Modèles d'intégration d'outils et de composants logiciels/matériels pour la conception des systèmes hétérogènes embarqués

A. Dziri

INSTUT NATIONAL POLYTECHNIQUE DE GRENBOLE

THESE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : Microélectronique

préparée au laboratoire **TIMA** dans le cadre de l'Ecole Doctorale
"Electronique, Electrotechnique, Automatique, Télécommunication, Signal"

Présenté et soutenue publiquement par

Mohamed-Anouar DZIRI

le 26 Mai 2004

Modèles d'intégration d'outils et de composants logiciel/matériel pour la conception des systèmes hétérogènes embarqués

Directeur de thèse
Ahmed Amine Jerraya

JURY

M. Régis Leveugle	**Président**
M. Jean-luc Dekeyser	**Rapporteur**
M. Joseph Borel	**Rapporteur**
M. Ahmed Amine Jerraya	**Directeur de thèse**
M. Wander Oliveira Cesário	**Co-encadrant**
M. Flávio Rech Wagner	**Examinateur**

A mes parents,
A Leila, et
A toute ma famille

Remerciement

Je remercie Monsieur **Ahmed Amine Jerraya**, Directeur de recherche au CNRS et responsable du groupe SLS du laboratoire TIMA d'avoir encadrer ce travail de thèse avec beaucoup de compétence et d'enthousiasme. Je lui remercie pour m'avoir accueilli dans son équipe, pour sa disponibilité et pour ses conseils précieux pendant les moments les plus difficiles de mon travail. Qu'il trouve ici l'expression de ma profonde reconnaissance.

Je tiens à remercier Monsieur **Pierre Gentil**, Directeur de l'école doctorale d'Electronique, d'Electrotechnique, d'Automatique et de Traitement de Signal (EEATS), de m'avoir donner la chance de suivre mes travaux de recherche au sein de son école doctorale.

Je tiens à remercier Monsieur **Bernard Courtois**, Directeur de recherche au CNRS et directeur du laboratoire TIMA, de m'avoir accueilli au sein de son laboratoire.

Je remercie Monsieur **Régis Leveugle**, Professeur à l'Institut National Polytechnique de Grenoble, pour l'honneur qu'il m'a accordé en acceptant de présider le Jury de ma thèse.

Je tiens à remercier Monsieur **Jean-luc Dekeyser**, Professeur à l'université Lille1 et responsable de l'équipe West du laboratoire LIFL, et Monsieur **Joseph Borel**, Docteur d'état en science physique et ex. Vice-président de ST Microélectronique, d'avoir juger mon travail en acceptant d'être rapporteurs et pour l'intérêt qu'ils ont portés à ce travail.

Je tiens à exprimer ma grande gratitude à Monsieur **Wander Oliveira Cesário**, Docteur ingénieur au sein de l'équipe SLS du laboratoire TIMA, pour son co-encadrement, pour sa participation au Jury de cette thèse, pour sa grande disponibilité et sa patience. Je lui remercie également pour l'aide qu'il m'a apporté aux moments les plus difficiles de mon travail. J'espère que cette thèse soit à la hauteur de ses efforts.

Que Monsieur **Flávio Rech Wagner**, Professeur à l'Universidade Federal do Rio Grande do Sul reçoit, ici, l'expression de mon meilleur respect pour sa participation au Jury de cette thèse, pour ses conseils pertinents et pour le suivit continu de l'avancement de mon travail.

Je tiens à remercier Monsieur **Frédéric Rousseau**, Maître de Conférence à l'université Joseph Fourrier de Grenoble et membre de l'équipe SLS, pour l'attention qu'il a accordé à mon travail, pour sa disponibilité et pour son aide lors de la rédaction de ce manuscrit.

Je souhaite exprimer ma reconnaissance à Monsieur **Mounir Zrigui**, Maître de Conférence à la faculté des sciences de Monastir, pour son soutien et son encouragement tout au long de mon travail de recherche.

Je tiens à exprimer mon respect à Monsieur **Règis Beuscart**, Professeur à l'université Lille2, pour m'avoir accueilli au sein de son laboratoire CERIM en tant que enseignant ATER et pour la confiance qu'il m'a accordé.

Je remercie tous ceux qui m'ont aidé à découvrir l'enseignement. Je remercie particulièrement Monsieur **Philipe Morat** et Monsieur **Philipe Genoud**.

Que **Samy Meftali**, Maître de Conférence à l'université Lille1, et **Samir Belfkih**, enseignant à l'université Lille2, sachent que je n'oublierai jamais leur soutien et leur amitié lors de mon séjour à Lille.

Enfin, une pensé particulière à tous mes amis de Grenoble et mes collègues du laboratoire TIMA et du laboratoire CERIM.

Table des matières

Liste des Figures

Liste des Tableaux

Chapitre 1. Introduction

Sommaire

1.1. Contexte : la conception des systèmes hétérogènes embarqués

La technologie de fabrication des circuits intégrés a permis de passer des composants spécifiques ASIC (*Application Specific Integrated Circuit*) aux systèmes embarqués sur une seule puce (*System-on-Chip – SoC*). Grâce à cette évolution, les SoC peuvent contenir ce que l'on mettait sur plusieurs puces ou cartes il y a une dizaine d'années. Ces systèmes intègrent plusieurs processeurs, de la mémoire, différents blocs IP (*Intellectual Property*), ainsi que des réseaux de communication complexes. Le tout étant intégré sur la même puce. Ces puces sont embarquées dans des systèmes très spécifiques qui nécessitent une miniaturisation, tels que la téléphonie mobile, l'automobile, l'aviation, les jeux vidéo, etc. Ces systèmes se situent donc au sommet de la technologie moderne et se trouvent soumis à des fortes contraintes de performances, de coût de production et de temps de mise sur le marché. Pour respecter ces contraintes, les méthodologies de conception récentes imposent à ces systèmes d'être construits par un assemblage de composants préconçus. Le concept de la réutilisation de composants IP existants est très répandu et permet de maîtriser la complexité croissante des SoC. Cependant, la conception de tels systèmes, à partir d'un modèle abstrait, représente encore plusieurs défis :

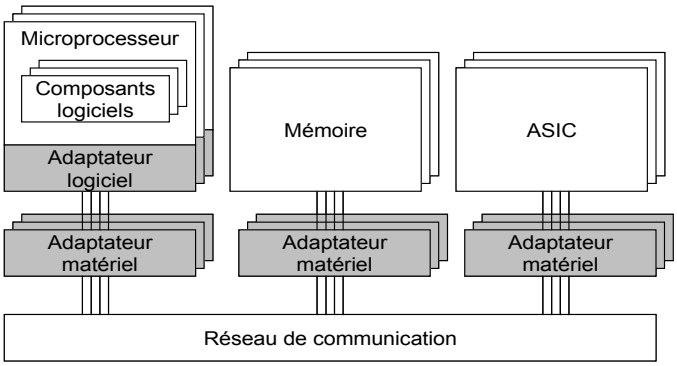

Figure 1. Architecture d'un système multiprocesseur monopuce

- l'architecture des systèmes sur puce est composée de différents composants logiciel/matériel hétérogènes, comme l'indique la Figure 1. Généralement, ces composants sont le fruit d'une stratégie de réutilisation qui a pour but de manier la complexité croissante des SoC. Cependant, plusieurs de ces composants sont fournis par des sources diverses et ont des domaines d'application spécifiques. L'intégration de ces composants dans un modèle globale du système requiert une adaptation de leur interface au réseau de communication embarqué. Cette adaptation nécessite la construction d'adaptateurs (*wrappers*) divers et très sophistiqués. En plus, des

modèles de composants, décrits en utilisant des langages et des styles différents, existent. Cela rend l'intégration de ces modèles de composants dans le système très difficile.

- un outil de conception complet pour les SoC n'existe toujours pas. Un flot de conception complet pour les systèmes sur puce inclut plusieurs étapes de conception et plusieurs niveaux d'abstraction, comme le montre la Figure 2. La construction d'un tel flot nécessite l'intégration de plusieurs outils de conception provenant de différentes sources et ayant des domaines d'application variés. Bien que les outils pour la conception des SoC soient disponibles, chaque outil utilise des formats et des modèles spécifiques, ce qui rend le modèle produit par un outil inconvenable pour être utilisé par un autre outil. Ce manque d'interopérabilité entre les outils et de compatibilité entre les modèles rend très difficile l'utilisation de plusieurs outils dans un même flot de conception. La difficulté principale de l'intégration d'outils consiste à les faire communiquer en adaptant leurs différents modèles dans le cadre d'un seul flot de conception complet.

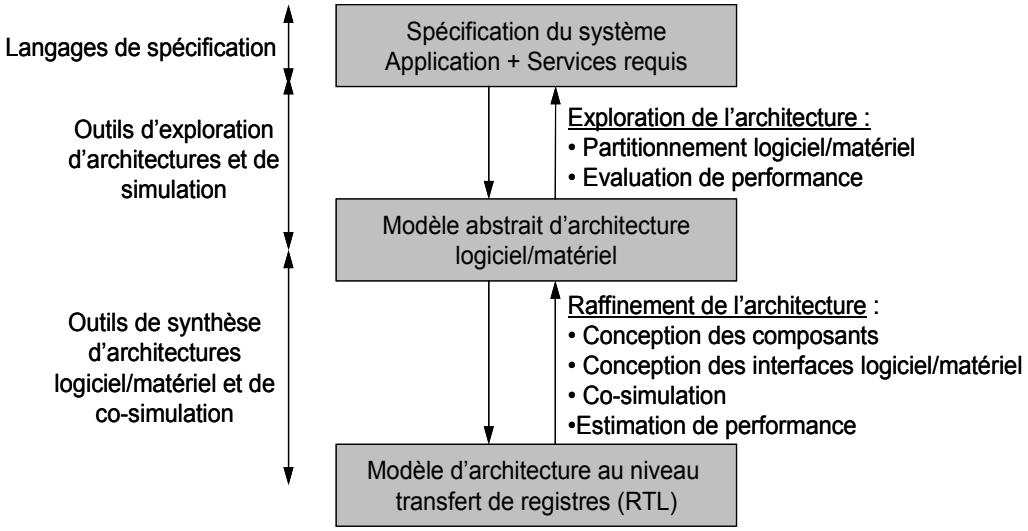

Figure 2. Flot de conception des SoC

- la manipulation d'outils et de composants logiciel/matériel dans un flot de conception est un travail très fastidieux et source d'erreurs, ce qui est très coûteux en terme de temps de conception.

1.2. Objectif

L'objectif général de cette thèse est de proposer un environnement de conception ouvert capable de :

- faciliter la manipulation des composants logiciel/matériel ;
- faciliter l'interfaçage entre des outils.

Le premier objectif nécessite des techniques de composition capables de manipuler des composants logiciel/matériel ayant des interfaces et des granularités différentes. Le deuxième objectif réclame une interopérabilité entre les outils de conception.

