intéresserons au modèle à base de composants virtuels et au modèle global en spécifiant l'environnement d'exécution et les adaptateurs.

5.3.3.2.1.1 Modèle d'exécution des composants

A ce stade de la conception, la décision d'une implémentation logicielle ou matérielle des différents composants de la chaîne n'est pas prise. Pour spécifier alors simplement le fonctionnement, on choisit une implémentation de ces composants en SystemC. Le modèle d'exécution correspondant à tous les composants est alors un modèle d'exécution fonctionnel.

Le modèle d'exécution de ces composants comporte une exécution du calcul qui sera en SystemC avec une description des interfaces de communications par des ports logiques (voir chapitre 3). Ce modèle est décrit par la Figure 5-15.



(a) Modèle d'exécution des modules FFT, Transmitter, Channel et Sink

(b) Modèle d'exécution du module DataSource

(d) Modèle d'exécution du module Compare

(c) Modèle d'exécution du module Receiver

Figure 5-15 Modèle d'exécution fonctionnel des composants de l'application de la chaîne 802.16

L'exécution du calcul représente une interprétation algorithmique de son fonctionnement (voir partie descriptive de l'application) décrite en SystemC.

5.3.3.2.1.1.1 Description de l'interface de communication du modèle d'exécution des composants de la chaîne audio

L'interface de la communication est l'ensemble des ports logiques (voir chapitre1) permettant la communication avec l'extérieur. Le protocole de la communication (FIFO) et la

topologie des interconnexions (Point à Point) étant fixés, le niveau d'abstraction de cette interface est donc le niveau transaction. Pour chaque composant, l'interface est constituée de deux ports. Un port d'entrée et un port de sortie pour chacun des modules « Transmitter », « Channel », « Receiver », et « Sink ». Le module « DataSource » contient deux ports de sorties alors que le module « Compare » possède deux ports d'entrées. Les sous modules du « Transmitter » et « Receiver » ont tous un port d'entrée et un port de sortie. Les actions sur les ports d'entrées sont la lecture sur ce port. Tandis que les actions associées aux ports de sorties sont l'écriture. Il faut noter que pour étudier cette application à différents niveaux d'abstraction, le module « FFT » du « Receiver » a été extrait. L'interface de ce dernier a été modifiée de telle sorte que la communication avec la FFT puisse apparaître.

Les différents détails de l'interface des composants sont donnés dans le tableau suivant.

Composants	**Ports logiques**		
	Nom	**Caractéristiques**	**Services sur le port**
DataSource	Out_d1	Port de sortie envoyant les données vers le port d'entrée « In1_C » du « Compare »	Ecriture
	Out_d2	Port de sortie envoyant les données vers le port d'entrée « in_T » du « Transmitter »	Ecriture
Transmitter	In_T	Port d'entrée recevant les données du port de sortie « Out_d1 » du « DataSource »	Lecture
	Out-T	Port de sortie envoyant les données vers le port d'entrée « In_Ch » du « Channel »	Ecriture
Channel	In_Ch	Port d'entrée recevant les données du port de sortie « Out_T » du « Transmitter »	Lecture
	Out_Ch	Port de sortie envoyant les données vers le port d'entrée « In_R » du « Receiver»	Ecriture
Receiver sans FFT	In_R	Port d'entrée recevant les données du port de sortie « Out_Ch » du « Channel »	Lecture
	Out_R	Port de sortie envoyant les données vers le port d'entrée « In_S »du « Sink »	Ecriture
	To_fft	Port de sortie envoyant les données vers le port d'entrée « In_fft »de la « FFT »	Ecriture
	Fr_fft	Port d'entrée recevant les données du port de sortie« out_fft »de la« FFT »	Lecture
FFT	In_fft	Port d'entréé recevant les données du port de sortie « To_fft » du receiver	Lecture
	Out_fft	Port de sortie envoyant les données vers le port d'entrée « fr_fft » du « receiver »	Ecriture
Sink	In_S	Port d'entrée recevant les données du port d'entrée « Out_R »du « Receiver»	Lecture
	Out_S	Port de sortie envoyant les données vers le port d'entrée « In2_C »du « Compare»	Ecriture
Compare	In1_C	Port d'entrée recevant les données du port d'entrée «Out_d1» du « DataSource »	Lecture
	In2_C	Port d'entrée recevant les données du port d'entrée « Out_d2 » du « Sink »	Lecture

Tableau 4 Description des composants et de leurs interfaces

L'exemple suivant présente la description de l'interface du module « FFT » au niveau transaction en SystemC (Figure 5-16).

```
SC_MODULE(FFT)
{
// Interface
  sc_port<sc_fifo_in_if<Mat> >  in_fft;
  sc_port<sc_fifo_out_if<Mat> > out_fft;
// Comportement
```

Figure 5-16 Description de l'interface de la FFT au niveau Transaction en SystemC

5.3.3.2.1.2 Modèle d'exécution global pour la description au même niveau d'abstraction

Le modèle global est l'assemblage des différents modèles décrits ci-dessus. Il est donné par la figure suivante.

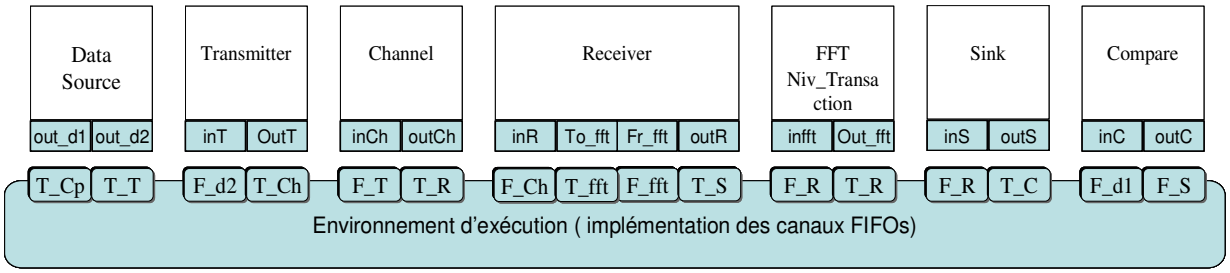


Figure 5-17 Modèle d'exécution du modèle homogène de la chaîne 802.16

Les caractéristiques de ce modèle sont :

- **Les adaptateurs** : à ce niveau de génération du modèle d'exécution global, les adaptateurs ne sont pas nécessaires. En effet, les différents composants sont décrits au même niveau d'abstraction et communiquent avec le même protocole de communication, c'est pourquoi nous n'avons pas besoin de générer des adaptateurs pour la communication entre les différents éléments.
- **L'environnement d'exécution** : il est défini comme une interprétation des canaux FIFO. Il est décrit en SystemC. L'interface de cet environnement d'exécution est définie comme étant des ports compatibles aux différents ports des différents composants de la chaîne. Son interface est décrite par les ports externes dans le Tableau 5. La définition de l'environnement d'exécution en tant que média, comportement et type de données est donnée comme suit :

- Média : dans ce modèle, la communication est entre composants SystemC décrit au même niveau d'abstraction (transaction). Pour ce type de communication, le média est défini en tant que canal abstrait à l'aide de primitives SC_CHANNEL (voir 5.2.1.4.1.1).
- Comportement : le comportement du média est défini par le protocole FIFO.
- Type de données : le type de données échangé est un type défini par une matrice de nombres complexes.

Il est à noter que pour tous les systèmes, nous décrivons le modèle à base de composants virtuels même s'ils sont décrits avec le même langage et au même niveau d'abstraction. Le modèle à base de composants virtuels dans ce cas, est donné par la Figure 5-18.

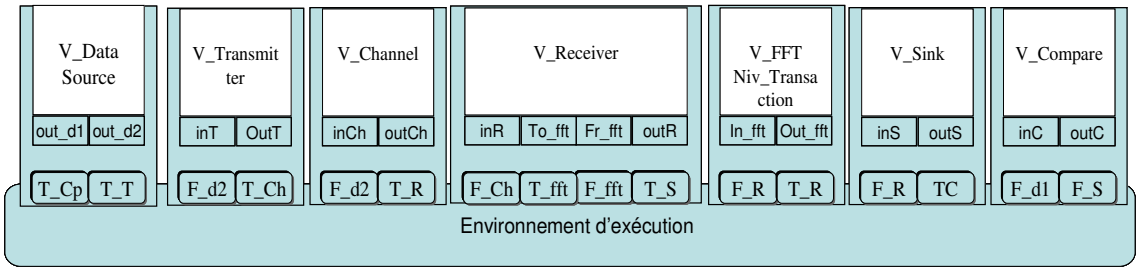


Figure 5-18 Modèle à base de composants virtuels du modèle homogène de la chaîne 802.16

Chaque composant est enveloppé par une interface abstraite. Cette interface, définie dans le chapitre 4, est un ensemble de ports internes et de ports externes. Les ports internes sont relatifs aux modules (voir Tableau 4). Les ports externes sont relatifs à l'environnement d'exécution qui permettra la réalisation de la mise en correspondance entre les ports des différents modules. Afin de déterminer le modèle à base de composants virtuels, il nous a fallu identifier les ports relatifs à l'environnement d'exécution qui seront au même niveau d'abstraction et supporteront le même protocole de communication (FIFO), voir Figure 5-18.

Après l'identification, une mise en correspondance entre les ports internes et externes est mise en place. Le tableau suivant résume cette correspondance. Le symbole de mise en correspondance est noté Θ.

Composant Virtuel	**Mise en correspondance** (port interne Θ port externe)
DataSource	Out_d1 **Θ** T_Cp
	Out_d2 **Θ** T_T
Transmitter	In_T **Θ** F_D1
	Ou_T **Θ** T_Ch
Channel	In_Ch **Θ** F_T
	Out_Ch **Θ** T_R
Receiver	In_R **Θ** F_Ch
	Out_R **Θ** T_S
	Fr_fft **Θ** F_fft
	To_fft **Θ** T_fft
FFT(niv.transaction)	In_fft **Θ** F_R
	out_fft **Θ** T_R
Sink	In_S **Θ** F_R
	Out_S **Θ** T_C
Compare	In1_C **Θ** F_D1
	In2_C **Θ** F_S

Tableau 5 Mise en correspondance entre les ports internes et les ports externes des composants virtuels

5.3.3.3 Modèle d'exécution global de la chaîne 802.16 à partir d'une description multi-niveaux

Dans cette partie, nous définissons un modèle d'exécution global du système de la chaîne 802.16 dont les composants sont décrits à différents niveaux d'abstraction. Un des composants est raffiné vers un niveau d'abstraction plus bas. La description des autres composants est restée invariable. Le composant raffiné est le module « FFT » du « Receiver ».

Par la suite, le modèle d'exécution du composant « FFT » va être détaillé ainsi que la définition du modèle d'exécution global de tout le système en se basant sur la description à base de composants virtuels.

5.3.3.3.1 Modèle d'exécution de la « FFT » au niveau RTL

L'implémentation de la « FFT » n'est pas encore déterminée. Cependant, son raffinement vers un niveau plus détaillé permettra la mesure des performances et une exploration d'architecture. Son modèle d'exécution est toujours fonctionnel. Cependant, au niveau du comportement et de l'interface, le cycle d'horloge est défini comme une unité temporelle pour le calcul et la communication. Les données manipulées sont décrites au niveau « bit ». Ainsi, on peut déduire que le niveau d'abstraction de l'interface est le niveau RTL. Le modèle d'exécution de la FFT est donné par la Figure 5-19.