Les résultats de cette thèse se basent sur les travaux du groupe SLS[1]. S'étant déroulé en parallèle avec plusieurs autres thèses, ce travail constitue une réflexion permettant d'unifier les différents travaux suivants :

- les travaux de Lovic Gauthier [Gau 01] et de Damien Lyonnard [Lyo 03] ont permis de définir des méthodes de composition de composants d'interface afin de construire des adaptateurs permettant d'adapter les composants à leur environnement ;
- les travaux de Gabriela Nicolescu [Nic 02] ont permis de définir une approche de composition de sous-systèmes hétérogènes en généralisant la notion d'adaptateur ;
- d'autres travaux du groupe [Ces 01] ont permis de définir COLIF un modèle de représentation des interconnexions dans le cas des systèmes hétérogènes. Les différents outils du groupe SLS sont développés de manière indépendante et intégrés grâce à ce modèle.

L'ensemble de ces travaux a rendu possible les travaux de cette thèse.

1.3. Contributions

En partant des travaux existants, cités ci-dessus, ce travail présente quatre contributions : (1) Etude des méthodologies de réutilisation de composants logiciel/matériel et d'outils de conception. Cette étude nous aidera pour la construction d'un environnement ouvert, pour l'intégration des composants logiciel/matériel et d'outils de conception, autour d'un format intermédiaire dédié à la spécification des systèmes hétérogènes ; (2) Présentation et analyse du flot ROSES pour la conception et la validation des systèmes hétérogènes embarqués ; (3) La proposition d'une méthodologie d'intégration de composants logiciel/matériel pour la conception et la validation des systèmes hétérogènes embarqués ; et (4) La proposition d'un

[1] System Level Synthesis

modèle d'intégration d'outils de conception pour la construction de flots de conception des systèmes hétérogènes embarqués.

Chaque contribution sera présentée plus en détail par la suite.

1.3.1. Etude des méthodologies de réutilisation de composants logiciel/matériel et d'outils de conception

Les architectures et les flots de conception des systèmes embarqués d'aujourd'hui sont devenus de plus en plus complexes. Pour maîtriser cette complexité et pour améliorer la productivité, la conception de tels systèmes tend vers la réutilisation des composants préconçus et la réutilisation des outils de conception provenant d'origines diverses. Dans une première partie nous proposons une étude des concepts fondamentaux de la réutilisation des composants. Dans une deuxième partie nous proposons une étude des concepts fondamentaux de la réutilisation des outils. Ces études mettent l'accent sur l'étape d'intégration des composants et des outils. Une présentation des approches académiques et industrielles existantes est faite afin de pouvoir analyser les avantages et les inconvénients de chaque solution. Cette étude a permis de définir un environnement ouvert à l'intégration d'outil et de composants logiciel/matériel autour d'un format de représentation intermédiaire.

1.3.2. Analyse du flot ROSES dédié à la conception et la validation des systèmes hétérogènes embarqués

Le flot ROSES implémente une méthodologie de conception avec un ensemble d'outils correspondant, qui visent l'intégration de composants logiciel/matériel. ROSES suit une approche qui permet la génération d'adaptateurs logiciels et matériels. Cette approche permet une réutilisation effective des composants logiciel/matériel. Nous proposons une présentation et une analyse de ce flot. Cette analyse a permis de constater que le flot ROSES nécessite un grand travail de réorganisation pour devenir un environnement de conception ouvert à : (1) l'intégration des composants logiciel/matériel et (2) l'intégration d'outils.

1.3.3. Méthodologie d'intégration des composants logiciel/matériel pour la conception et la validation des systèmes hétérogènes embarqués

La plupart des méthodologies de conception des systèmes monopuce supposent la réutilisation des composants préconçus. Cependant, puisque ces composants sont hétérogènes leur intégration nécessite des sous-systèmes d'interface logiciel/matériel très sophistiqués (adaptateurs). Ces sous-systèmes peuvent aussi être construits par un assemblage de

composants d'interface élémentaires et préconçus. Désormais, la conception et la validation des systèmes sur puce nécessite des techniques de composition de composants à deux niveaux différents : (1) au niveau système, pour construire des grands systèmes par une composition de composants logiciel/matériel et (2) au niveau interface, pour construire des sous-systèmes d'interface par un assemblage de composants d'interface logiciel/matériel élémentaires et homogènes. Nous proposons une méthodologie unifiée pour l'intégration automatique de composants permettant aux concepteurs une réutilisation effective des composants préexistants à tous les deux niveaux sous-jacents. Les sous-systèmes d'interface sont générés automatiquement et sont ensuite traités eux-mêmes en tant que composants au cours de la phase d'intégration au niveau système.

1.3.4. Modèle d'intégration d'outils de conception pour la construction de flots de conception des systèmes hétérogènes embarqués

La réutilisation des outils est aussi difficile que celle des composants. Actuellement, la construction des flots de conception se fait par une collaboration de plusieurs outils. Cette collaboration implique des échanges de modèles de conception entre les différents outils au cours du processus de conception. Les outils sont souvent très spécialisés et proviennent de sources différentes. Ils produisent/consomment aussi des modèles variés, et utilisent des formats et des représentations de données en interne qui sont complètement différents. La principale difficulté est d'assurer une interopérabilité entre les outils et de garantir l'intégrité du modèle du système. Pour remédier à ce problème, nous proposons une méthodologie d'intégration automatique d'outils de conception. Cette méthodologie est basée sur la définition d'un modèle générique dédié à l'intégration d'outils autour d'un format de représentation intermédiaire appelé COLIF. Différents traducteurs sont utilisés pour assurer un lien automatique entre les outils à intégrer et le format intermédiaire. Ce dernier doit être cohérent et complet pour le bon fonctionnement du processus de conception. Pour assurer la cohérence et l'intégrité du modèle du système, un processus de vérification a été mis en place à travers un outil appelé COLIF-*Checker*. Cet outil a été développé au cours de ces travaux de thèse, il est configurable par le concepteur en utilisant une base de données contenant un ensemble de règles.

1.4. Plan du document

Ce manuscrit est composé de quatre chapitres. Le deuxième chapitre présente : (1) une étude des méthodologies de réutilisation de composants logiciel/matériel, et (2) une étude des

méthodologies de réutilisation d'outils pour la conception des systèmes hétérogènes embarqués. Le troisième chapitre décrit et analyse le flot de conception ROSES. Le quatrième chapitre présente : (1) une méthodologie d'intégration de composants logiciel/matériel pour la conception et la validation des systèmes hétérogènes embarqués, et (2) un modèle d'intégration d'outils de conception pour la construction de flots de conception. Le dernier chapitre illustre l'utilisation des techniques développées dans le cadre de cette thèse pour :

- l'intégration de l'outil VCC (*Virtual Component Co-design*) de cadence dans l'environnement ROSES. Ceci, nous a permis d'utiliser les capacités d'exploration d'architectures de VCC et celles d'intégration de composants logiciel/matériel de ROSES pour la conception d'un système multiprocesseur – le modem VDSL ;
- l'intégration d'un IP de communication de haut niveau en utilisant la modélisation des composants au niveau transaction – le modèle TLM (*Transaction Level Modeling*).

Chapitre 2. Méthodologies de réutilisation de composants logiciel/matériel et d'outils pour la conception des systèmes hétérogènes embarqués

Sommaire

2.1 Introduction

Grâce à la technologie des semi-conducteurs, nous pouvons construire des systèmes embarqués en intégrant plusieurs centaines de millions de transistors sur une seule puce. Cette capacité d'intégration va continuer à doubler tous les dix-huit mois selon les prévisions de l'industrie microélectronique [Roa 01]. La technologie sous-jacente promet non seulement des nouvelles capacités d'intégration de composants sur une seule puce, mais aussi présente des défis importants aux concepteurs. En résultat, plusieurs développeurs d'ASIC et des vendeurs de semi-conducteurs font évoluer leurs méthodologies de conception, cherchant un moyen pour la réutilisation effective d'un grand nombre de composants déjà existants.

Les outils et les méthodologies de conception courants sont insuffisants pour développer complètement tous les composants des grands systèmes sur puce à partir de zéro (*from scratch*). De plus, il y a une pression importante pour maintenir la taille des équipes et les plannings de conception constants alors que la complexité des systèmes ne cesse de croître. Les outils ne fournissent pas les gains de productivité nécessaires pour aller aussi vite que l'accroissement des capacités d'intégration. La réutilisation – l'utilisation de composants préconçus et pré-vérifiés – est l'opportunité la plus appropriée pour faire le lien entre la capacité d'intégration disponible et la productivité des concepteurs.

La complexité des systèmes embarqués évolue très rapidement de façon à ce qu'aucune méthodologie de conception ne peut fournir une solution complète pour ce problème extrêmement dynamique. La réutilisation d'outils spécialisés est aussi une opportunité équivalente à celle de la réutilisation des composants préconçus. Cette réutilisation permet de tirer des bénéfices importants à partir des performances et des compétences des outils existants.

La conception des systèmes embarqués nécessite des connaissances dans des domaines différents, ainsi que l'utilisation de composants variés, et parfois très spécifiques, qu'il est impossible à une seule équipe de concepteurs de posséder. La réutilisation de composants et d'outils est la seule issue à ces problèmes. En effet, elle permet de bénéficier des compétences et des expertises déjà existantes pour achever plus efficacement des étapes de conception et pour répondre à de fortes contraintes de performance et de temps de mise sur le marché.

Ce chapitre est organisé de la manière suivante. La section suivante introduit l'architecture en couches des systèmes hétérogènes embarqués. La troisième section présente le flot de conception idéal de ces systèmes. Ensuite une introduction à la réutilisation de composants et d'outils de conception est faite. La cinquième section présente les concepts de

la réutilisation de composants, et analyse les différentes méthodologies ainsi que les styles de conception existants. Enfin, la dernière section présente les concepts de réutilisation d'outils, introduit le problème d'interopérabilité entre les outils, analyse les approches proposées actuellement comme solutions possibles pour l'interopérabilité entre les outils et présente un état de l'art sur les outils de conception existants.

2.2. Architecture des systèmes hétérogènes embarqués

La Figure 3 montre l'architecture, structurée en couches, d'un système hétérogène embarqué. Cette structure permet de maîtriser la complexité de tels systèmes. La partie matérielle est composée de deux couches. En partant du bas vers le haut :

- la première couche contient les principaux composants utilisés par le système. Ces derniers sont composés de composants standard tels que des processeurs (DSP, MCU) et des composants matériels « IP – *Intellectual Property* » (ASIC, mémoire).