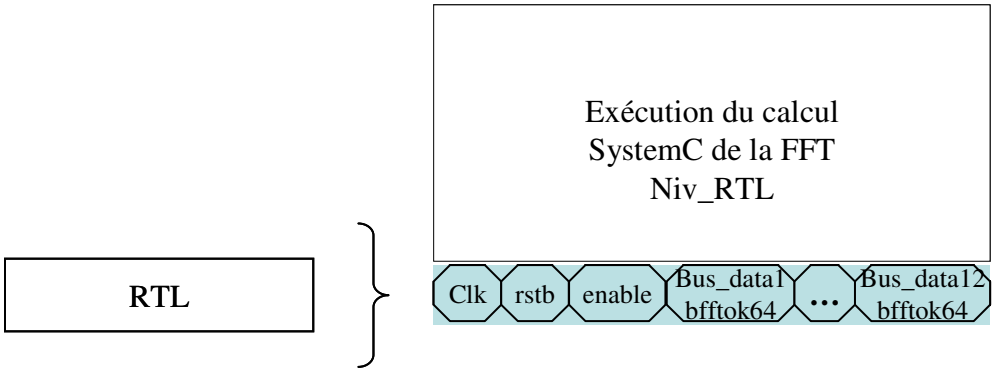


Figure 5-19 Modèle d'exécution fonctionnel au niveau RTL de la FFT

L'interface de communication est un ensemble de ports physiques. Les différents ports sont des ports spécifiques aux contrôles de la transmission et de la réception des données. Ces ports sont « Clk », « rstb » et « enable ». De plus, il existe des bus de données décrits au niveau physique. Ils sont au nombre de douze, notés « Bus_Data_i_bfftok64 », i=1..12. La Figure 5-20 décrit l'interface du composant FFT en SystemC.

```
SC_MODULE(FFT_RTL) {
  /* Interface : Ports physiques */
  sc_inout<sc_lv<16> > bus_data_1_bfftok64;
  sc_inout<sc_lv<16> > bus_data_2_bfftok64;
  sc_inout<sc_lv<16> > bus_data_3_bfftok64;
  sc_inout<sc_lv<16> > bus_data_4_bfftok64;
  sc_inout<sc_lv<16> > bus_data_5_bfftok64;
  sc_inout<sc_lv<16> > bus_data_6_bfftok64;
  sc_inout<sc_lv<16> > bus_data_7_bfftok64;
  sc_inout<sc_lv<16> > bus_data_8_bfftok64;
  sc_inout<sc_lv<16> > bus_data_9_bfftok64;
  sc_inout<sc_lv<16> > bus_data_10_bfftok64;
  sc_inout<sc_lv<16> > bus_data_11_bfftok64;
  sc_inout<sc_lv<16> > bus_data_12_bfftok64;
  sc_in<sc_logic> enable;
  sc_in<sc_logic> clk;
  sc_in<sc_logic> rstb;

/* Comportement : déclarations et instanciations des composants de la FFT niveau
RTL*/
```

Figure 5-20 Description de l'interface de la FFT au niveau RTL en SystemC

En ce qui concerne les autres composants, ils ont le même modèle d'exécution que celui décrit dans la section 5.3.3.2.1.1**.**

5.3.3.3.2 Modèle d'exécution global pour la description multi niveaux de la chaîne 802.16

Dans cette partie, nous définissons le modèle d'exécution global de la description multi niveaux de l'application. Il est défini par l'assemblage du modèle de la « FFT » et du reste du système. Il est donné par la figure suivante. Pour des raisons de clarté, l'interface RTL du composant « FFT » n'est pas détaillée.

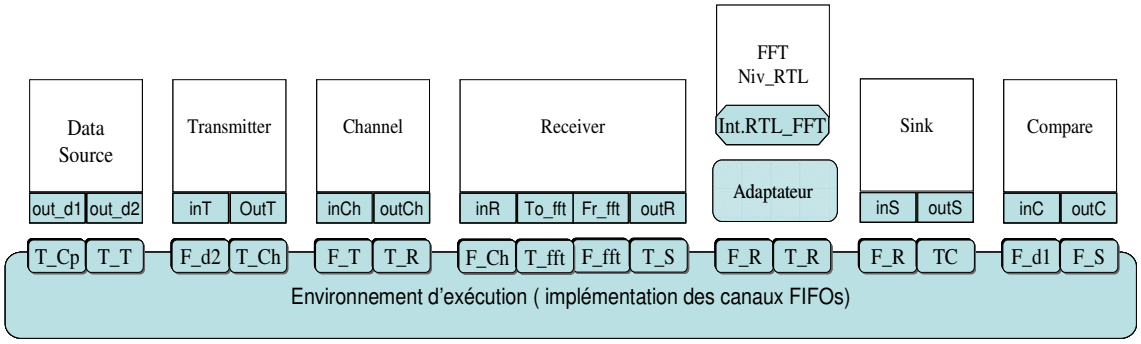


Figure 5-21 Modèle d'exécution global de la description multi niveaux de la chaîne 802.16

Les caractéristiques de ce modèle sont :

- **L'environnement d'exécution** : l'environnement d'exécution, dans ce cas, est toujours vu comme une implémentation des canaux FIFO du modèle de Khan. Cette implémentation est restée inchangée pour conserver la sémantique d'exécution du processus de Khan. Il a été nécessaire de redéfinir son interface. Ceci sera expliqué ultérieurement.
- **L'adaptateur** entre la « FFT » et le reste du système seulement : la différence du niveau d'abstraction, du protocole de communication entre la « FFT » et le reste du système nécessite une adaptation pour réaliser la communication entre les différents composants. L'adaptateur est réalisé selon la configuration donnée dans le chapitre3.

5.3.3.3.2.1.1 Adaptateur entre les différents niveaux d'abstraction (RTL et Transaction)

L'adaptateur dans cette application est défini entre le composant « FFT » décrit au niveau « RTL » et le reste du système décrit au niveau « transaction ». Cet adaptateur est décrit par le graphe de dépendance de services. Cet adaptateur est l'interprétation des mises en correspondance entre les ports internes de la « FFT » et les ports externes de l'environnement d'exécution. La Figure 5-22 montre le graphe de dépendance des services entre l'interface de la FFT et le port « T_R » de l'environnement d'exécution.

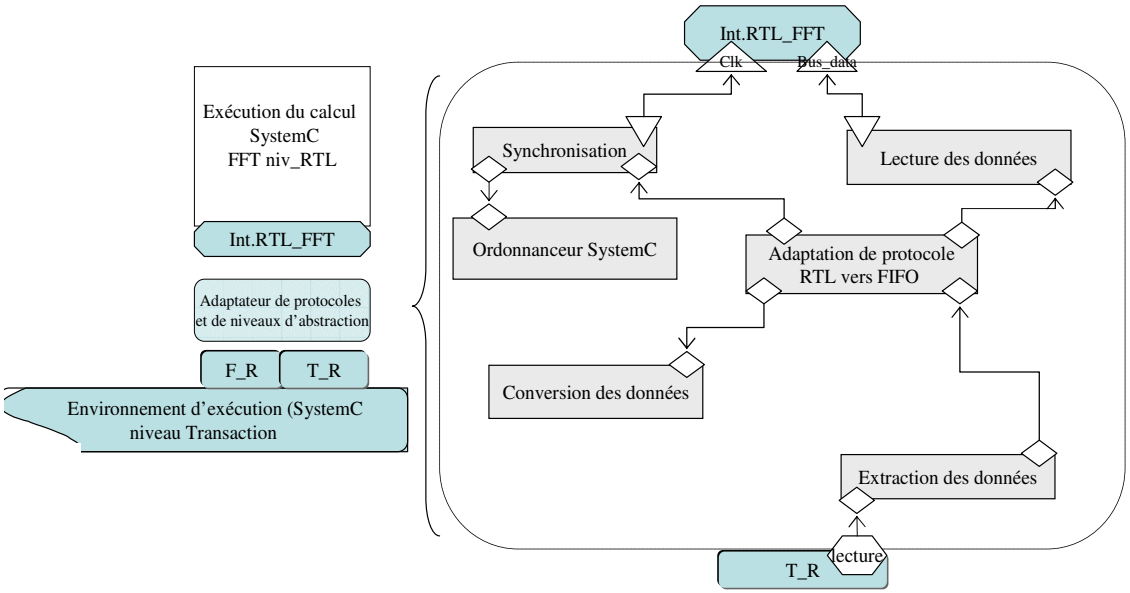


Figure 5-22 Adaptateur entre la FFT niveau RTL et l'environnement d'exécution niveau Transaction

L'architecture de cet adaptateur permet de définir les différents éléments et services nécessaires pour l'adaptation entre les niveaux d'abstraction et les protocoles.

Les ports internes de la « FFT » sont des ports physiques. Pour y accéder, on utilise les points d'accès aux ports notés « PAP ». Les ports externes de l'environnement d'exécution sont des ports logiques. Pour y accéder, on utilise les points d'accès aux services notés « SAP » voir 4.5.3.1.2.

Le SAP associé au port « T_R » de l'environnement d'exécution est la « lecture ». Les données transmises de la « FFT » vers le « Receiver » sont transportées à travers ce port.

L'adaptateur doit contenir les différents éléments pouvant transporter les données de la FFT vers le port « T_R ». Pour cela, on définit les éléments offrants les services suivants :

- Extraction des données : ce service est nécessaire pour lire les données de l'interface RTL de la « FFT ». Cependant, le protocole, qui lui est lié, est le protocole FIFO. Ainsi, l'élément correspondant nécessite un service d'adaptation de protocole.
- Adaptation de protocole RTL vers FIFO : ce service permet l'adéquation entre le protocole de la FFT au niveau RTL vers le protocoles FIFO. Pour cela trois autres services sont nécessaires : conversion des données, synchronisation et lecture des données de la « FFT ».
- Conversion des données : Au niveau Transaction, on manipule des structures de données prédéfinies (matrices de réels) alors qu'au niveau RTL, on manipule des données physiques (des bits). Il est donc nécessaire d'avoir un élément qui fournira l'adaptation des différentes données.

- Lecture des données : afin d'avoir les résultats de la « FFT », il est nécessaire d'accéder aux ports RTL via leur « PAP » pour lire les données sur ces ports notés dans la figure ci-dessus « Bus_data ».
- La synchronisation : ce service est utile parce qu'au niveau RTL, on utilise une horloge physique et au niveau Transaction, on utilise une horloge logique. Un cycle d'horloge logique peut correspondre à plusieurs cycles d'horloge physique. Ainsi, il faut synchroniser les différentes actions de lecture et écriture des données. Cet élément nécessite bien sûr l'Ordonnanceur SystemC. De plus cet élément de synchronisation aura besoin d'un accès à l'horloge physique de la FFT via le « PAP » de l'horloge noté dans la figure ci-dessus « Clk ».

La Figure 5-22 présente alors une partie de l'adaptateur qui est l'interprétation de la mise en correspondance des ports internes de la FFT et le port externe « T_R ».

5.3.3.3.2.1.2 Modèle à base de composants virtuels de la description multi niveaux de la chaîne 802.16

Pour générer le modèle d'exécution décrit précédemment, et pour abstraire l'hétérogénéité entre les différents composants, nous utilisons le modèle à base de composants virtuels. Comme décrit dans le **chapitre 3**, les différents composants sont encapsulés par une interface abstraite. Elle est constituée de ports internes relatifs aux composants et ports externes relatifs à l'environnement d'exécution. La Figure 5-23 illustre le modèle à base de composants virtuels de l'application.

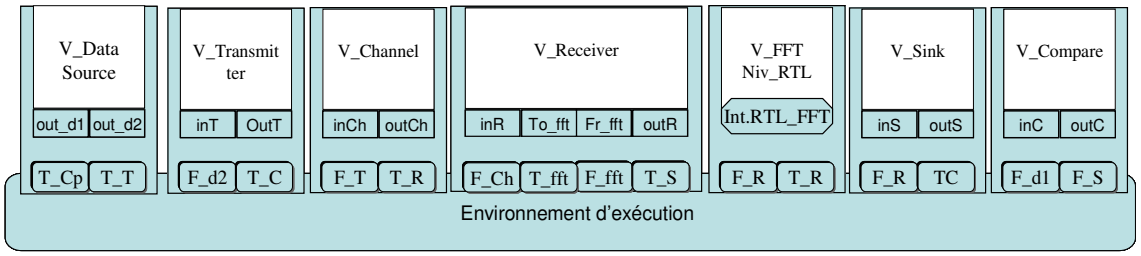


Figure 5-23 Modèle à base de composants virtuels de la description multi niveaux de la chaîne 802.16

5.3.4 Résultats

Dans cet exemple, nous avons défini et réalisé un modèle d'exécution global pour un modem d'une chaîne audio 802.16 (avec une description multi niveaux des différents composants). Le modèle généré est conforme aux concepts de base définis dans le chapitre 3.

Nous avons pu identifier les différents niveaux d'abstraction des différents modèles d'exécution des composants. Ainsi, nous avons pu définir les modèles à base de composants

virtuels des différentes descriptions (modélisation au niveau transaction de tout le système et modélisation mixte).

Les perspectives pour cette application sont de raffiner les autres composants et de faire une exploration d'architecture pour l'implémentation logicielle ou matérielle des différents composants. La deuxième perspective est de déterminer les modèles d'exécution globaux de toutes les descriptions ainsi que les modèles à base de composants virtuels correspondants.

5.4 Modélisation de l'application SDR

5.4.1 Présentation générale de l'application

L'application SDR (Software Defined Radio) représente une radio reconfigurable pouvant être programmée par le logiciel. Cette application est définie dans le cadre d'un projet européen.

Le terme « Radio Logicielle » se réfère à une collection de technologies et une architecture standard pouvant séparer l'application de la plateforme matérielle. Grâce à cette notion, les nouvelles applications et les corrections des programmes existants peuvent être téléchargées. Ainsi, cette approche permet de surmonter la nécessité de faire des modifications physiques à un ensemble considérable de radios.

Une radio est généralement caractérisée par une variété de fonctions permettant la conversion de la voix en données à partir d'une fréquence donnée. Ces fonctions sont :

- Traitement de signal analogique (par exemple, amplification/désamplification, filtrage, etc.).
- Modulation et démodulation de l'onde avec une éventuelle correction d'erreur.
- Traitement du signal de la bande passante (par exemple, routage vers les périphériques, ajout de protocoles, etc.).

Une radio logicielle est une radio où les fonctions de modulation et de démodulation sont définies en logiciel. Du côté émetteur, ceci équivaut à un envoi d'onde générée comme de simples signaux numériques convertis en signaux analogiques. Du côté du récepteur, l'onde analogique reçue est extraite, convertie, démodulée en un signal numérique.

L'architecture de la radio logicielle est constituée de trois blocs de base :

- Les composants (logiciel ou matériel).
- Les règles de composition définissant les adaptations nécessaires entre les composants.
- Les règles du comportement fournissant la sémantique du système et définissant le comportement des différents composants et leurs interactions.

L'objectif pour cette application de modulation et de démodulation est de la décrire avec un ensemble de composants décrits eux-mêmes avec un langage de haut niveau (Figure 5-24). Ces composants seront exécutés sur une plateforme fournissant des ressources et des services standards. Le langage de description de nos composants sera CORBA (voir Chapitre 4). Dans la section 5.4.2, un petit rappel de la description avec ce langage est présenté.

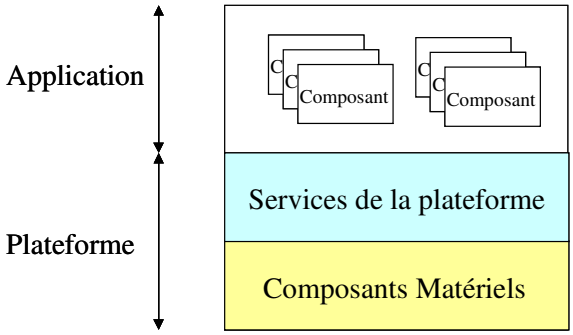


Figure 5-24 Exécution des composants de modulation et démodulation sur une plateforme

Ainsi, à travers cette application, nous essayons de promouvoir la réutilisation et la portabilité des composants logiciels sur différentes plateformes matérielles. Ceci est réalisé à travers la création de liens entre les outils d'exploration d'architecture et les outils de conception logiciels. Cette application comporte des composants de natures différentes (logiciels et matériels). Pour atteindre notre objectif, différents modèles d'exécution sont nécessaires ainsi qu'un développement des interfaces abstraites logicielles/matérielles.

Afin d'atteindre notre but, il a été nécessaire de diviser le travail en deux parties :

- **La première partie** consiste à porter une description d'une application décrite en CORBA (utilisant des composants et des conteneurs) vers une description en SystemC sans changement de code des composants. Un modèle d'exécution sur une plateforme abstraite en SystemC équivalente à la plateforme CORBA est réalisé. Il est noté le modèle CORBA-SystemC. Ce modèle sera décrit dans la section 5.4.4.1. Cette partie est utile pour l'apport que SystemC pourra fournir (annotations temporelles et portabilité). De plus, le modèle CORBA-SystemC va permettre une simulation de nos composants à un haut niveau d'abstraction qui permettra certaines mesures d'estimation de performance, en ajoutant des détails temporels sur le temps d'exécution des composants.
- **La deuxième partie** consiste, à partir du modèle d'exécution de notre application basé sur SystemC, à raffiner ce modèle en un modèle décrit à un niveau d'abstraction plus bas où il y aura la description abstraite de la plateforme sur laquelle les composants vont être exécutés sur : trois processeurs (un GPP, « General Purpose Processor », et deux DSP, « Digital Signal Processing ») et un composant matériel (un FPGA). Cette partie est effectuée en collaboration avec [You05].

Dans le cadre de mon étude, la première partie est seulement réalisée.

Par la suite, nous ferons un rappel sur la modélisation CORBA. Dans cette partie, nous expliciterons les différents composants CORBA avec nos concepts de base définis dans le chapitre 2. La deuxième partie présentera une application décrite en CORBA. Cette partie nous a permis de définir le sous-ensemble d'API décrit dans la section 5.4.3.1.2. La partie suivante sera consacrée au modèle équivalent en SystemC et à l'ensemble des API implémentées.

5.4.2 Description CORBA :

Le modèle CORBA est une présentation d'une architecture qui est apparue en 1990. Il permet la communication d'un ensemble d'applications dans des environnements hétérogènes (langages, systèmes d'exploitations, machine, etc.). CORBA présente une norme d'architecture distribuée ouverte. Elle permet de définir les composants et leurs interconnexions [Cor04].

5.4.2.1 Les composants CORBA

Les composants CORBA présentent des implémentations différentes pouvant s'exécuter sur des environnements différents. Ces composants sont encapsulés par une interface abstraite (voir Figure 5-25-a). Dans cette interface, une description des différents services requis et/ou fournis est déterminée. Elle présente un accord de communication entre les différents composants. Pour la description de ces interfaces, CORBA définit le langage de définition des interfaces IDL (Interface Description Language). Il permet de décrire les différents services fournis ou requis de chaque composant CORBA.

Dans notre approche, le composant CORBA est présenté comme étant un contenu et une interface. L'interface est un ensemble de services (définis à travers le IDL), alors que le contenu est l'implémentation ou l'appel des services. Par exemple, le composant réalisant le produit de deux matrices fournit le service nommé « product_matrix » à travers l'interface IDL, alors que le contenu est l'implémentation de ce service en un langage bien spécifique (C++ par exemple), voir Figure 5-25-b.

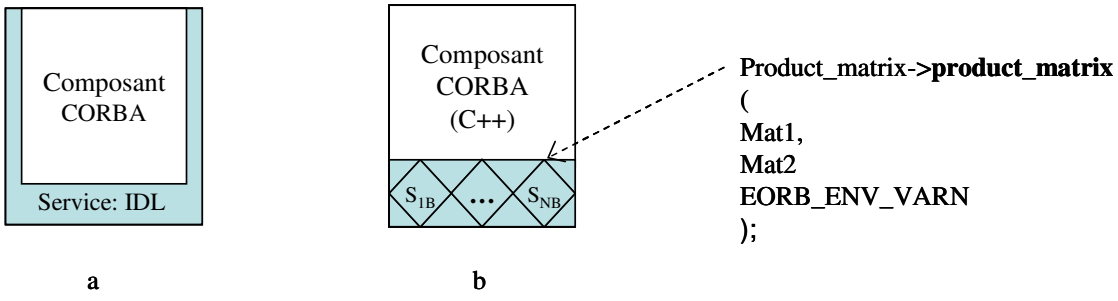


Figure 5-25 Description du composant CORBA a- selon les concepts CORBA ; b- selon notre approche

5.4.2.2 L'interconnexion CORBA

L'interconnexion CORBA est définie à travers le bus d'objets de CORBA noté ORB (de l'anglais Object Request Broker). Souvent appelé *bus logiciel*, l'ORB est au cœur de l'architecture présentée par l'OMG (Object Management Group). Il est responsable de la mise en relation et de la communication entre tous les composants présents dans cette architecture distribuée. Il prend en charge le dialogue entre les composants serveurs et les différents clients qui s'y connectent. Il rend ce dialogue transparent quels que soient les plates-formes, systèmes d'exploitation ou langages utilisés.

Il représente une interprétation particulière des interconnexions à un très haut niveau d'abstraction. Pour les systèmes embarqués, certaines fonctionnalités de l'ORB sont omises pour être suffisamment minimal. Il est noté dans ce cas ORB embarqué.

A ce niveau on peut représenter le modèle de spécification d'un système décrit en CORBA par la figure suivante.

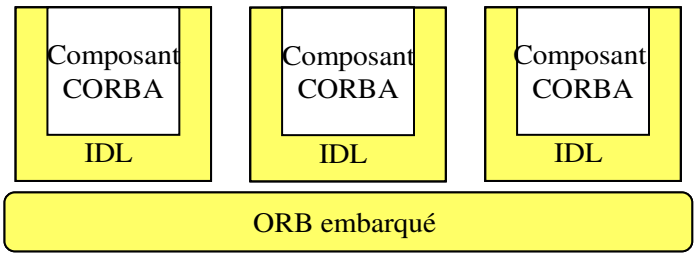


Figure 5-26 Modèle de spécification CORBA embarqué

Cependant, cette description n'est pas exécutable. Ceci est dû à l'hétérogénéité des composants encapsulés. Pour effectuer l'exécution du système global, l'environnement CORBA offre une génération automatique d'adaptateurs spécifiques. Ces adaptateurs d'objet sont chargés de la gestion des références aux objets et de l'activation des différentes implémentations. Ils représentent le moyen par lequel l'ORB peut accéder aux différents services fournis et offerts par les différents composants. Ces adaptateurs portent des noms différents selon le rôle que porte le composant correspondant. Pour les composants « serveurs » les adaptateurs sont nommés « squelettes » tandis que du côté des composants « clients » ces adaptateurs sont nommés « souches ». Dans notre cas d'application, les adaptateurs sont nommés « conteneur ».

Dans la section 4.4.1, l'environnement d'exécution de CORBA (l'ORB embarqué) définit une interface à travers certaines API. L'ensemble d'API de l'ORB embarqué est vu comme un **ensemble de services,** (représenté par des losanges selon la spécification du chapitre 2) requis par les conteneurs et les composants CORBA. Ces API seront décrites dans la section 5.4.3.1.2. Ainsi, le modèle d'exécution de CORBA est décrit par la Figure 5-27.