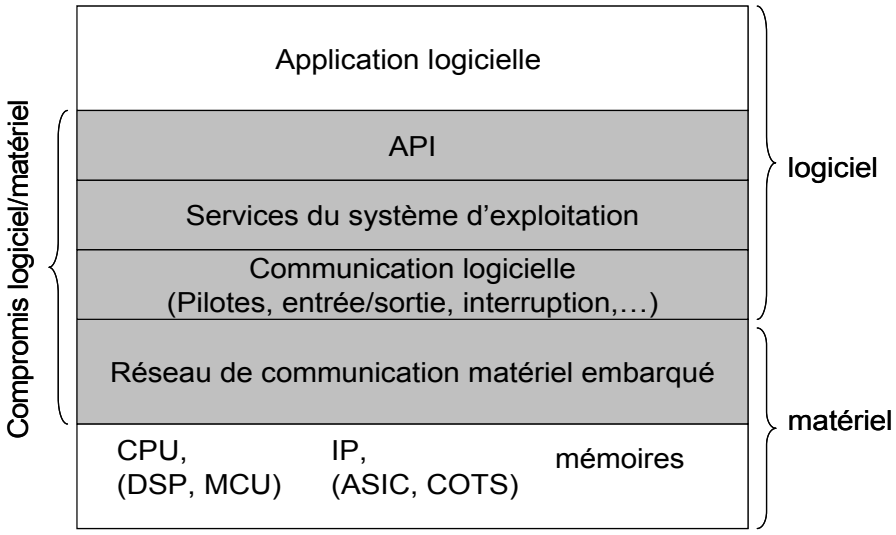

Figure 3. Représentation d'un système hétérogène embarqué sous forme de couches

- la deuxième couche présente la communication matérielle embarquée sur la puce. Cette couche est nécessaire pour l'interaction entre les composants et contient tous les dispositifs nécessaires pour se faire. Elle peut contenir un réseau de communication complexe, allant du simple bus au réseau de communication de paquets. Bien que l'interface du réseau de communication suive des standards existants, il est souvent nécessaire d'ajouter des couches d'adaptation. Ces couches d'adaptation se situent entre l'interface du réseau de communication et les interfaces des composants de la première couche.

La partie logicielle embarquée se compose aussi de deux couches principales :

- la première couche présente le code de l'application logicielle. Elle est généralement conçue sous forme de composants logiciels complètement indépendants de la partie matérielle.
- la deuxième couche présente la couche de gestion de ressources matérielles, elle permet d'adapter l'application logicielle à l'architecture matérielle. Cette couche est généralement organisée en trois sous-couches différentes [Yoo 03]:
  - une couche API (*Application Programming Interface*) qui permet d'isoler l'application logicielle du reste des couches logicielles ;
  - une couche logicielle indépendante de la partie matérielle, qui présente plus précisément des services de haut niveau fournis par un système d'exploitation, tels que l'ordonnancement des tâches et les primitives de communication de haut niveau ;
  - une couche logicielle très dépendante de la partie matérielle, qui se contient essentiellement :
    - des pilotes d'entée/sortie permettant l'accès et le contrôle de la partie matérielle ;
    - le code du démarrage (*boot code*) ;
    - des parties du système d'exploitation telles que le code de changement de contexte (*context switching code*), ainsi que des algorithmes spécifiques à certains périphériques qui inter-réagissent directement avec le matériel.

La standardisation ou la normalisation de la couche API, qui peut être vue comme étant une collection de services offerts par un système d'exploitation, est essentielle pour la réutilisation du logiciel au dessus et au dessous de cette couche. Par contre, les couches de gestion de ressources fournies par les systèmes d'exploitation standard sont généralement trop volumineuses pour être embarquées.

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

2.3. Conception des systèmes hétérogènes embarqués

La Figure 4 présente un flot de conception idéal. Ce flot démarre à partir d'une spécification au niveau système pour atteindre une architecture au niveau transfert de registres

(RTL), à partir de laquelle la synthèse des composants logiciel/matériel peut être lancée en utilisant les outils de conception conventionnels. La première étape du flot de conception est un contrat entre le client d'une part, et les concepteurs du système d'autre part se chargeant d'une modélisation informelle de l'application. Ensuite, ces derniers construisent une spécification formelle du système – il s'agit d'un modèle fonctionnel ou comportemental – que le client peut valider. Après, ils fixent l'architecture du système, en utilisant typiquement des composants IP (des processeurs, des mémoires, des bus, etc.), et décident de l'affectation des blocs fonctionnels aux composants architecturaux pour la fonctionnalité du système. Cette étape utilise généralement un modèle de spécification exécutable qui permet aux concepteurs d'aller à travers une boucle d'analyse des performances du système. Pour obtenir un modèle de spécification exécutable à ce stade, les concepteurs système construisent un modèle de simulation en utilisant des profils logiciels et des modèles abstraits pour les modules matériels.

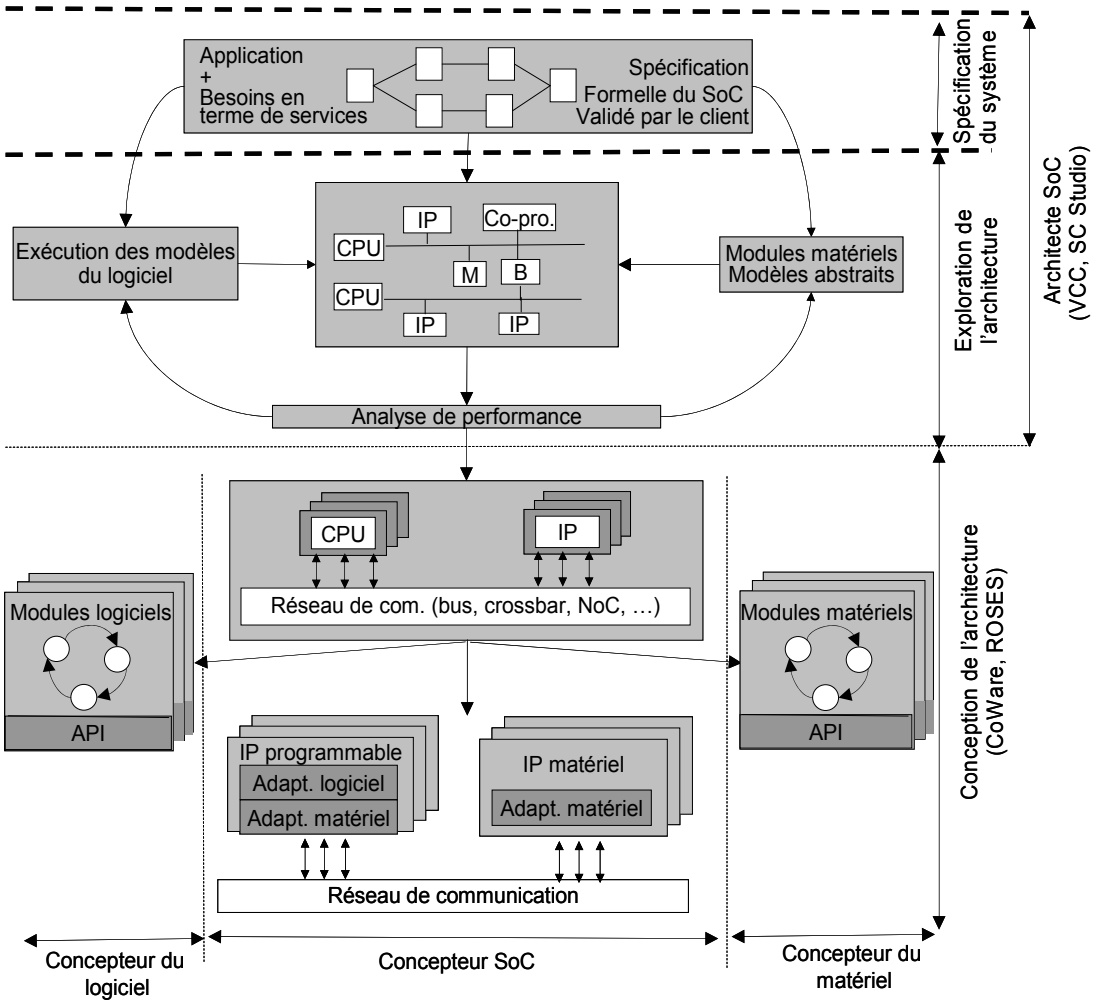

Figure 4. Flot de conception système idéal

L'exploration de l'architecture du système fixe la spécification des composants matériels (par exemple, la sélection d'unités de calcul existantes ou des IP spécifiques), le partitionnement des tâches fonctionnelles entre les composants logiciels et les composants matériels, et la structure globale du réseau de communication embarqué. Cette étape produit une architecture abstraite, qui ne contient pas tous les détails d'interfaces entre les composants et qui est utilisée comme un modèle architectural qui guidera l'étape de conception suivante : la conception de l'architecture SoC.

Les concepteurs des SoC sont obligés d'intégrer les composants logiciels et les composants matériels tout en respectant les contraintes de performance décrites au niveau de l'architecture abstraite. Cette étape aboutit à une architecture RTL (appelé aussi micro-architecture), ce qui implique la génération de tous les interfaces logicielles et matérielles au niveau de précision du cycle d'horloge. En parallèle, les concepteurs du logiciel et ceux du matériel peuvent implémenter ces composants conformément à l'architecture abstraite.

A cause de cette diversité de modèles, d'étapes de conception, et des niveaux d'abstraction, un flot de conception système ne peut être réalisé sans le support d'un ensemble d'outils. Une interopérabilité entre ces outils doit être assurée pour préserver l'intégrité du modèle du système et pour accélérer le processus de conception.

2.4. Réutilisation

A cause de la complexité croissante des systèmes embarqués, les fortes contraintes de performance et de temps de mise sur le marché, et la nécessité d'avoir des experts en différents domaines, la réutilisation est devenue l'issue principale pour mener à bien la conception de ces systèmes :

- la réutilisation des blocs développés en interne par une équipe de conception ainsi que des composants IP obtenus d'une source extérieure doivent être intégrés afin d'avoir un environnement de conception compétitif ;
- la réutilisation d'outils ou de flots de conception existants est nécessaire pour bénéficier de leur compétence et de leur performance afin d'accomplir les différentes étapes de conception de façon automatique. Ceci nécessite une interopérabilité entre les différents outils et flots de conception.

2.5. Réutilisation de composants

Les composants IP peuvent être sous plusieurs formes [VSIA]. Ils peuvent être de type « *hard* », quand toutes les portes et les interconnexions sont placées et routées, de type

« *soft* » lorsqu'ils sont représentés au niveau transfert de registres, ou de type « *firm* » quand leur description RTL et quelques placements physique sont fournis. Le processus de réutilisation de composants est très complexe. Pour maîtriser cette complexité, ce processus est généralement décomposé en plusieurs étapes dont quelques unes sont supportées par les fournisseurs de composants, qui créent des composants qualifiés, et les autres étapes sont supportées par les consommateurs de composants, qui cherchent et évaluent les composants réutilisables et les intègrent dans leur système.

Les producteurs et consommateurs de composants IP peuvent appartenir à une même organisation, alliée par une technologie Intranet, ou à des organisations différentes, échangeant les composants à travers un marché électronique ouvert qui utilise la technologie Internet. Les étapes principales du processus de la réutilisation sont présentées dans les sections suivantes.

2.5.1. Les producteurs

- ***création des composants*** **:** les composants doivent être créés en suivant des directives ou des guides de conception (par exemple [Kea 02]). Ainsi ils peuvent être facilement réutilisés, de préférence en utilisant des langages de conception standard ;
- ***qualification des composants*** **:** les fournisseurs de composants doivent suivre une méthodologie et utiliser l'ensemble d'outils correspondant pour garantir une qualification des composants attendue par les consommateurs (par exemple [See 02]) ;
- ***classification des composants*** **:** les fournisseurs classifient leurs composants dans des catalogues en ligne (par exemple [Des 97] et [Fau 99]) et selon des critères adéquats, pour faciliter la tâche des consommateurs durant une éventuelle recherche d'un composant en vue d'un achat possible.

2.5.2. Les consommateurs

- ***recherche de composants*** **:** les consommateurs de composants trouvent les composants ayant les fonctionnalités et les propriétés appropriées en effectuant une recherche dans des catalogues mis à leur disposition par les producteurs de composants ;
- ***transfert de composants*** **:** quand les composants recherchés sont trouvés, ils doivent être délivrés par le producteur avec toutes les informations nécessaires pour leur évaluation et leur intégration. Ces informations incluent les données de conception, les

documentations, les spécifications, les programmes de test, etc. (par exemple le *Virtual Component Transfer standard* de VSIA [VCT]) ;

- ***évaluation de composants* :** généralement, une évaluation plus précise des composants est nécessaire avant leur intégration dans le système. Ceci peut être accompli par une instanciation des composants dans des programmes de test (*testbench*) correspondants et procéder à une simulation (par exemple en utilisant une infrastructure de co-simulation basée sur le Web [Dal 99]) ;
- ***intégration de composants* :** la conception des systèmes hétérogènes embraqués commence à partir d'une spécification de haut niveau, qui peut être validée par le client. Cette spécification doit déjà suivre une séparation claire entre la partie calcul et la partie communication du système [Row 97], ceci permet une conception et une évaluation concurrentes des composants. Ensuite, une architecture qui abstrait les détails de la communication, appelée architecture abstraite, est utilisée pour évaluer la fonctionnalité du système. L'intégration de composants est un ensemble de tâches nécessaires pour l'assemblage de composants préconçus afin de répondre aux exigences du système. Cette intégration permet une implémentation de l'architecture abstraite du système.

2.5.3. Méthodologies et styles de conception des systèmes hétérogènes embarqués

2.5.3.1. Séparation entre calcul et communication

Dans le cas d'une spécification des systèmes hétérogènes embarqués, les différents composants et/ou les interconnexions de communication sont spécifiés à des niveaux d'abstraction différents ou à l'aide de langages de spécification différents. Afin de permettre la spécification des systèmes hétérogènes embarqués, des nouveaux concepts sont souvent utilisés. Parmi ces derniers : la séparation du comportement (ou du calcul) de la communication connu sous le terme « *interface-based design* » [Row 97]. Ce concept permet l'abstraction de l'interconnexion entre les composants hétérogènes du système.

2.5.3.2. Niveaux d'abstraction de la communication

Un système hétérogène embarqué est spécifié en utilisant plus ou moins de détails dépendant du niveau d'abstraction utilisé par le modèle de spécification. Si un modèle abstrait est utilisé, des détails de description complexes peuvent être évités. Pour concevoir des systèmes complexes, il est préférable de commencer à partir d'un modèle abstrait de haut

niveau et de raffiner progressivement la spécification en diminuant le niveau d'abstraction utilisé pour modéliser le système. Plusieurs classifications sont utilisées pour les niveaux d'abstraction. Cette section présente une classification plutôt orientée vers les concepts de la communication [Nic 02]. Au niveau système, quatre niveaux d'abstraction de la communication existent : service, transaction, macro-architecture ou encore message, et le niveau transfert de registres (RTL) ou encore micro-architecture.

- au niveau service**,** le système est modélisé sous forme d'un ensemble de modules qui fournissent/requièrent des services. Le réseau de communication garantit le routage des requêtes et la synchronisation entre les modules lors d'échange de données. La primitive de communication typique sera une requête de service, comme par exemple « imprimer (fichier) ». La notion de temps est complètement abstraite, il y a seulement la notion d'ordre partiel pour les événements de communication. CORBA (*Common Object Request and Broker Architecture*) [Omg 97] est un exemple de ce niveau d'abstraction ;
- au niveau transaction, les différents modules du système utilisent des canaux de communication, qui sont dits actifs, pour communiquer entre eux. Ces canaux assurent la synchronisation et la conversion (si nécessaire) des protocoles spécifiques aux différents modules communicants. Les détails des protocoles de communication sont englobés par des primitives de communication de haut niveau (par exemple *send/receive*) et aucune hypothèse sur la réalisation des protocoles de communication n'est faite. Des exemples de langages modélisant les concepts spécifiques à ce niveau sont SDL (*System Description Language*) [Sdl 87] et ObjecTime [Obj 00]. Certaines implémentations de StateCharts [Har 87] peuvent être placées à ce niveau ;
- le niveau macro-architecture ou message est le niveau spécifique à la communication par des fils abstraits, avec des protocoles de communication pour les entrées/sorties. Un modèle de ce niveau implique par conséquent le choix d'un protocole de communication et la topologie des interconnexions. Des exemples typiques de primitives de communication spécifiques à ce niveau sont l'envoi d'un message, l'écriture d'une valeur dans un registre ou l'attente d'un événement ou d'un message. Les langages caractérisant le mieux un système à ce niveau d'abstraction sont : CSP [Hoa 85], SystemC 1.1 [Sys 00], Cossap [Syn 97] [Syn 02] et StateCharts [Har 87] ;
- au niveau RTL ou micro-architecture, le système est décrit sous forme d'un ensemble de registres, de circuits combinatoires, de circuits de contrôle, de fils et de bus physiques. L'unité de temps devient le cycle d'horloge. Les primitives de

communication sont : *set*/*reset* des valeurs des signaux par des ports matériels et l'attente d'un nouveau cycle d'horloge. A ce niveau d'abstraction, les modèles sont créés en utilisant les HDLs (*Hardware Description Languages*) tels que SystemC [Sys 00], Verilog [Moo 98] et VHDL [Iee 93].

Le Tableau 1 résume les caractéristiques principales d'un système à des niveaux d'abstraction différents. Les caractéristiques présentées sont : le média de communication, les primitives de communication, et les modèles typiques de chaque niveau d'abstraction.

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

Tableau 1. Niveaux d'abstraction pour la communication

2.5.3.3. Approche de conception basée sur une plateforme

Selon l'approche de conception basée sur une plateforme (*platform-based design*) [Keu 00], on utilise un modèle architectural souvent spécifique à un domaine d'application bien particulier. Ce modèle inclut une plateforme matérielle, qui se compose d'une structure de communication donnée, plusieurs types de composants (processeurs, mémoires, blocs matériels) et une plateforme logicielle, qui se trouve sous la forme d'une API de haut niveau. Le système embarqué est conçu par une dérivation de ce modèle. Durant cette dernière, la structure de communication, les composants et la plateforme logicielle sont tous taillés pour s'accommoder aux besoins particuliers de l'application.

2.5.3.4. Approche de conception basée sur des IP

L'approche de conception basée sur les IP (*IP-based design*) propose l'assemblage de composants logiciel/matériel réutilisables. L'étape d'intégration de composants utilise un ensemble de composants logiciel/matériel qui ont été sélectionnés pour implémenter une architecture abstraite. Le résultat de cette étape d'intégration est une micro-architecture où les composants matériels sont décrits au niveau RTL. Tandis que les composants logiciels sont décrits en utilisant un langage de programmation approprié, qui peut être directement compilé sur le processeur cible de l'architecture.

Dans une situation idéale, les composants sont directement connectés entre eux (ou à la structure de communication) de façon à se marier exactement avec la fonctionnalité désirée du système. Dans une situation plus générale, le concepteur est amené à adapter les différents composants (une étape appelée dérivation de composant « *IP derivation* ») et à synthétiser des adaptateurs (*wrappers*) logiciels et matériels pour les interconnecter. Pour les composants programmables, bien que la dérivation soit faite facilement par une re-programmation de la fonctionnalité désirée, le concepteur a besoin de développer des adaptateurs logiciels (souvent des pilotes de périphériques et/ou de bus) pour adapter l'application logicielle à l'infrastructure de communication. La génération d'adaptateurs logiciels et matériels est connue sous les termes : synthèse de la communication ou synthèse d'interfaces. En plus de ces deux types d'adaptation, l'application logicielle a besoin aussi d'être recompilée sur les processeurs et les systèmes d'exploitation de l'architecture choisie.

2.5.3.5. Approche de conception basée sur un bus standard

Dans l'approche basée sur un bus standard (*bus-based approach*), les composants communiquent à travers un ou plusieurs bus (interconnectés par des ponts « *bridges »* de bus). Puisque la spécification du bus est standardisée, des bibliothèques de composants ayant des interfaces directement compatibles avec cette spécification sont développées. Même si les composants respectent le standard du bus, des adaptateurs très simples de l'interface du bus sont souvent nécessaires [Ber 00]. Pour les composants qui ne se marient pas directement avec la spécification du bus, la construction d'adaptateurs est obligatoire. Plusieurs compagnies offrent des bibliothèques très riches de composants et des environnements de développement et de simulation spécialisés pour permettre la conception des systèmes autour de leur bus [Cor 99][Arm 99][Son 03].

2.5.3.6. Approche de conception basée sur des cœurs

Dans l'approche basée sur des cœurs (C*ore-based approach*), les composants obéissent à un standard d'interface indépendant du bus et sont par conséquence directement connectés les uns aux autres. Bien que les standards puissent supporter une grande variété de fonctionnalités, chaque composant peut avoir une interface qui contient seulement les fonctions qui lui sont applicables. Ces composants peuvent être aussi interconnectés à travers un bus. Dans ce cas, des adaptateurs standard adaptent les interfaces des composants au bus de communication.

La conception basée sur un bus et la conception basée sur les cœurs sont deux approches d'intégration qui dépendent de composants ou d'une interface de bus standardisés. Elles permettent l'intégration de composants homogènes, qui respectent un certain standard et sont directement interconnectés, sans avoir recours au développement des adaptateurs complexes. Le problème rencontré est que plusieurs standards co-existent. Ils proviennent d'organisations ou de compagnies différentes, empêchant de ce fait un vrai échange de bibliothèques de composants développées pour des standards différents.