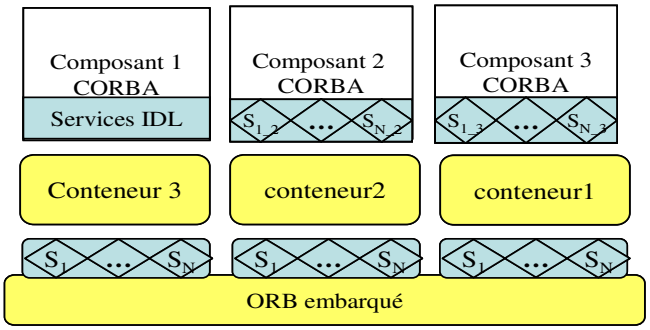


Figure 5-27 Modèle d'exécution basé sur CORBA embarqué

Dans notre cas d'étude, on considère un composant CORBA comme étant le composant initial plus son adaptateur (conteneur). La remarque générale pour ce modèle est que les conteneurs définissent aussi un ensemble de services (un ensemble d'API) permettant la communication avec l'ORB embarqué.

Par conséquent, un composant, pour notre cas d'étude, est vu comme le montre la Figure 5-28. Ce composant contient alors trois types d'interfaces définissant un ensemble de services pour la communication avec l'extérieur :

1- Les services requis ou fournis décrits par IDL (type 1). Ces services dépendent de l'application. Ils ne seront pas traités pour la transformation du modèle CORBA en modèle CORBA-SystemC. Ces services seront transformés par la génération des conteneurs en services de type (3).
2- Les services du composant d'accès direct avec l'ORB embarqué (type 2).
3- Les services du conteneur accèdent à l'ORB embarqué (type 3).

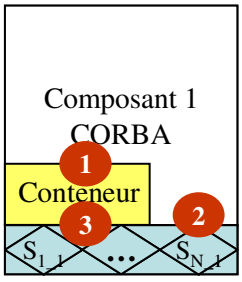


Figure 5-28 Composant CORBA (conteneur + implémentation) selon notre approche

5.4.3 Description de l'application CORBA

Dans cette section, nous présentons l'application développée pour valider notre approche de modélisation. Comme ORB embarqué, l'e*ORB (*embedded ORB)*, commercialisé par PrismTech [Pri05] est utilisé. Les étapes de l'implémentation de l'application seront détaillées également.

L'application est constituée par 3 composants : Un composant qui fournit des services (jouant le rôle d'un serveur) et deux composants qui requièrent des services (jouant le rôle de clients).

La conception de cette application était divisée en deux étapes :

1- Modélisation selon le concept CORBA pour générer les conteneurs et définir l'interface et l'ensemble de services de communication des composants avec l'extérieur (voir 5.4.3.1).

2- Modélisation selon le modèle CORBA-SystemC pour réaliser des analyses de performances et permettre le raffinement et l'implémentation matérielle/logicielle (voir 5.4.4).

5.4.3.1 Implémentation de l'application sur l'environnement e*ORB

La première étape en vue de l'implémentation de cette application sur un environnement d'exécution e*ORB consiste à spécifier les interfaces entre le serveur et les deux clients : Il s'agit d'écrire l'interface de chaque composant en décrivant leurs différents services *via* un fichier de description d'interfaces (.idl). Nous avons défini deux interfaces différentes, chacune représente les services fournis par le serveur pour un client. La Figure 5-29 représente la description de l'interface IDL pour le composant 1. L'interface « Product_Service » définit trois services :

1- Le service « Greeting » qui a pour but d'afficher une chaîne de caractères indiquant les différents états du composant (actif ou en attente) selon le paramètre d'entrée « greetstr ».

2- Le service « Product_ Matrix » qui effectue la multiplication de deux matrices. Il a comme paramètres deux matrices d'entrées et comme valeur de sortie, un entier indiquant la complétion du calcul.

3- Le service « shutdown » qui permet d'arrêter l'activité de l'objet effectuant le calcul ou d'arrêter l'exécution totale de l'application.

```
interface Product_Service
{
  string Greeting (in string greetstr);
  short Product_Matrix (in Matrix var_one, in Matrix var_two);
  oneway void shutdown();
};
```

Figure 5-29 Spécification de l'interface IDL du composant 1 (Serveur)

La seconde étape consiste à déterminer le modèle d'exécution CORBA pour cette application. Il s'agit de générer les conteneurs qui pouvaient réaliser la communication entre les

différents composants. Les conteneurs sont générés en utilisant un outil de transformation (*idlcpp*) fournis avec e*ORB.

L'implémentation du contenu des composants CORBA est réalisée avec un des langages de programmation (dans notre cas c'est le C++). Cette implémentation est réalisée en utilisant le principe d'héritages (avec le langage C++) des conteneurs et des interfaces.

Le code de l'application ainsi obtenu est validé fonctionnellement sur e*ORB. Le but de cette application est de disposer d'un exemple afin de fixer un sous ensemble d'API (services) CORBA à implémenter dans le modèle CORBA-SystemC.

Le modèle d'exécution CORBA de cette application est équivalent à la Figure 5-27. L'exécution globale de l'application est décrite dans la section 5.4.3.1.1. La section 5.4.3.1.2 décrira l'interface (ensemble de services) des composants CORBA.

5.4.3.1.1 Mode de fonctionnement du modèle d'exécution CORBA

Le mode de fonctionnement du modèle d'exécution des composants CORBA se base sur le fait que tous les composants se connectent à l'e*ORB. Du côté serveur, un enregistrement de tous les objets offrant les différents services doit se faire. Ainsi, lorsqu'un composant requiert un service, l'e*ORB pourra effectuer une recherche sur l'ensemble des objets déjà enregistrés.

L'enregistrement de ces objets est un mécanisme complexe afin d'assurer la transparence de recherche et d'exécution à l'utilisateur. Pour cela, du côté du composant fournissant les services, le conteneur est défini comme un adaptateur portable d'objet noté « POA » (Portable Object Adaptor). A chaque objet, il associe un servant qui va gérer l'exécution des services appelés sur l'objet concerné.

Pour assurer la transparence de la recherche de différents objets, d'autres services sont requis tel que le service de nommage dans l'e*ORB.

Pour tous les composants décrits par l'utilisateur, certaines API doivent être utilisés. Les API de communication entre les clients et les serveurs sont abstraites pour l'utilisateur. Elles sont appelées indirectement lors de l'invocation des services requis.

5.4.3.1.2 Description des interfaces (les APIs ou services) des composants CORBA

Ce paragraphe décrit les différentes API (services) nécessaires pour la description d'un modèle d'exécution CORBA. Ces API définissent un procédé de communication entre les composants et l'ORB, elles constituent l'interface avec laquelle les composants communiquent avec l'extérieur. Certaines API sont définies au niveau du composant CORBA (type 2 de la Figure 5-28) et d'autres sont définis au niveau des conteneurs (type 3 de la Figure 5-28).

API CORBA pour les utilisateurs : type 2

- *CORBA::ORB_ptr CORBA::ORB_init*

```
(
  int & argc,
  char ** argv
)
```

La fonction ORB_init est utilisé par le serveur et par le client pour se connecter à l'e*ORB et pour pouvoir effectuer les opérations de communication (les services requis et/ou fournis).

Selon l'implémentation de e*ORB de « PrismTech », elle retourne au composant appelant un pointeur sur un objet ORB. L'objet ORB offre une interface regroupant certaines méthodes utiles pour lancer l'exécution et la terminer. Ces méthodes sont « run » et « Shutdown ». La fonction « run » est utilisée par le serveur tandis que la fonction Shutdown est invoquée indirectement.

- *void CORBA::ORB::run* (CORBA::Long timeout_msec)

Comme expliqué précédemment, cette fonction permet de lancer l'exécution de l'ORB. Lors de l'appel de cette fonction, il est impératif pour le serveur d'avoir enregistré les différents objets qu'offrent les différents services. Le temps d'exécution de cette fonction peut être fixé en donnant en paramètre une variable temporelle « timeout_msec ». Le client ne peut s'exécuter correctement qu'après que le serveur ait exécuté cette API. Cette contrainte n'est pas imposée par l'implémentation de « PrismTech ».

Cette fonction peut être arrêtée aussi si l'e*ORB exécute la fonction « Shutdown ». Elle sera expliquée ultérieurement.

- *CORBA::Object_ptr CORBA::ORB::resolve_initial_references*

```
(
  const CORBA::ObjectId & id
  EORB_ENV_ARGN
)
```

Cette fonction est exécutée par tous les composants. C'est un service offert par l'ORB. Elle fait partie des services de nommage. Elle permet de faire la résolution de référence. Elle n'est utilisée que si on utilise dans les paramètres d'exécution du serveur ou du client des adresses de localisation de ressources (URL). Elle est exécutée dans l'e*ORB qui permet de retourner un pointeur sur un objet à partir de son identifiant.

Une autre méthode est utilisée pour la recherche de cet objet si l'URL n'est pas utilisé. Cette méthode est :

- *CORBA::Object_ptr IORUtil::read*

```
(
  const CORBA::ORB_ptr orb,
  char * name
  EORB_ENV_ARGN
)
```

Elle retourne un pointeur sur un IOR (de l'anglais Interoperable Object Reference). Ce pointeur représente une référence extraite d'un fichier qui contient des informations sur un objet déjà inscrit dans l'e*ORB « orb ». La recherche de ce fichier s'effectue à partir du nom attribué dans la liste des paramètres de la fonction « name ».

Cette fonction est effectuée par le demandeur de service seulement.

- *void IORUtil::write*

```
(
  const CORBA::ORB_ptr orb,
  const CORBA::Object_ptr objref,
  char * name
  EORB_ENV_ARGN    )
```

Cette fonction est effectuée par les composants fournissants les services. Elle permet d'enregistrer une référence d'un objet « objref » dans l'e*ORB « orb » dans un fichier ayant le nom « name ».

- *PortableServer::ObjectId_ptr PortableServer::POA::activate_object*

```
(
  PortableServer::Servant p_servant
  EORB_ENV_ARGN
)
```

Cette API est effectuée au sein du serveur, elle permet d'associer un servant « p_servant » à l'objet offrant le service. Ce servant fera partie de la liste des servants du squelette et sera actif, c'est-à-dire, il est prêt à exécuter les requêtes demandées. Cette API retourne un identifiant de cet objet dans le POA.

- *PortableServer::POA_ptr PortableServer::POA::_narrow*

```
(
CORBA::Object_ptr obj
EORB_ENV_ARGN
)
```

Cette API est utilisée au niveau du serveur. Elle permet de forcer le type : une surcharge de la classe de base (CORBA::Object) vers la classe dérivée (PortableServer::POA) lors de

l'exécution. Cette fonction est nécessaire pour garantir la transparence d'exécution et de recherche des services.

***Remarque* :**

Dans la modélisation donnée par « PrismTech » des macros sont définies. Elles sont utilisées dans les paramètres des API. Parmi ces macros on a : « EORB_ENV_ARGN ». Elle permet de définir l'environnement dans lequel on exécute le client et/ ou le serveur.

- **API CORBA des conteneurs : type 3**

Certaines API sont utilisées au niveau des adaptateurs qui sont générés automatiquement. Ces API sont décrites dans la suite. Elles permettent de cacher les méthodes de communication et de recherche du service par rapport à l'utilisateur.

- *void CORBA::ORB::shutdown*

(

CORBA::Boolean wait_for_completion

EORB_ENV_ARGN

)

C'est la fonction qui va permettre d'arrêter le fonctionnement global de l'e*ORB. Elle s'effectuera si l'événement « wait_for_completion » est activé.