2.5.3.7. Approche de conception basée sur des réseaux de communication embarqués NoC

Pour des raisons de flexibilité, les systèmes hétérogènes embarqués sont construits autour d'un réseau de communication embarqué NoC (*Network-on-Chip*) [Mic 02] dédié, très sophistiqué, et qui peut fournir de très grandes performances en connectant un grand nombre de composants. Dans ce cas, l'approche basée sur un bus ou sur les cœurs peut être adoptée pour connecter les composants au réseau de communication.

2.5.3.8. Intégration des composants logiciels

Les composants programmables sont très importants dans une plateforme architecturale réutilisable, puisque ça devient plus facile de tailler cette plateforme pour des applications différentes. Ceci est achevé à travers une simple adaptation du logiciel de bas niveau ou une configuration de certains paramètres liés aux composants matériels, tel que la taille de la mémoire. La partie logicielle d'un système hétérogène embarqué se compose de deux couches différentes, présentées au début de ce chapitre. Au niveau de la couche application logicielle, plusieurs composants logiciels qui se trouvent dans des bibliothèques fournissent un grand nombre de fonctions nécessaires pour implémenter cette couche. Ceci pour un domaine d'application donné. Cependant, si ces composants logiciels ne se marient pas directement avec la couche API de la plateforme, le re-ciblage de l'application logicielle sera nécessaire. Manuellement, ce processus représente un travail très fastidieux et une source d'erreurs. Pour faire face à ce problème, une technique automatique pour la synthèse du logiciel s'avère une solution concrète.

La réutilisation des composants logiciels peut être aussi obtenue au dessous de la couche API [Sha 02] : au niveau de la couche logicielle indépendante de la plateforme matérielle, les composants logiciels implémentant cette couche sont plus facilement réutilisés, en particulier si l'interface entre cette couche et la couche logicielle dépendant de la plateforme matérielle

est normalisée et cache les détails spécifiques aux composants matériels. Il existe plusieurs alternatives industrielles et académiques qui fournissent des services du système d'exploitation. Cependant le problème de la plupart des approches est qu'elles ne prennent pas en considération des besoins spécifiques aux systèmes hétérogènes embarqués, tels que la minimisation de l'utilisation de la mémoire et la consommation d'énergie. Des efforts de recherches récents proposent le développement de systèmes d'exploitation temps réel (*Real Time Operating System – RTOS*) qui contiennent seulement un ensemble minimal de fonctions nécessaire pour une application donnée [Gau 01] [Bok 00] ou incluent des techniques dynamiques pour la gestion de l'énergie [Ben 98]. Les méthodologies d'intégration de composants peuvent prendre en considération la génération des systèmes d'exploitation temps réel optimisés pour les besoins spécifiques à des applications données.

Au cours des dernières années, plusieurs efforts de standardisation visant la réutilisation des composants matériels sont apparus. Actuellement, des efforts similaires sont nécessaires pour la réutilisation des composants logiciels. VSIA [VSIA] [See 99] [Len 00] a récemment créé un groupe de travail pour s'occuper de la partie logicielle dépendante du matériel et de la conception basée sur une plateforme.

2.5.3.9. La synthèse de la communication

Des solutions pour la synthèse automatique des adaptateurs de communication, qui connectent les composants matériels ayant des interfaces incompatibles, ont été déjà proposées. Dans l'outil PIG [Pas 98], les interfaces des composants sont spécifiées en tant que protocoles décrits sous forme d'expressions régulières. Une machine d'états finis (*Finite State Machine – FSM*) est générée automatiquement. L'outil Polaris [Smi 98] génère des adaptateurs basés sur des machines d'états pour convertir les protocoles des composants en un protocole standard interne, avec des tampons (*buffers*) d'envois et de réception et un arbitre. Cependant, ces approches ne s'adressent pas à l'intégration des composants logiciels. L'outil TEReCS [Bok 00] synthétise la communication logicielle pour connecter les composants logiciels, étant donnée une spécification de l'architecture de communication et une affectation des composants logiciels aux processeurs de l'architecture. L'approche TEReCS est associée à la synthèse automatique d'un système d'exploitation minimal construit à partir d'un assemblage d'objets réutilisables et configurés selon les exigences de l'application et le matériel correspondant.

2.5.3.10. Dérivation de composants

L'intégration des composants de type « *hard* » ne peut pas avoir lieu par adaptation de leur comportement interne et/ou leur structure. Ces composants sont moins flexibles et par conséquent moins réutilisables que les autres types de composants. Plusieurs approches, qui ont pour but d'améliorer la réutilisation, sont basées sur les composants adaptables. Bien que l'on puisse penser à une configuration très simple du composant (par exemple par une sélection du largeur du bit « *bit width* »), un degré plus élevé de réutilisation peut être atteint par les composants ayant des comportements qui peuvent être modifiés librement. L'approche orientée objet est un moyen pour la modélisation de haut niveau et l'adaptation des composants réutilisables [Bar 99] [Sch 99]. Cette approche, qui peut être mieux classée sous le terme dérivation de composants, est adéquate non seulement pour les composants matériels de type « *soft* » et « *firm* », mais aussi pour les composants logiciels [Ram 00]. Bien que la réutilisation de composants soit améliorée par cette approche, l'intégrateur système fournit un effort de conception très important, et la performance des composants est plus difficile à prévoir.

La dérivation de composants et la synthèse de la communication sont des approches différentes pour la résolution d'un même problème : l'intégration des composants, qui ne suivent pas un standard commun. La dérivation de composants est une solution basée souvent sur les concepts orientés objet provenant de la communauté de l'ingénierie du logiciel. Elle peut être appliquée pour l'intégration des composants logiciels de l'application et les composants matériels de type « *soft* » et « *firm* », mais elle ne peut être utilisée pour l'intégration des composants de type « *hard* ». D'autre part, la synthèse de la communication suit le chemin de la communauté du matériel en ce qui concerne l'automatisation de la synthèse logique et de haut niveau. Cette solution est la seule pour pouvoir intégrer des composants de type « *hard* » hétérogènes, bien qu'elle peut être aussi utilisée pour l'intégration des composants logiciels et des composants matériels de type « *soft* » et « *firm* ». La dérivation de composants est généralement un processus manuel, tandis que la synthèse de la communication est un processus qui peut être automatisé.

2.5.4. Etat de l'art sur la réutilisation de composants

Pour créer des composants IP destinés à la réutilisation, un effort de conception additionnel est nécessaire. Cet effort consiste à définir des styles et des règles de conception, codification, paramétrisation et classification. Dans le domaine logiciel, UML [Zia 02] est un standard pour la modélisation et l'intégration des composants logiciels. La génération de code

et d'autres techniques sont utilisées pour l'intégration de composants homogènes. Bien que la modélisation des composants hétérogènes soit possible avec UML, elle est surtout orientée vers le développement du logiciel. Pour l'intégration de composants matériels, le manuel RMM (*Reuse Management Manual*) de Mentor et Synopsys [Kea 02] détaille les règles de codification pour la synthèse RTL. RMS (*Reuse Management System*-Tübingen) [RMS] propose une stratégie complète de conception pour la réutilisation (description, documentation, testbench, scripts, contrôle de versions, historique d'accès, classification de composants IP) avec des outils associés. Des approches utilisant un langage unique pour la création de composants existent tels que SystemC [Sys 00] et Superlog [Slog]. Rosetta [Ale 01] propose un standard pour la définition, la composition, et l'interaction entre des modèles hétérogènes faisant référence à des domaines spécifiques. D'autres solutions sont basées sur une approche multi-langage. Des efforts de vérification du style de conception sont fournis pour qualifier les composants destinés à la réutilisation, tels que OpenMore [Ope 00] de Mentor et Synopsys, et la méthodologie de qualification FZI [Vor 01]. En ce qui concerne la classification et la recherche de composants IP, Design&Reuse [Des 97] propose un portail Web basé sur l'encapsulation des composants en utilisant la technologie XML. Ce portail permet l'échange d'information entre les différents catalogues de composants des producteurs et le catalogue intranet d'un centre de conception. Synchronicity propose son IP Gear Publisher Suite [GEAR] qui représente un ensemble d'outils pour la construction et l'accès à des catalogues Web à partir d'une base de données de composants. Pour le transfert de composants, l'organisation VSIA propose VCT (*Virtual Component Transfert*) [VCT] qui offre des règles, des formats de classification et de recherche pour catalogues, ainsi que des formats de transfert. L'évaluation de composants se trouvant dans des endroits géographiquement distants se fait par une simulation via le Web. JavaCAD [Dal 99] utilise Java comme un langage de modélisation et propose une approche client-serveur avec un accès protégé aux estimateurs de surface et de puissance fournis par le producteur. Sonics propose SOCworks [SOC], un site Web interactif pour la construction/l'évaluation d'un SoC basé sur son µNetwork, des composants IP de plusieurs producteurs sont aussi disponibles. DISCOE [Wil 99] propose un « *backplane* » de simulation hétérogène distribuée basée sur CORBA. Les différents simulateurs sont adaptés par conversion vers un format standard propriétaire. L'université de Verona [Fin 00] propose une méthodologie qui permet une simulation distribuée de modèles VHDL/Verilog en utilisant l'approche client-serveur. Pour l'intégration de composants, CoreConnect [Cor 99] de IBM suit l'approche de conception basée sur un bus standard et propose une bibliothèque de composants et un environnement de développement

et de simulation. VSIA propose VCI (*Virtual Component Interface*) et SLIF (*System-Level InterFace*) [Vah 95] [Vah 98] : le premier offre une interface standard pour les composants et des adaptateurs pour connecter ces interfaces directement à des structures de communication. Le deuxième est basé sur des transactions et des classes de messages pour la description de couches d'abstraction d'interfaces du niveau système au niveau RTL. OCP-IP (*Open Core Protocol*) [Ocp 03] propose un protocole indépendant du bus.

D'autres travaux existent pour intégrer les composants à travers une synthèse de la communication logiciel/matériel. Dans l'environnement IPChinook [Cho 99] les protocoles de communication abstraits sont synthétisés en des protocoles de bus de bas niveau selon une architecture cible. L'environnement IPChinook génère aussi un ordonnanceur (*scheduler*) spécifique à un partitionnement donné entre les processus et les processeurs. Paradise [PARA] de C-Lab permet la construction d'un *RTOS* dédié à partir d'une bibliothèque de composants de base, et la synthèse de communication basée sur des composants et selon une architecture de communication choisie. Les solutions les plus récentes manipulent uniformément les interfaces logicielles et matérielles entre les composants. Dans l'approche COSY [Bru 00], le système fonctionne par une séparation explicite entre les blocs fonctionnels et l'architecture. Ensuite, les blocs fonctionnels sont affectés aux composants architecturaux. Les interactions entre les blocs fonctionnels sont représentés par des transactions de haut niveau et sont par la suite affectés à des schémas de communication entre les parties logicielles et les parties matérielles. Une bibliothèque fournit un ensemble fixe d'adaptateurs de composants et contient des implémentations en logiciel et/ou en matériel pour des schémas de communication donnés.

2.6. Réutilisation d'outils : problème d'interopérabilité

La complexité des systèmes hétérogènes embarqués ne peut être maîtrisée sans le support d'outils d'aide à la conception. L'industrie souffre de ne pas avoir un flot de conception complet qui peut implémenter une méthodologie de conception partant du niveau système jusqu'au silicium. Par conséquent, plusieurs outils académiques et industriels ont été développés. Le processus de conception des systèmes embarqués doit impliquer un grand nombre de ces outils (les outils les plus performants qui existent pour chaque étape de conception) afin d'accomplir les différentes étapes de conception. La difficulté principale réside dans le fait que les outils impliqués dans le même processus de conception doivent coopérer et échanger des modèles et des données différents.

Pour pouvoir bénéficier des nouveaux outils de conception qui apparaissent sur le marché et de la compétence des outils et des flots de conception existants, les concepteurs des systèmes doivent compter sur un environnement de conception ouvert. Un environnement de conception est ouvert quand il offre des mécanismes appropriés pour une introduction facile des nouveaux outils. Ces derniers doivent coopérer avec les outils déjà existants pour compléter ou étendre un flot de conception donné. Les outils sont dit inter-opérables si et seulement s'ils coopèrent de façon transparente dans un flot de conception donné. C'est-à-dire qu'ils échangent directement des données ou n'importe quelle information de conception nécessaire, en utilisant des mécanismes de communication appropriés (fichiers, protocoles), sans avoir besoin d'intervention manuelle pour convertir des protocoles ou des données complexes. L'intégration d'outils est le processus avec lequel un outil externe est introduit dans un environnement de conception.

Il existe plusieurs mécanismes qui peuvent être offerts par un environnement de conception pour pouvoir intégrer des outils et assurer une interopérabilité entre eux. Du point de vue de l'utilisateur, l'interopérabilité entre les outils est obtenue au moyen des langages de modélisation courants ou des modèles de calcul. Du point de vue des outils internes d'un flot de conception, des modèles de représentation de données peuvent être utilisés en interne. Enfin, du point de vue du constructeur ou d'intégrateur d'outils, plusieurs types de structures de données et de mécanismes d'accès peuvent être adoptés : des structures de données en mémoire, des fichiers, des bibliothèques et des bases de données. Ces approches et mécanismes seront présentés et comparés plus en détail dans les prochaines sections. Les standards sont vraiment nécessaires pour assurer une interopérabilité entre plusieurs vendeurs d'outils.

2.6.1. Types d'intégration d'outils

L'intégration d'outils dans un environnement de conception est un problème similaire au problème d'intégration de composants dans un système. Supposons qu'une architecture d'un système embarqué soit développée autour d'un bus standard donné. Un composant peut être directement intégré dans ce système s'il est compatible avec le standard. Sinon, il doit être intégré avec un adaptateur spécial ou il doit subir une dérivation (cf. section 2.5.3). Cette dernière alternative est possible seulement pour un composant ayant une description connue. Il est de même pour les outils. L'intégration peut se faire facilement autour d'un environnement standard. Pour les outils on parlera d'une base de données ou environnement de développement au lieu des bus. Enfin, puisque les outils sont introduits dans un

environnement de conception pour étendre un flot de conception donné, il est clair que l'intégration d'outil et l'organisation du flot de conception sont deux problèmes étroitement liés. Par conséquent, quelques approches qui visent l'organisation du flot de conception SoC offrent aussi une solution pour le problème d'intégration d'outils. Selon le degré d'exposition des outils à l'environnement de conception, deux cas peuvent se présenter : les outils sont soit intégrés, soit encapsulés. Le premier cas est connu sous le terme intégration de type « *white-box* » et le deuxième cas est connu sous le terme intégration de type « *black-box* » :

- une intégration de type « *white-box* » est possible quand le code source de l'outil est conforme à une solution d'interopérabilité entre les outils offerte par un environnement de conception cible. Dans ce cas, l'interopérabilité est directement accomplie. Dans l'exemple d'un environnement de conception offrant un accès à des données partagées à travers une couche API, l'intégration de type « *white-box* » se produit quand l'outil a été développé selon les formats de données de conception et les définitions de services qui sont disponibles à travers l'API ;
- l'intégration de type « *black-box* », à son tour, se produit quand le code source de l'outil n'est pas conforme à la solution d'interopérabilité offerte par l'environnement de conception et l'outil ne peut pas être directement connecté à l'environnement. Ainsi, l'intégration d'outil nécessite le développement d'un adaptateur spécial entre l'outil et l'environnement de conception. Dans le cas de l'API, l'adaptateur d'un outil de simulation, qui doit obtenir ses données en entrée à travers cette API, devrait correspondre à un convertisseur de format de données et doit être exécuté avant l'invocation de la simulation.

Les deux alternatives d'intégration d'outils nécessitent un effort de développement, mais la première n'est possible que si le code source de l'outil est disponible.

2.6.2. Approches pour l'échange de données et l'interopérabilité entre les outils

Le processus de conception peut nécessiter plusieurs outils fournissant/nécessitant différents formats et modèles. Le problème est comment faciliter l'échange des données et des modèles entre les différents outils. Ceci nécessite un mécanisme d'échange de données qui doit vérifier certaines conditions :

- être suffisamment flexible pour maintenir les sémantiques entre les outils ;
- préserver les décisions fondamentales dont dépend la performance du système ;

- être neutre, c'est-à-dire indépendant des outils, des méthodes et des processus de conception

Plusieurs approches s'intéressant au problème d'échange de données et d'interopérabilité entre les outils existent, et les plus importantes sont décrites dans les sous-sections suivantes.

2.6.2.1. Traduction directe entre les outils

C'est une solution qui utilise des « traducteurs de fichiers » qui effectuent un processus de traduction directe entre les formats de fichiers des différents outils (Figure 5). Les traducteurs de fichiers sont des programmes spécialisés. Leurs tâches consistent à lire le modèle généré par un outil et à convertir son contenu en un autre modèle valide afin d'être consommé par un autre outil. Malheureusement cette approche possède quelques inconvénients :

- le constructeur des traducteurs de fichiers doit être extrêmement compétent à la fois en formats de fichiers, la sémantique de chaque outil, et en langages de programmation, pour être capable d'écrire les traducteurs ;
- l'écriture d'un traducteur est un processus très long ;
- le nombre de traducteurs nécessaires pour intégrer *n* outils est proportionnel au carré du nombre d'outils. Autrement dit, à chaque fois qu'un outil est ajouté à un ensemble de *n* outils existants, il faut construire *n* nouveaux traducteurs. Cet argument admet, bien sur, la nécessité d'interopérabilité totale entre les outils.

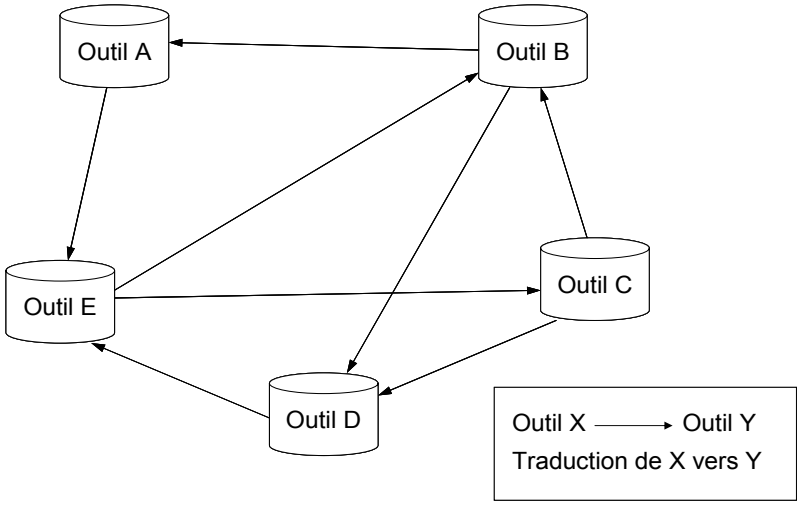

Figure 5. Traduction directe entre les outils

2.6.2.2. Générateur de traducteur

Comme le montre la Figure 6, cette technique consiste à utiliser un générateur de traducteurs pour construire des traducteurs entre les outils de conception. Ce générateur prend en entrée les syntaxes des données de chaque outil et génère un traducteur. Ce dernier permet d'extraire les informations de conception, en utilisant une API, à partir du modèle en sortie de l'outil A, pour produire le modèle d'entrée de l'outil destinataire B.

Le coût de cette méthode est réduit, elle permet une intégration facile et une extensibilité limitée par les traducteurs. L'effort fourni pour la maintenance est raisonnable puisque l'API ne change pas fréquemment. Parmi les inconvénients de cette technique on peut citer le fait qu'elle est limitée à des relations sémantiques directes ayant des syntaxes multiples, nécessite une compréhension de la grammaire syntaxique des données à traduire, elle fait confiance aux outils externes et nécessite une duplication d'effort pour chaque outil.

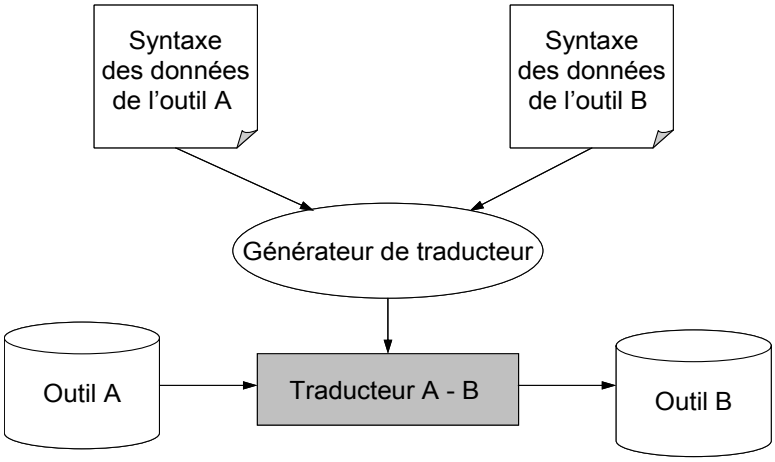

Figure 6. Générateur de traducteur entre les outils

2.6.2.3. Utilisation d'un standard d'échange : base de données ou format de référence

Dans ce cas, les outils de conception s'appuient sur un standard d'échange tel qu'une base de données ou un modèle de référence (méta-modèle) comme le montre la Figure 7. Ce standard d'échange doit être neutre, c'est-à-dire indépendant des outils qui participent aux échanges de données. Des traducteurs doivent être construits entre chaque outil et le standard d'échange et les échanges de données se font par des opérations de type import/export à partir et vers ce standard. Cette solution est efficace parce que le nombre de traducteurs est linéairement proportionnel au nombre d'outils.