- *EORB::Codec::RequestId CORBA::Object::invoke_request*

(

CORBA::String opname,

CORBA::ULong oplen,

CORBA::OperationMode mode,

EORB::Codec::Param * putParams,

CORBA::ULong putElems,

EORB::Codec::Param * getParams,

CORBA::ULong getElems

EORB_ENV_ARGN

)

Lors de l'appel d'un service fourni par l'interface décrite en « idl », le composant requérant ce service va faire un appel indirect à cette API à travers le conteneur. Elle permet d'invoquer un service de l'e*ORB. Elle comprend certains paramètres. Ces paramètres sont comme suit : « opname » identifie le nom du service requis, les pointeurs « putParams » et « getParams » identifient les variables d'entrées et de sortie pour ce service. A partir de ces paramètres, l'ORB génère une requête qui sera acheminée jusqu'au serveur.

- *void EORB::Codec::Request::get_args*

  (

  EORB::Codec::Param * params,

  CORBA::ULong elems

  EORB_ENV_ARGN

)

Lors de l'invocation d'un objet pour exécuter un service donné, cet objet a besoin d'avoir les paramètres effectifs sur lesquels il doit travailler. Ces paramètres ont été envoyés par le composant requérant le service grâce à l'API « invoke_request » décrite précédemment. La primitive « get_args » permet d'avoir les paramètres d'entrées du service à fournir à partir de l'e*ORB.

- *void EORB::Codec::Request::put_args*

  (

  EORB::Codec::Param * params,

  CORBA::ULong elems

  EORB_ENV_ARGN

)

Pour mettre à jour les paramètres de sorties d'un service réalisé, l'objet concerné doit envoyer à l'e*ORB les nouvelles valeurs de ces paramètres. Ceci est effectué par la primitive « put_args ».

- *CORBA::Boolean CORBA::Object::_is_a*

  (

  const char * logical_is_type

  EORB_ENV_ARGN

)

Cette API est utilisée par le squelette et la souche pour tester si le type sur lequel on travaille est le même que «logical_is_type». Ceci est nécessaire pour pouvoir effectuer certaines opérations sur cet objet.

- *CORBA::Object_ptr CORBA::Object::_duplicate*

(

CORBA::Object_ptr obj

)

Cette API est nécessaire pour dupliquer un objet en un paramètre et retourner un pointeur sur ce nouvel élément dupliqué.

5.4.4 Description du modèle d'exécution équivalent en SystemC, noté CORBA-SystemC

Le but de cette partie est de définir un modèle d'exécution équivalent à CORBA mais décrit en SystemC. Le modèle CORBA-SystemC permettra un lien entre les composants CORBA et l'architecture finale sur laquelle ils vont être exécutés.

Pour réaliser cette description, il a été nécessaire de redéfinir le modèle d'exécution global de CORBA par un modèle équivalent CORBA-SystemC.

Le modèle d'exécution global CORBA-SystemC définira une réalisation (logicielle, matérielle ou fonctionnelle) des composants CORBA. Comme nous avons vu, le travail pour l'application SDR est divisé en deux étapes. Pour la première étape, l'implémentation des composants CORBA est considérée comme fonctionnelle au niveau Service. Pour la deuxième partie, les composants seront répartis sur la plateforme de destination (des composants logiciels exécutés sur les trois processeurs, et des composants matériels réalisés sur un FPGA), voir la section 5.4.4.1.

Afin de faciliter la génération de ce modèle d'exécution, comme vu dans la section 4.3, il est nécessaire de définir un modèle à base de composants virtuels CORBA-SystemC. Ce modèle permettra d'abstraire l'hétérogénéité des composants CORBA, voir 5.4.4.2.

5.4.4.1 Modèle d'exécution global CORBA-SystemC

Le modèle d'exécution global est défini par un assemblage des modèles d'exécution des différents composants CORBA, un environnement d'exécution et des adaptateurs si nécessaires (voir la section 4.2).

Le modèle global d'exécution CORBA-SystemC est donné par la Figure 5-30. Les caractéristiques de ce modèle sont définies par :

- Les composants CORBA : La description des différents composants CORBA est inchangée. Ces composants possèdent des interfaces de communication comme décrites dans 5.4.3.1.2. Les modèles d'exécution de ces composants sont définis en tant que modèles fonctionnels au niveau service (voir 3.2.1.3).
- L'environnement d'exécution : Il définit l'interconnexion entre les composant CORBA décrite en SystemC. Cette interconnexion doit préserver la politique de communication de l'e*ORB. Sa définition en tant que media, comportement et type de données est donnée comme suit :
    - Media : il est défini comme étant un canal SystemC nommé « ORB_Channel ». Ce canal implémente toutes les primitives décrites par l'interface de communication notée Int.EE. Cette interface sera détaillée dans la section 5.4.4.2.1. Ce canal peut être connecté à plusieurs composants CORBA à la fois.

- Comportement : le comportement définit le mode de fonctionnement du canal ORB de CORBA, mais implémenté en SystemC.
- Types de données : les types de données supportés par ce canal sont ceux de CORBA et de SystemC. Il présente des structures spécifiques qui permettent de garder la notion de transparence de communication entre les clients et les serveurs.

- Les adaptateurs : Il a été nécessaire de définir des adaptateurs CORBA-SystemC entre l'environnement d'exécution et les composants CORBA (deux langages différents). Ces adaptateurs posséderont l'architecture généralisée telle que décrite dans la section 4.5.3. Ils seront détaillés dans la section 5.4.4.1.1.

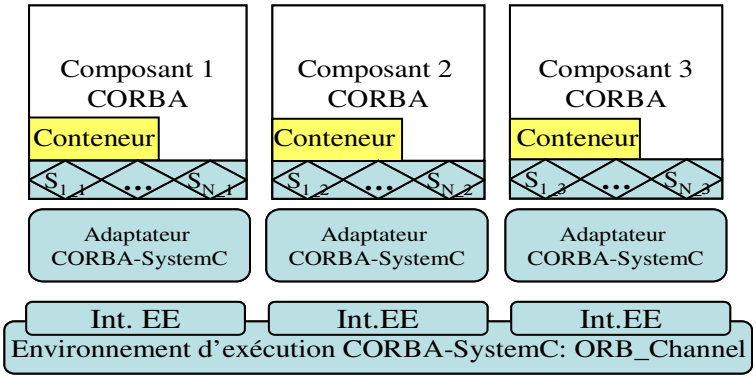


Figure 5-30 Modèle d'exécution global CORBA-SystemC

5.4.4.1.1 Adaptateur CORBA-SystemC

L'adaptateur CORBA-SystemC représente l'adaptation entre les interfaces des composants CORBA et celle l'environnement d'exécution SystemC :

- L'interface des composants CORBA définie en tant qu'ensemble de services (voir la section 5.4.3.1.2).
- L'interface de cet environnement est définie par un port logique ayant plusieurs services (voir 5.4.4.2.1).

La structure de l'adaptateur est donnée selon le graphe de dépendance de services (voir Figure 5-31).

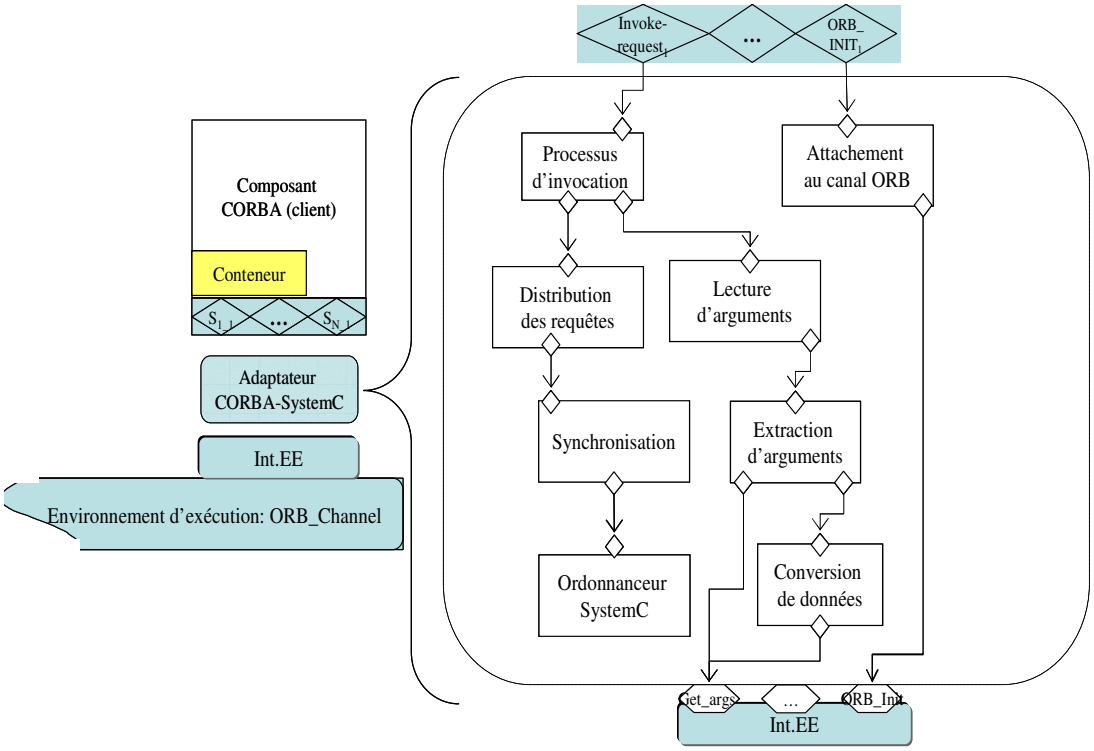


Figure 5-31 Exemple d'adaptateur CORBA-SystemC

L'adaptateur comprend alors des éléments offrants des services qui eux mêmes peuvent requérir d'autres services d'autres éléments ou du port logique de l'environnement d'exécution *via* les points d'accès aux services (SAP).

Dans l'exemple d'adaptateur de la Figure 5-31, le composant demandant des services (jouant le rôle de client) est considéré. Dans cet exemple, seulement deux services requis sont définis (« invoke_request » et « ORB_INIT ») pour illustrer le graphe de dépendance des services.

Tout d'abord, le composant CORBA doit s'attacher au canal ORB pour pouvoir communiquer avec l'extérieur. Ce service est fourni par l'élément « Attachement au canal ORB ». Cet élément nécessite un service d'initialisation fourni directement par le SAP « ORB_INIT » du port logique de l'environnement d'exécution.

Pour invoquer un service, le composant doit utiliser l'API « Invoke_request ». Pour cela, un élément nommé « Processus d'invocation » est utilisé. Ce dernier permet d'envoyer la requête au canal ORB_Channel. Cependant, il nécessite deux autres services : la distribution des différentes requêtes vers le canal et l'extraction des paramètres des requêtes pour les envoyer au composant fournissant le services *via* l'ORB_Channel. La distribution des requêtes est nécessaire s'il existe plusieurs composants qui demandent des services en même temps. Elle requiert un service de synchronisation fourni par l'Ordonnanceur SystemC. L'extraction des

paramètres a besoin d'une conversion de données et d'un accès direct au SAP « Get-Args » du port logique de l'environnement d'exécution. La conversion de données nécessite aussi l'accès à ce SAP pour pouvoir changer les types de données selon les types de l'ORB_Channel.

Le graphe de dépendance de services, présenté dans cet exemple, est incomplet. En effet, à chaque API présentée dans la section 5.4.3.1.2 correspond un graphe de dépendance de services. Certains éléments de ce graphe peuvent se chevaucher et être réutilisés selon les services fournis ou requis des composants.

5.4.4.2 Le modèle à base de composants virtuels CORBA-SystemC

Le modèle à base de composants virtuels CORBA-SystemC est défini comme l'assemblage des modèles d'exécution des composants CORBA enveloppés dans des interfaces virtuelles. Le modèle consiste donc à envelopper les composants CORBA par des modules en SystemC, notés « SC_CORBA_MODULE ». Ce module définira une interface virtuelle qui lui permettra d'abstraire l'hétérogénéité et un contenu qui représente le comportement des composants CORBA. L'interface virtuelle est constituée par une interface interne relative aux composants CORBA et une interface externe relative à l'environnement d'exécution (voir Figure 5-32).

- L'interface interne représente l'ensemble d'API décrit dans la section 5.4.3.1.2.
- L'interface externe représente un port logique contenant un ensemble d'accès à un ensemble de service (SAP) et un point d'accès à un port (PAP) représentant une abstraction d'un port d'accès SystemC.

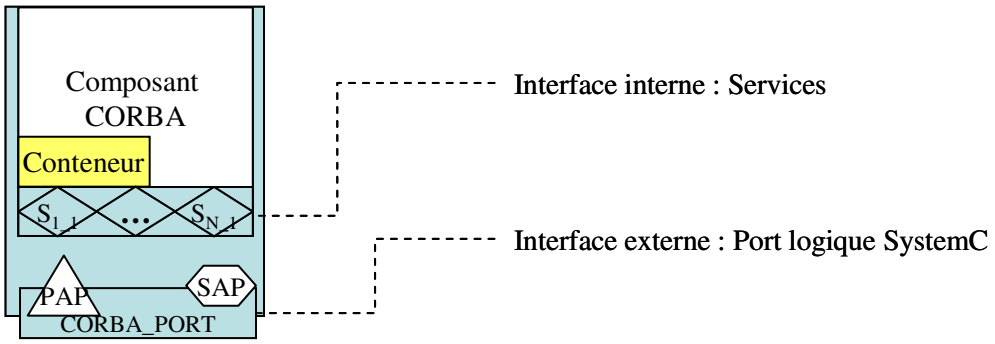


Figure 5-32 Composant virtuel CORBA-SystemC

Par la suite, nous décrirons l'interface externe ainsi que la modélisation du module d'encapsulation.

5.4.4.2.1 Description des interfaces externes du modèle CORBA-SystemC

L'interface externe représente celle de l'environnement d'exécution. Elle est définie comme un port logique ayant des points d'accès à des services (SAP) et un point d'accès à un port (PAP).

L'interface externe est décrite en SystemC et nommée « CORBA_Port ». Le port « CORBA_Port » permettra l'interconnexion avec le canal « ORB_Channel ». Sur ce port, on a besoin d'indiquer un identifiant de rôle. Cet identifiant est « 1 » pour un composant fournissant des services et « 0 » pour un composant qui requiert des services. Par exemple, dans un programme, l'instanciation d'un port est comme suit : ***CORBA_Port*** *port(1)* ;

Au niveau de ce port, des SAP sont définis. Ils représentent des services équivalents à ceux de CORBA mais implémentés en SystemC. Cependant, les services nécessaires pour une modélisation SystemC sont ceux de la communication entre les serveurs et les clients. Ces services vont être implémentés de telle manière que le fonctionnement du modèle initial de CORBA soit réservé. Ainsi, un classement des services décrits dans 5.4.3.1.2 a été rétabli selon l'importance pour l'implémentation en SystemC. Certaines API sont nécessaires pour la communication, tandis que les autres le sont pour le fonctionnement global sans changement du code des composants CORBA.

Le tableau suivant présente l'ensemble des SAP requis et fournis au niveau du port logique CORBA_Port.

API nécessaires pour la cohérence de modélisation	**API nécessaires pour la communication entre les composants en SystemC**
➢ CORBA::ORB_ptr CORBA::ORB_init() ➢ CORBA::Object_ptr CORBA::ORB::resolve_initial_references() ➢ CORBA::Boolean CORBA::Object::_is_a() ➢ CORBA::Object_ptr CORBA::Object::_duplicate () ➢ PortableServer::POA_ptr PortableServer::POA::_narrow () ➢ CORBA::Object_ptr IORUtil::read() ➢ void IORUtil::write()	➢ PortableServer::ObjectId_ptr PortableServer::POA::activate_object() ➢ EORB::Codec::RequestId CORBA::Object::invoke_request() ➢ void EORB::Codec::Request::get_args() ➢ void EORB::Codec::Request::put_args() ➢ void CORBA::ORB::run ()

Tableau 6 Différents SAP du port CORBA_Port implémentés en SystemC

Entre les ports internes et les ports externes il y a une mise en correspondance. Cette mise en correspondance est fournie à travers l'implémentation de ce port avec SystemC2.0 grâce à la notion de l'interface associée à un port. Cette interface est notée « Corba_if ». Elle définit

l'ensemble des opérations pouvant être appelées sur ce port et fournit l'ensemble des API CORBA décrites ci-dessus.

ATTENTION : l'interface SystemC associée aux ports n'est pas définie comme dans notre terminologie

5.4.4.2.2 Description des composants virtuels CORBA-SystemC

Pour décrire les composant virtuels CORBA-SystemC, une implémentation en SystemC a été effectuée. L'implémentation actuelle de ces composants est réalisée avec les concepts du module SystemC, « SC_MODULE ». Trois implémentations des composants virtuels sont définies dépendant du rôle auquel on associe le composant CORBA (fournit un service, requiert un service ou spécifie les deux).

Les modules enveloppant les composants CORBA seront nommés, en fonction de leurs rôles.

- Pour un module enveloppant un composant fournissant un service, on le note « *SC_CORBA_SERVER_MODULE* ».
- Pour un module requérant un service, noté « *SC_CORBA_CLIENT_MODULE* » et
- Pour un module ayant un double rôle, noté « *SC_CORBA_CLISER_MODULE* ».

➢ Description des modules à rôle unique

Le contenu des modules SystemC, le composant CORBA et son conteneur, est défini comme un processus SystemC. Dans ce processus, on exécute le comportement des composants CORBA. La demande ou la fourniture d'un service passe par la mise en correspondance entre l'interface interne et l'interface externe.

Le schéma suivant présente le modèle SystemC d'une telle description.

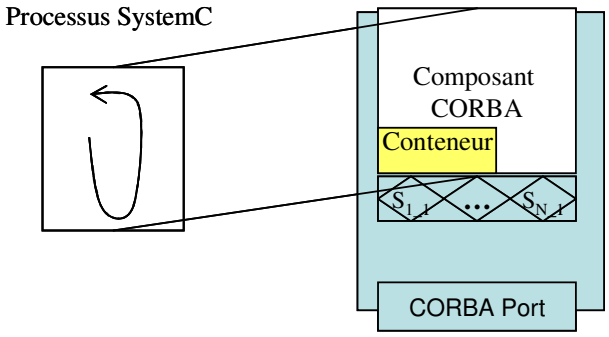


Figure 5-33 Description d'un module CORBA en SystemC

➢ Description des modules ayant un double rôle

Le contenu de ces modules est défini par deux processus SystemC différents, un pour chaque rôle. Un processus pour les objets serveurs et un autre pour les objets clients. Chacun d'eux opère sur une interface dédiée. L'interface du module comporte alors deux ports « CORBA_Port ». Le modèle SystemC d'une telle description est donné par la figure suivante.

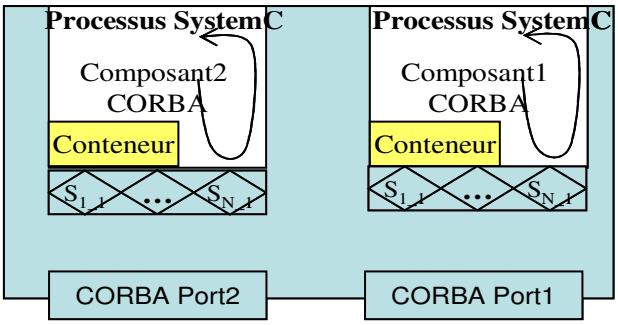


Figure 5-34 Description d'un module CORBA à double rôle en SystemC

Pour la description du modèle à base de composants virtuels global d'une application CORBA-SystemC, le code donné par la Figure 5-35 est employé.

```
int sc_main (int argc , char *argv[])
 {
// Déclaration des composants  CORBA et leurs conteneurs
  corba_proc server, client, client1;
  server= main_server;
  client= main_client;
  client1= main_client1;

// Définition de  l'ORB SystemC Channel
 ORB_Channel orb_sysc;

// Définition des modules SystemC_CORBA
 SC_CORBA_SERVER_MODULE  *Ser_mod;
 SC_CORBA_CLIENT_MODULE  *Cli_mod;
 SC_CORBA_CLIENT_MODULE  *Cli_mod1;

// Encapsulation des modules CORBA en SystemC
 Ser_mod = new SC_CORBA_SERVER_MODULE("SERVER",server);
 Cli_mod= new SC_CORBA_CLIENT_MODULE("CLIENT", client);
 Cli_mod1= new SC_CORBA_CLIENT_MODULE("CLIENT1", client1);

// Interconnexion entre les différents modules et le canal
 (*(Ser_mod->P_server))(orb_sysc);
 (*(Cli_mod->P_client))(orb_sysc);
 (*(Cli_mod1->P_client))(orb_sysc);

/lancement de l'exécution
 sc_start(-1);
 return (0);
 }
```

Figure 5-35 Description du modèle d'exécution en SystemC d'un modèle équivalent CORBA

.

5.4.4.3 Résultats et perspectives

Dans le cadre de cette application, nous avons pu développer :

- Une application en CORBA qui tourne sur e*ORB.
- Un modèle CORBA_SystemC qui permet de simuler/exécuter toute application CORBA qui utilise les mêmes services CORBA que l'exemple qui a été développé.
- Un modèle CORBA_SystemC qui sera le modèle de base pour le raffinement de ce modèle vers un modèle plus détaillé.

Comme perspectives pour ce travail, le modèle d'exécution détaillé dit modèle au niveau HAL (Hardware Acess Layer), raffiné à partir du modèle CORBA-SystemC, sera à réaliser. Cela va permettre d'obtenir certaines évaluations des performances grâce aux informations additionnelles sur l'architecture matérielle réelle et aux annotations des API CORBA par SystemC.

5.5 Conclusion

Ce chapitre a présenté l'application des concepts étudiés sur des application complexes : un système embarqué d'un centre d'information d'un véhicule, d'un modem d'une chaîne audio 802.16 et d'un système de radio définie par logiciel. Les objectifs étaient de :

- Valider l'approche du modèle à base de composants virtuels pour la spécification d'un système hétérogène.
- Définir des modèles d'exécution globaux pour des descriptions de systèmes hétérogènes à plusieurs niveaux d'abstraction et décrits par des langages différents.
- Générer les éléments du modèle d'exécution global pour chacune des différentes applications étudiées à savoir les adaptateurs et l'environnement d'exécution.

Le modèle à base de composants virtuels s'avère très efficace pour l'abstraction de l'hétérogénéité des sous-systèmes en se basant sur la séparation de la communication et du comportement grâce aux concepts des ports internes et ports externes.

Finalement, nous pouvons dire que le modèle d'exécution global est identique quels que soient les langages, les niveaux d'abstraction et le domaine d'application. Le modèle défini dans le chapitre 4 s'avère un modèle assez général en se basant sur les concepts définis dans le chapitre 2.

Chapitre 6 Conclusion et perspectives

Conclusion

Les systèmes hétérogènes mono-puce se composent de différents éléments interconnectés. Chaque élément possède ses propres caractéristiques et ses particularités. La conception de ces systèmes opte pour la réutilisation et l'assemblage des composants. Le but pour la conception de ces systèmes est de définir des modèles (de calcul ou d'exécution) de tels systèmes. Cependant plusieurs difficultés sont liées à ce flot de conception, d'une part, par la diversité de l'existant et d'autre part, par la nécessité de la multidisciplinarité de l'intégration de ces composants. Dans cette thèse, nous avons abordé deux principaux problèmes : la spécification et la définition d'un modèle d'exécution global pour la validation de ces systèmes tout au long du flot de conception. Nous proposons alors une étude sur ces systèmes afin de faciliter la définition des modèles de calcul et les modèles d'exécution des systèmes hétérogènes.