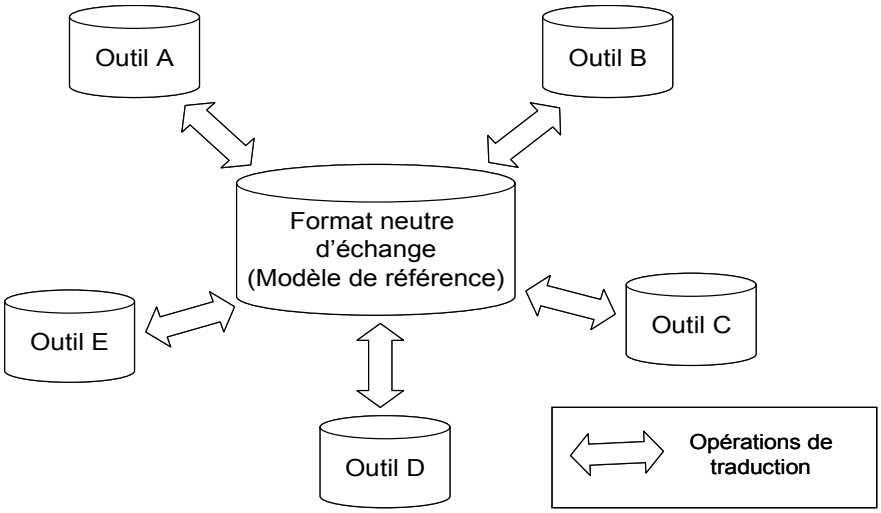

Figure 7. Utilisation d'un standard d'échange

MoML [Lee 00] proposé par le projet Ptolemy, permet de composer plusieurs domaines discrets et analogiques en utilisant une sémantique abstraite commune qui permet l'interaction entre les différents domaines. Le SRMIF (*Semantic Reference Model Intermediate Format*) [SRMIF] d'Accellera, propose une référence sémantique commune pour plusieurs modèles de calcul et une implémentation est possible en plusieurs langages de modélisation. OpenAccess [OACC] est une base de données dédiée à la conception physique, fournie avec une couche API en C++. Cette base de données offre des modèles de données complexes orientés objets et des mécanismes « *callback* » pour la communication entre outils.

2.6.2.4. Utilisation du Web

L'utilisation du Web (Figure 8), en tant que plateforme pour construire des environnements de conception distribués, lance une forme spéciale d'interopérabilité entre les outils. Dans ce cas, l'intégration d'outils se fait à travers des traducteurs. Ces derniers utilisent des API pour accéder aux différents modèles produits par les outils de conception sources, et à les connecter à un « *backplane* » permettant la construction d'environnements distribués (tel que CORBA). Les modèles produits par les traducteurs sont véhiculés à travers le Web vers leurs outils destinataires.

Des solutions particulières pour l'intégration d'outils sont associées au Web, et l'environnement de conception résultant présente des qualités qui sont plus difficiles à obtenir au niveau des environnements conventionnels. En particulier, la coopération entre les concepteurs se trouvant dans des sites géographiquement distants est assurée, améliorant de cette manière la réutilisation des compétences. La protection des composants, que se soit sous forme d'outils ou de données, est aussi possible. DCB [Mel 02], offre un environnement de

simulation via le Web basé sur le standard HLA (*High-Level Architecture*) et une génération semi-automatique d'adaptateurs pour des simulateurs quelconques.

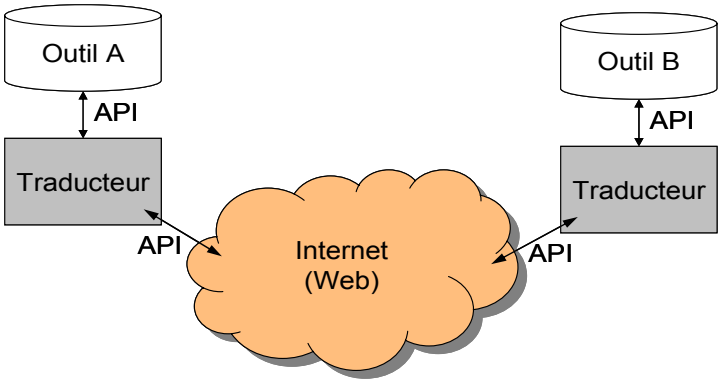

Figure 8. Intégration d'outils via le Web

2.6.3. Les outils de conception existants

Cette section présente trois types d'outils : des outils qui essayent de couvrir un flot de conception complet, des outils dédiés à l'exploration d'architectures et d'autres pour la conception d'architectures. On s'intéressera uniquement aux outils de raffinement jusqu'au niveau RTL. Ainsi, il existe plusieurs autres types d'outils nécessaires à la conception d'un système embarqué. On peut citer les outils de validation et les outils de test en plus des outils de conception physiques.

2.6.3.1. Les outils couvrants le flot de conception complet

Un outil couvrant le flot de conception complet n'existe toujours pas. Cependant, les outils qui essayent de couvrir un flot de conception complet sont basés sur des modèles de SoC très simples et très restreints. Parmi ces outils, le plus connu est COSYMA [Ern 93] : c'est un environnement de conception logiciel/matériel permettant la génération d'architectures spécifiques SoC se composant d'un seul processeur RISC et plusieurs accélérateurs matériels. Les différents composants de cette architecture communiquent par le moyen d'une ou plusieurs mémoires partagées. L'entrée de cet outil est un graphe de contrôle et de flot de données (*Control & Data Flow Graph* – CDFG) qui représente les différentes tâches du système à concevoir. Le comportement de chaque tâche est décrit en langage C. L'outil permet aussi d'évaluer les performances de l'architecture générée. COSYMA fournit en sortie le code des tâches à exécuter pour le processeur RISC et le code VHDL décrivant le comportement des différents accélérateurs au niveau transfert de registres (RTL).

2.6.3.2. Les outils d'exploration d'architectures

Parmi les outils d'exploration d'architectures, on peut citer :

- Archimate : est le fruit des travaux de recherche présentés dans [Val 01] [Ism 96]. C'est un outil d'exploration d'architectures qui permet de générer un modèle de l'architecture matérielle au niveau RTL ainsi que le code logiciel en langage C qui sera exécuté par un ou plusieurs processeurs spécifiques. Cet outil prend en entrée une description structurelle de l'application représentée sous forme de plusieurs modules interconnectés et s'exécutant en concurrence. Ces modules peuvent être décrits à des niveaux d'abstraction différents et en utilisant des langages divers tels que C, Matlab, HDL, SDL [Hes 00]. Archimate fournit en sortie :
  - une description structurelle de l'architecture matérielle raffinée ;
  - des connexions et des protocoles « point à point » implémentent les canaux de communication ;
  - une implémentation logicielle de parties de l'application en hiérarchie de machines d'états entrelaçant le comportement de plusieurs tâches (machine d'états feuilles). Ces machines d'états sont codées en langage C ;
  - des modèles RTL codés en VHDL pour les implémentations matérielles de sous-systèmes de l'application ;
  - des modèles RTL codés en VHDL et du code C pour les interfaces logiciel/matériel.

  Le partitionnement des blocs fonctionnels de l'application est implicite. Une affectation bijective est établie entre chaque groupe de fonctionnalité et une unique ressource de calcul (processeur logiciel ou processeur matériel). Malheureusement, les implémentations de la communication restent trop simplistes pour autoriser l'utilisation de cet outil pour des raisons autres que l'exploration d'architectures. En effet, on regrette l'absence d'adaptation logicielle de la communication, une topologie de communication et d'autres solutions architecturales figées ;
- VCC*: Virtual Component Co-design* de Cadence est un outil de conception qui s'intéresse à l'exploration d'architectures [Cad 01]. Ceci est accompli par une exploration des partitionnements logiciel/matériel pour atteindre une performance optimale du système en tenant compte des contraintes données. Quand un partitionnement convenable est trouvé, des modèles de composants compatibles avec le modèle d'exploration d'architectures sont recherchés dans des bibliothèques mises à la disposition du concepteur par les vendeurs de ces composants. La méthodologie de

VCC (Figure 9), commence par une spécification fonctionnelle et une simulation du système décrit sous forme de blocs comportementaux interconnectés par des connexions « point à point », sans l'implication d'une éventuelle implémentation de l'architecture. Ensuite, une plateforme architecturale composée de plusieurs composants IP (CPU, IP, mémoires, bus, etc.), est capturée. En plus des composants qui correspondent aux IP effectifs dans le système sur puce final, un modèle de système d'exploitation temps réel (*RTOS*) ou un ordonnanceur (*scheduler*) est attaché à chaque processeur. Ainsi l'architecture obtenue est appelée architecture relaxée, puisque seulement la topologie de base est définie, sans aucune information détaillée sur les ports et les signaux. Les composants comportementaux sont par la suite affectés aux composants architecturaux, définissant de cette façon un partitionnement logiciel/matériel. Un schéma de communication (c*ommunication pattern*) est affecté à chaque fil de connexion dans le diagramme comportemental. Ce schéma explicite la façon avec laquelle la communication entre les blocs comportementaux est implémentée.

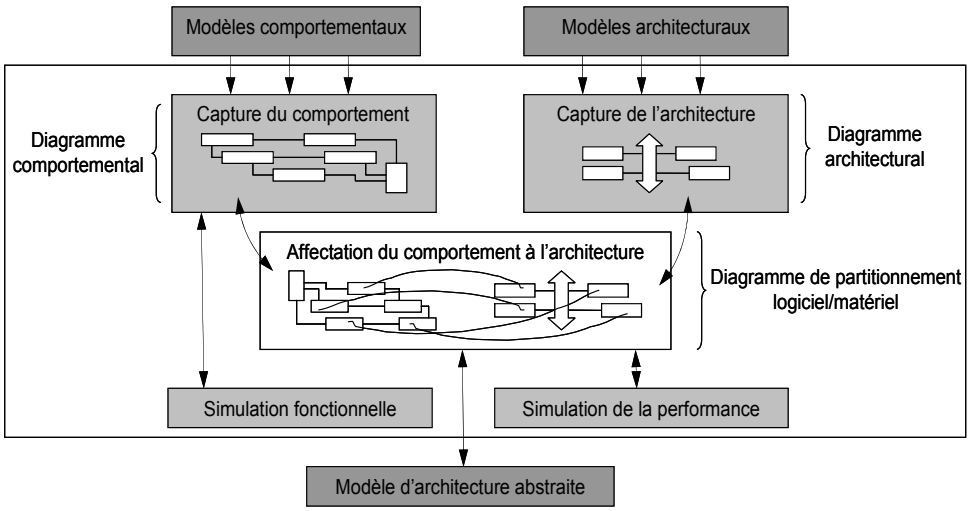

Figure 9. Méthodologie de conception de VCC

Dans le but de simuler la performance du système, chaque composant architectural possède un modèle de performance décrit en terme de services fournis par le composant, tel qu'un ordonnanceur pour un système d'exploitation, et les requêtes d'interruptions et de communication avec un bus pour un processeur, et des transactions de type lire et écrire (*read/write*) pour une mémoire. Malheureusement, cet outil ne fournit aucun modèle dédié à l'implémentation ;

- Artemis : [Pim 01] est un environnement de modélisation et de simulation qui vise l'exploration de l'espace de conception d'architectures à des niveaux d'abstraction multiples.