Le chapitre 2 a présenté une étude sur la spécification et la modélisation des systèmes hétérogènes embarqués. La première partie décrit la terminologie commune extrapolée de l'étude de langages de modélisation et de spécification les plus utilisés de ces systèmes. Cette terminologie couvre un certains nombres de concepts offerts par ces langages tels que le comportement, l'interface, l'interconnexion, la communication, la hiérarchie et la composition. Ils présentent alors les définitions de base nécessaires pour la modélisation des systèmes embarqués. La deuxième partie était dédiée à l'importance de séparer les notions de modèles de calcul et de modèles d'exécution pour parvenir à la modélisation des systèmes hétérogènes. Dans cette partie, nous avons recensé les différents modèles de calcul des composants et des interconnexions. Deux modèles sont distingués : les modèles de base élémentaires et ceux par composition. Cette distinction vient du choix du critère de sélection des différents modèles. Pour chacun de ces modèles on distingue des sous modèles. Pour les modèles de base et en se basant sur le critère d'ordre d'occurrence du calcul, on distingue les modèles flux de données et ceux à flux de contrôle. Pour les modèles de composition, certains critères de composition ont fait qu'il existe deux modèles : les modèles de composition par hiérarchie comportementale et

ceux de composition par hiérarchie d'interconnexion. Ces dernières sont spécifiques aux modèles distribués. Ceci étant, une étude sur les caractéristiques des interconnexions a été réalisée. Cet ensemble de définition nous a permis de définir un modèle d'assemblage des systèmes hétérogènes basé sur les modèles distribués ayant plusieurs interconnexions.

Finalement, cette étude nous a permis de mettre l'accent sur la difficulté de modéliser un système hétérogène en tant qu'ensemble de sous-systèmes interconnectés. En effet, les caractéristiques définies par les modèles de calcul restent insuffisantes pour déterminer l'exécution finale du système. L'implémentation en matériel ou en logiciel du sous-système influe sur ces caractéristiques. Le chapitre 3 était donc consacré aux modèles d'exécution des systèmes hétérogènes embarqués. Pour étudier un système, il est nécessaire d'analyser son implémentation. Cette implémentation définit le modèle d'exécution du système. Elle peut être logicielle, matérielle ou fonctionnelle. Dans une première partie, nous avons étudié les différentes exécutions et nous avons déterminé leurs différents niveaux d'abstraction de chaque modèle d'exécution. L'exécution des systèmes ne s'arrête pas au comportement, elle s'étend aussi à l'interconnexion entre les différents sous-systèmes. La deuxième partie de ce chapitre était alors consacrée à l'étude des exécutions des interconnexions. Les implémentations des interconnexions ont été définies aussi à travers plusieurs niveaux d'abstraction. La troisième partie de ce chapitre est une classification des langages de modélisation et d'exécution selon les différents niveaux d'abstraction des interconnexions. Cette classification nous a permis d'approfondir les concepts de base déjà déterminés dans le chapitre précédent.

Le chapitre 4 a présenté l'importance d'avoir un modèle d'exécution global pour la validation des systèmes hétérogènes tout au long du flot de conception et que les modèles d'exécution et de validation sont validés et sont devenus standards. Dans ce cas, il serait dommage de ne pas les réutiliser. Dans ce chapitre, et dans une première partie, nous avons défini un modèle global d'exécution par composition pour les systèmes hétérogènes embarqués. Ce modèle est utilisé tout au long du flot de conception de ces systèmes. Il permet de définir deux éléments essentiels : l'environnement d'exécution et les adaptateurs. L'intérêt de ce modèle est de supporter les interconnexions hétérogènes et de réutiliser l'existant. Dans une deuxième partie, nous nous sommes intéressés au concept d'interconnexion hétérogène et nous l'avons généralisé et défini grâce à la séparation entre modèle de calcul et modèle d'exécution. Cette généralisation était introduite grâce au modèle à base de composants virtuels. Certains travaux qui l'ont utilisé étaient détaillés dans la troisième partie. La dernière partie était consacrée à la présentation d'un flot automatique pour la génération des modèles d'exécution globaux des systèmes hétérogènes. Il comporte trois parties essentielles : (1) la sélection des différents éléments des modèles d'exécution, (2) la configuration de ces éléments selon l'application et (3) la génération de code correspondant. Grâce à ce flot, nous avons pu

déterminer une architecture générique des adaptateurs des modèles d'exécution. Cette architecture est basée sur un graphe de dépendance de services noté GDS.

Le chapitre 5 a présenté l'application des concepts proposés sur des applications complexes : un centre d'information pour les véhicules, un modem d'une chaîne audio 802.16 et un système de radio défini par le logiciel (SDR). Les objectifs ont été (1) de définir pour chacune de ces applications le modèle d'exécution global en définissant les éléments nécessaires à savoir l'environnement d'exécution et les adaptateurs, sachant que chaque application présente des concepts différents (langages de modélisation, niveaux d'abstraction, types d'exécution), (2) prouver l'efficacité de notre modèle d'abstraction des interconnexions hétérogènes : le modèle à base de composants virtuels et (3) de montrer l'avantage de notre modèle en prouvant la généralité de l'architecture des adaptateurs pour tous les niveaux d'abstraction et les différents langages de modélisation, etc. tout au long du flot de conception.

Finalement, le modèle d'exécution ainsi défini peut être appliqué pour tous les systèmes et à n'importe quel niveau d'abstraction. Ceci est réalisé grâce à l'architecture générique des adaptateurs, qui est la même pour toute adaptation (logicielle, matérielle ou fonctionnelle).

Perspectives

Notre étude sur la modélisation des systèmes hétérogènes nous a permis de définir les concepts de base pour la modélisation. Cette terminologie s'est heurtée au début à la difficulté de la définition du modèle de calcul des systèmes hétérogènes et à la difficulté de la définition de l'interconnexion. C'est pourquoi elle reste non formelle. Une des perspectives est alors d'approfondir ces notions et de trouver un fondement mathématique permettant de mieux comprendre les interconnexions et le comportement. Ainsi, elle pourra être considérée comme un début pour plusieurs études sur la formalisation et la définition formelle d'un modèle d'exécution global tout au long du flot de conception des systèmes hétérogènes embarqués.

Le modèle d'exécution global générique représente une solution efficace pour la validation des systèmes hétérogènes embarqués à différents niveaux d'abstraction. Cependant, pour définir un flot de conception, il faut fixer des modèles d'exécution à différents niveaux d'abstraction et de méthodes de raffinements permettant de passer d'un modèle à un autre. Une telle approche nécessite une définition formelle des modèles d'exécution intermédiaire. L'approche apportée par le modèle « MDA » (Model Driven Architecture) [Sam04] semble être une bonne démarche pour établir les différentes transformations entre les modèles aussi bien pour le logiciel que pour le matériel.

Références

[Ahb99] "AMBA Specification", édité par ARM Limited, 1999, disponible en ligne http://www.gaisler.com/doc/amba.pdf

[Ali98] M. Alissali, "Introduction au génie logiciel", support de cours novembre, 1998, disponible en ligne, http://www-ic2.univ-lemans.fr/~alissali/Enseignement/Polys/GL/gl.html

[And98] C. ANDRÉ, H.BOUFAIED, S.DISSOUBRAY, « SyncCharts/Esterel : un modèle synchrone pour systèmes réactives complexes », dans les anales de Real-Time and Embedded Systems, Paris, 14-16 Janvier 1998, pp 175-194, Teknea. 1ier prix de la meilleure présentation scientifique.

[Ard99] L.Arditi, A.Bouali, H.Boufaied, G.Clave, L.Leblanc, R.De Dimone, "Using Esterel and Formal methods to increase the confidence in the functional Validation of a Commercial DSP", In proceedings of ERCIM workshop on Formal Methods for Industrial Critical Systems, Toronto, Italy, 1999.

[Bou05] A.Bouchhima, « Modélisation du logiciel embarqué à différents niveaux d'abstraction en vue de la synthèse et validation de systèmes sur puces, Thèse de Doctorat INPG, Spécialité Microélectronique, 2005.

[Bur03] M. Burton, F. Ghenessia, Stuart Swan, white paper, "Transaction Level Modeling: Overview and Requierements for SystemC Methodology", May 13, 2003

[But97] "Programming with Posix Thread", D. R. Butenhof, Published by Addison Wesley Professional, May 16, 1997.

[Cal96] J.P. Calvez, D.Heller, O.Pasquier, "System Performance Modeling and Analysis With VHDL", Proc. VHDL Forum for CAD in Europe conference.

[CDF05] Service en ligne, “Modeling Mixed Control/Data Flow Systems using PCC Models”, http://www.iss.rwthaachen.de/Projekte/Tools/MOLIERE/pcc_intro/pcc_intro.html.

[Ces01] W.O. Cesario, G. Nicolescu, L. Gauthier, D. Lyonnard, 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, Las Vegas, USA.

[Con03] J. Connell, “ARM System-Level Modelling”, white paper, June 25, 2003, www.arm.com.

[Cor04] Service en ligne, “Common Object Request Broker Architecture: Core specification”, v3.03.3, March 2004, OMG, Inc. http://www.omg.org/technology/documents/corba_spec_catalog.htm.

[Cos97] Synopsys, Inc., 700 E. Middlefield Rd., Mountain View, CA 94043, USA, COSSAP User's Manual, January 1997.

[Cos99] P. Coste, F. Hessel, P. Lemarrec, Z.Sugar, M.Romdhani, R.Suescun, N.Zergainoh, A.A. Jerraya, “Multilanguage Design of Heterogeneous Systems”, in Proc.of CODES, Rome, Italy, 1999.

[Cow05] Coware, service en ligne, http://www.coware.co.jp/

[Dco05] DCOM, Définition en ligne, http://www.webopedia.com/TERM/D/DCOM.html

[Dic05] Ordonnancement, Définition en ligne, http://w3.granddictionnaire.com/btml/fra/r_motclef/index1024_1.asp

[Dup03] S.Dupin, « Le langage C++ », édité Campus Press édition , 2003.

[Est05] Estelle, service en ligne, http://www.infres.enst.fr/~cottin/protocol/estelle.htm.

[Fou05] A.M. Fouilliart, E. Vaumorin, R. Nouacer, W. Cesario, L. Kriaa, P. Kajfasz, P. Coussy, F. Blanc, “SystemC Mantic : A high level Modeling and Co-design Framework For Reconfigurable Real Time Systems”, Forum on specification and Design Languages (FDL), Lausanne, Swiss, September 27-30, 2005.

[Gaj98] D. Gajski, F.Vahid, S. Narayan, and J. Gong, “SpecSyn: An Environment Supporting the Specify-Explore-Refine Paradigm for Hardware/Software System Design”, , IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 6, NO. 1, MARCH 1998.

[Gau01] 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 microélectronique, INPG, Laboratoire TIMA, Décembre 2001.

[Gei97] J.M. Geib, C. Gransart, P. Merle, “Corba, des concepts à la pratique”, Masson Editeur, Paris, 1997.

[Ger93] G. Gerrit de Jong, “Generalized data flow graphs theory and applications”, these Eindhoven, 1993.

[Gir99] A. Girlaut, B. Lee, E. Lee, “Finite State Machines with multiple concurrency models”, Fellow, IEEE, 1999.

[Gra05] A.Grasset, « synthèse des interfaces matérielles dans la conception des systèmes monopuces : de la spécification à la génération automatique », Thèse de Doctorat INPG, Spécialité Microélectronique, à soutenir fin 2005.

[Gro96] W. Gropp, E.Lusk, A.Skjellum, “Using MPI: Portable Parallel Programming with the Message-Passing Interface", MIT Press, 1996.

[HAL02] Service en ligne, “Hardware Abstraction Layer in RISC OS 5”, December 2002, http://www.iyonix.com/32bit/HAL.shtml.

[Har87] D.Harel, “A Visual Formalism for Complex Systems”, Science of computer Programming, 1987, 8, pp 231-274.

[Hav02] A. Haverinen, M. Leclercq, N, Weyrich, D. Wingard, "SystemC™ based SOC communication Modeling for the OCP™ Protocol", White paper, v1.0, October 2002.

[Hin96] Hines, R. John, "Software Engineering", *IEEE Spectrum,* January 1996, pp 60-64.

[Hol02] G. Holloway, M.D. Smith, "The Machine-SUIF Control Flow Graph Library", Release version 2.02.07.15, July 15 2002.

[Hol90] G.J.Holzmann, "Basic Spin manual", 10 Edition, Volume 2, Saunders College Publ., 1990, pp. 429-450.

[Idl05] OMG IDL, service en ligne, http://www.omg.org/gettingstarted/omg_idl.htm, 2005.

[Itr01] ITRS, International technology Roadmap for Semiconductors, Design, Edition 2001.

[Jag96] R. Jagannathan, "Parallel and Distributed Computing Handbook", Chap8: Dataflow Models, Edited by Y. Zomaya, 1996.

[Jan03] A. Jantsch, Morgan Kaufmann, "Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation", HardCover, June 2003.

[Kai02] C. Kaisar, "Synchronisation Par Message", support de cours, Mai 2002.

[Kea99] M.Keating, P.Bricaud, "Reuse methodology manual", Kulwer Academic Publisher, 1999.

[Laj99] M.Lajolo, M.Lazaraescu, A.Sangiovanni Vicentelli, "A Compilation based software estimation scheme for hardware/Software Cosimulation", Proc. Int'1 Workshop on hardware Software Codesign, May, 1999.

[Lam02] L. Lamport, "Specifying systems: the TLA+ Language and tools for hardware and software engineers", Pearson Education, Inc, 2002.

[Lee01] E.Lee, “Computing for Embedded Systems”, IEEE Instrumentation and Measurement Technology Conference Budapest, Hungary, May 21-23, 2001.

[Lee99] E. Lee, “Heterogeneous Concurrent Modeling and Design”, Superseded by "Overview of the Ptolemy Project," UCB-ERL No. M99/40, July 19, 1999.

[Liu01] Jie Liu, “Responsible Frameworks for Heterogeneous Modeling and Design of Embedded Systems”, theses report, 2001.

[Lut00] G.Luttgen, M.Von Der Beek, R.Cleavland, “A Compositional Approach to Statecharts Semantics”, ICASE Report N°2000-12, March 2000.

[Mad97] “Shore Data Language”, Reference Manual, Madison, WI Version 1.1, August 9, 1997.

[Mar91] F. Maraninchi, “The Argos Language: Graphical Representation of Automata and Description of Reactive Systems,” in Proc. IEEE Workshop on Visual Languages, Kobe, Japan, Oct. 1991.

[Mat00] “SIMULINK, Dynamic System Simulation for Matlab”, Using Simulink, Version 4, 2000 by the Math Works, Inc.

[Mat05] MATHWORKS.2005, Matlab, service en ligne, http://www.mathworks.com.

[Mod05] MODELSIM, service en ligne, http://www.model.com, 2005.

[Naj99] W. A.Najjar, E.A.Lee, G.R.Gao, “Advances in the Dataflow computational Model”, CAPSL technical Memo, 29 April1, 1999.

[Nic01] G. Nicolescu, K.Svarstad, W.Césario, 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, 8 Avril 2001.

[Nic03] G. Nicolescu, « Spécification et validation des systèmes hétérogènes embarqués », Thèse de Doctorat INPG, Spécialité Microélectronique, 2003.

[Occ05] "OCCN, On Chip Communication Network", disponible en ligne, http://occn.sourceforge.net/

[Pas99] O.Pasquier, J.P Calvez, "An Object-Based executable Model For simulation of DATE, 1999.

[Pec05] F.Pêcheux, C.Lallement, A.Vachoux, "VHDL-AMS and Verilog-AMS as Alternative Hardware Description Languages for Efficient Modeling of Multidiscipline Systems", IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 24, NO. 2, FEBRUARY 2005.

[Pri05] PrismTech Productivity Tools and Middelware, disponible en ligne, http://www.prismtech.com/

[Pto05] Service en ligne, http://ptolemy.eecs.berkley.edu

[Pto91] J. Buck, S. Ha, E. A.lee and D.G. Messerschmitt, "Ptolemy: a mixed paradigm simulation/prototyping platform in C++", C++ Conference, Santa Clara, CA, November 1991.

[QoS99] Service en ligne, « Sécurisation des liens d'interconnexion », http://www.art-telecom.fr/dossiers/spectech/4-99_1-0.doc

[Roo05] Room, service en ligne, http://www.smartdraw.com/tutorials/software-room/room.htm

[Roo94] Selic.B, Gullekson.G, Ward.P.T, "Real-Time Object-Oriented Modeling", Wiley Professional Computing edition, 1994.

[Rpc05] Service en ligne, http://www.faqs.org/rfcs/rfc1831.html.

[Sam04] M. Samyn, S. Meftali, J.L. Dekeyser, "MDA based, systemc code generation, applied to intensive signal processing applications". Forum on specification and Design Languages, Lille, France, September 13-17, 2004.

[Sar05a] A.Sarmento, « Génération Automatique de Modèles de Simulation pour la Validation de Systèmes Hétérogènes », Thèse de Doctorat UJF, Spécialité Microélectronique, à soutenir fin 2005.

[Sar05b] A. Sarmento, L.Kriaa, A. Grasset, M.W. Youssef, A. Bouchhima, F. Rousseau, W. Cesario, A.A. Jerraya, "Service Dependency Graph, an Efficient Model for Hardware/Software Interfaces Modeling and Generation for SoC Design" , Hardware/Software Codesign and SystemSynthesis (CODES-ISSS), New York, USA, Septembre 19-21, 2005.

[Sch99] S. Schneider, "Concurrent and real time systems the CSP approach", Wiley Professional Computing Edition, September 1999.

[Sdl87] Computer networks and ISDN Systems. CCITT SDL, 1987.

[Sel94] B. Selic, G. Gullekson, P.T.Ward, "REAL-TIME OBJECT-ORIENTED Modeling", Wiley Professional Computing edition, 1994.

[Sgr00] M.Sgroi, L.Lavagno, A.Sangiovanni, "Formal Models for Embedded System Design", IEEE design & test computers, April-June 2000.

[Sim05] JB. Dabney, TL. Harman, "Mastering Simulink" , Prentice Hall; 1st edition, September 2003.

[Sim98] D.Sima, T. Fountain, P.Kacsuk, "Advanced computer architectures a design space approach", Biddles Ltd.,Guildfort and King'sLynn edition, 1998.

[Sma05] Dolphin Integration, service en ligne, http://www.dolphin.fr/ , 2005.

[Soc05] Sockets, service en ligne, http://www.commentcamarche.net/sockets/sockintro.php3

[SyC02] SystemC Design language, disponible en ligne, http://www.systemc.org, 2002.

[Syn05] Définition : « Modèle Synchrone et modèle asynchrone », service en ligne, http://searchsmb.techtarget.com/sDefinition/0,290660,sid44_gci213080,00.html.

[Tan99] Andrew S. Tanenbaum, "Structured Computer Organization", Prentice Hall, Inc, Fourth edition 1999.

[Tsi03] A.Tsikhanovich, E.M.Aboulhamid, G.Bois, "Object-Oriented Techniques in Hardware modeling using SystemC", NEWCAS, Montreal, Canada 2003, June 17-20.

[Uml02] Rational UML, Disponible en ligne, http://www.rational.com/uml/, 2002.

[Vhd93] "IEEE Standard VHDL Language Reference manual", IEEE, Institute of Electrical and Electronically Engineers, 1993, STD 1076-1993.

[Xi05] C. Xi, F. Pétrot, W. Cesário, A. Bouchhima, A.Jerraya, "A New HW/SW Interfaces Refinement Approach from a Unified Abstract Model", submitted at ASPDAC 2006.

[Xml05] "XML 1.0 Specification", 2nd ed., W3C Recommendation, October 2000, diponible en ligne, http://www.w3c.org/XML

[You05] W. Youssef, « Exploration et conception et d'interfaces logiciel/matériel, pour les systèmes sur puce multiprocesseurs hétérogènes et hautement parallèles, basées sur les modèles de programmation parallèle de haut niveau », Thèse de Doctorat UJF, Spécialité Informatique, à soutenir fin 2005.

[Zen05a] A. Zenati, Y. Ammar, K. Matou, S. Basrour, "Global simulation and cosimulation of self powered microsystems", DTIP, Montreux, Suisse June1-3, 2005.

[Zen05b] A.zenati, « Modélisation et simulation multi langage d'un SoC/SoP incluant des MEMS », thèse microélectronique INPG, soutenance prévue fin 2005.

[Zit93] M. Zitterbart, "A Model for Flexible High performance Communication Subsystems", IEEE Journal on selected areas in communication, VOL. 11, NO, 4, MAY 1993.

RESUME

Les systèmes sur puces sont un ensemble de composants hétérogènes. La conception des ces systèmes peut être vue sous deux aspects : la conception des composants et leurs intégration dans le même système. La conception des composants nécessiterait des compétences de différentes équipes de développement selon les besoins, les compétences et, les fonctionnalités, etc. L'intégration de ces composants requerrait alors (1) des connaissances multidisciplinaires des différents concepts utilisés par les composants, (2) des modèles globaux du système pour la description des composants et de leurs interactions et (3) un modèle de validation tout au long du flot de conception par assemblage de composants. Toutefois, la diversité des descriptions et de modélisations des différents composants rendent l'intégration un processus très complexe et laborieux. L'objectif de cette thèse est de généraliser le concept de modélisation et de validation globale des systèmes embarqués sur puce afin de faciliter le processus d'intégration et d'assemblage de composant de système sur puce. Cette généralisation concerne (1) les concepts communs pour la modélisation des systèmes hétérogènes avant la synthèse de leurs circuits notamment pour la simulation, l'exécution, etc. et (2) la définition d'un modèle de d'exécution global pour la validation des systèmes hétérogènes par composition d'autres modèles. Les concepts proposés dans cette thèse sont validés à travers trois applications complexes : un centre d'informations automobile, un modem d'une chaîne audio 802.16 et un système de radio définie par logiciel.

MOTS-CLES

Modélisation, Validation, Systèmes hétérogènes, intégration et assemblage, composants et interconnexions.

TITLE

Modeling and Validation of Heterogeneous Systems:Definition of an Execution Model

ABSTRACT

Systems on chip are an assembly of heterogeneous components. Their design needs two steps: components design and components integration. The integration process needs (1) a multidisciplinary and deep conceptual skills, (2) a global model for the description of the components and their integration and (3) a suitable global environment that allows fast and effective validation of heterogeneous systems. However, this design seems to be complicated and tedious work.

Our work is to generalize concepts of modeling and validation process of heterogeneous systems on chip in order to assist integration process and components assembly. This generalization is focused on (1) common concepts of modeling heterogeneous systems before synthesizes process (for simulation, execution etc.) and (2) definition of global execution model for validation of heterogeneous systems. The proposed concepts were validated using complex applications: a car information card, an audio tool chain 802.16 modem and a software defined radio system.

INTITULE ET ADRESSE DU LABORATOIRE

Laboratoire TIMA, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France.

ISBN : 2-84813-063-6 (version brochée)
ISBNE : 2-84813-064-4 (version électronique)