2.6.3.3. Les outils de conception d'architectures

Parmi les outils de conception d'architectures, on peut citer :

- CoWare Napkin-to-Chip (N2C): [Ver 98] [Ver 96] [Ver 99] est un environnement de conception et de simulation des systèmes hétérogènes. Comme l'indique la Figure 10, le flot de génération d'architectures de CoWare prend en entrée une architecture abstraite, une bibliothèque de composants (processeurs, mémoires, protocoles de communication) et une bibliothèque d'adaptateurs spécifiques aux scénarios (schémas) de communication. L'architecture cible de ce flot est fixe. Elle est composée d'un seul processeur connecté à plusieurs composants matériels via un réseau de communication.

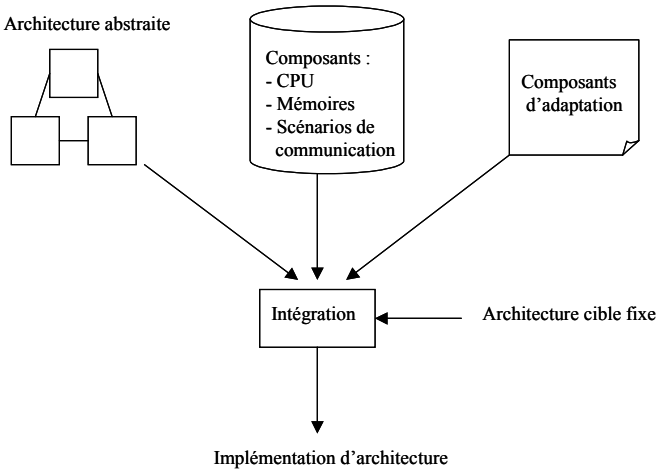

Figure 10. Flot de génération d'architectures de Coware

L'architecture abstraite est décrite en langage de spécification logiciel/matériel CoWareC. Elle spécifie l'application sous forme d'un réseau de modules communicants. La sémantique de la communication entre les tâches est basée sur des appels de procédures à distance (*Remote Procedure Call* – RPC). La communication entre les modules est assurée par des canaux de communication reliant les ports des modules. L'implémentation de cette architecture abstraite consiste à :

- transposer les modules sur un modèle de processeur et des modèles de composants matériels déjà existants dans la bibliothèque ;

- synthétiser la communication entre les différents modules ;
- générer les adaptateurs de communication.

La transposition (ou encore affectation) consiste à affecter chaque module de la spécification abstraite soit à un composant logiciel (processeur logiciel), soit à un composant matériel (accélérateur, contrôleur, périphérique) de l'architecture cible. La synthèse de communication consiste à raffiner les canaux de communication abstraits et à choisir un scénario de communication pour chaque canal. La bibliothèque des protocoles de communication contient plusieurs scénarios restreints au protocole poignée de main (*handshake*). CoWare propose une approche basée sur la connexion des composants de l'architecture au réseau de communication à travers des adaptateurs de communication. Ces adaptateurs sont générés automatiquement en sélectionnant dans une bibliothèque les bons éléments à partir des caractéristiques du réseau et du composant à connecter ;

- O'Nils et ProGram : [Nil 99] [Nil 99bis] présentent une méthodologie de conception d'architecture matérielle à base de processeurs communicants par des protocoles différents. Ces travaux visent la spécification et la génération d'adaptateurs de communication, mais cette génération n'est pas globalement automatisée. Cependant un environnement de co-synthèse et de prototypage est proposé : [Tam 96]. Dans cet environnement, la génération d'adaptateurs matériels de composants est adressée par ProGram [Obe 99]. Conscient que les approches « *interface-based* » (telles que [Raw 97]) sont limitées par la disponibilité de bibliothèques suffisamment fournies pour adapter toute la diversité de protocoles existants, ProGram présente une nouvelle approche pour la spécification d'interface matérielle de communication, basée sur la modélisation du comportement de l'interface par une grammaire régulière. Un outil est alors utilisé pour générer un automate de reconnaissance des lexèmes, conceptuellement très proche des outils d'aide à la conception d'analyseur syntaxique pour les compilateurs tels que *GNU Yacc*. Il en résulte un modèle VHDL RTL, implémentant par des machines d'états finis, les automates de reconnaissance. La contribution principale de ces travaux est une méthodologie et un outil de conception d'adaptateur basé sur une modélisation « grammaticale » du comportement de l'adaptateur, certes plus concise que par l'utilisation d'un HDL, mais encore manuelle ;
- GAUT *:* [Bag 97] est un outil de synthèse comportementale. Il permet aussi la génération des interfaces matérielles appelées Unités de Communication (*UCOM*). Le

réseau d'interconnexion mis en œuvre est une nappe de connexions « point à point » reliant un unique processeur à plusieurs accélérateurs matériels. Il est basé sur une analyse formelle des communications de l'application et sur leur affectation à des protocoles faisant partie des spécifications. Il permet de générer des modèles VHDL RTL synthétisables tant pour les accélérateurs que pour les unités d'adaptation. Gaut est orienté architecture monoprocesseur. L'utilisation de cet outil est envisageable dans le cadre d'architectures multiprocesseur, si son analyse globale des communications peut supporter la complexité des réseaux embarqués. De plus l'utilisation de connexions « point à point » rend coûteuse la mise à l'échelle du modèle architecturale et donc son utilisation pour des systèmes multiprocesseurs fortement communicants.

- IP Chinook : Chinook propose une méthode systématique de génération d'interfaces. Cette méthode consiste à générer une logique séquentielle qui permet d'adapter un port d'entrée/sortie d'un microcontrôleur à un périphérique. Cette méthode exige en entrée :
  - une description du comportement du système : c'est un programme écrit par un utilisateur en utilisant un langage de haut niveau. Cette description est composée d'une partie déclarative qui alloue les éléments de mémorisation statiques et une partie opérationnelle qui inclut les fonctions de communication avec le périphérique via des appels de pilotes. Pour éviter les problèmes de synchronisation, cette spécification est supposée mono-processus ;
  - une description du périphérique : c'est une classe à trois attributs. Elle est composée d'une liste des ports du périphérique, une représentation matérielle du périphérique sous forme de règles statiques associées à chaque broche et une représentation logicielle sous forme de séquences de comportement ;
  - une description du microprocesseur : elle contient une liste de ports du microprocesseur avec des informations sur la direction et le type de données supportées par chaque port. Chaque port peut avoir des instructions de communication multiples. Une instruction de communication représente la sémantique d'accès de chaque port par le microprocesseur.

  La sortie de cette méthode est un réseau de modules matériels synthétisables connectés au microprocesseur via des parties logiques ou des adaptateurs. Ces adaptateurs interfacent chaque port de périphérique au port du microcontrôleur. Ils sont sélectionnés à partir d'une bibliothèque matérielle tout en respectant les paramètres de

la spécification d'entrée. Bien que cette approche puisse être étendue à l'assemblage de plusieurs IP, elle se limite à un seul processeur et elle suppose que la spécification de l'application d'entrée ne contient pas de tâches ou de processus concurrents.

- COSY : Dans le cadre du projet *ESPRIT/OMI-COSY* qui résulte d'une collaboration entre l'université de Pierre-Marie Curie de Paris et les laboratoires de Philips, un processus de raffinement de la communication logiciel/matériel est proposé en partant d'un modèle de réseau de processus de Khan étendu pour la spécification des systèmes. Le standard d'interfaces VCI est utilisé pour la conception des adaptateurs matériels génériques [Bru 00].

D'autres outils existent et offrent des standardisations qui se basent sur un bus système standard auquel plusieurs types de composants peuvent être connectés. On peut citer à titre d'exemple :

- Peripheral Interconnect Bus d'OMI [PIBus] ;
- AMBA d'ARM Inc [Arm 99] ;
- CoreConnect d'IBM [Ibm 02] ;
- OpenCore de Sonics [Ocor].

Malheureusement, ces efforts de standardisation ne répondent pas toujours aux exigences de l'application en terme de protocoles de communication. En plus, les interfaces de communication sont parfois trop générales et utilisent des protocoles qui ne sont pas réclamés par l'application.

2.7. Résumé

L'étude présentée dans la section 2.6.3 nous a permis de constater que :

- les outils qui essayent de couvrir un flot de conception complet, utilisent des modèles de SoC très simples et très limités ;
- un grand nombre d'outils existe, cependant ces outils sont spécifiques à des domaines d'application différents. En plus, ces outils permettent d'accomplir avec efficacité seulement des étapes de conception bien particulières telles que l'exploration d'architectures, la conception d'architectures, la simulation, etc.

Le problème qui se pose est comment intégrer ce grand nombre d'outils, provenant de sources différentes et ayant des domaines d'application différents, afin de bénéficier de leur spécificité et de leur performance et de construire des flots de conception complet.

L'étude présentée dans la section 2.6.2 nous a permis de constater l'existence de plusieurs mécanismes pour l'échange de données de conception entre les outils. L'utilisation

d'un format de référence ou encore un standard d'échange est une solution qui possède plusieurs avantages. Dans les travaux de cette thèse nous proposons l'utilisation d'un format de représentation des systèmes hétérogènes embarqués, appelé COLIF [Ces 01]. Ce format est considéré comme un modèle de référence ouvert à l'intégration d'outils de conception et de composants logiciel/matériel. COLIF est présenté dans la section 3.5 du chapitre suivant.

2.8. Conclusion

La conception des systèmes électroniques nécessite, d'une part, l'intégration de plusieurs composants hétérogènes sur une puce, et d'autre part, l'interopérabilité entre plusieurs outils de conception provenant d'origines diverses qui doivent collaborer au sein d'un même flot de conception.

Ce chapitre a présenté, dans une première partie, les concepts de base de la réutilisation de composants logiciel/matériel. Dans une deuxième partie, ce chapitre a présenté les concepts de base de la réutilisation d'outils de conception. Les deux parties ont mis l'accent particulièrement sur l'aspect intégration : l'intégration de composants logiciel/matériel dans un système donné et l'intégration d'outils dans un environnement de conception donné.

Une analyse des solutions proposées actuellement pour la réutilisation et plus particulièrement l'intégration de composants et d'outils a été présentée. Cette analyse aboutit à la conclusion qu'un environnement ouvert et flexible est une solution nécessaire pour pouvoir intégrer facilement des composants hétérogènes et des outils provenant de sources différentes et ayant des domaines d'application spécifiques. Le reste de ce document propose l'utilisation d'un format intermédiaire dédié à la représentation des systèmes hétérogènes embarqués, appelé COLIF, comme un modèle de référence ouvert à l'intégration des composants logiciel/matériel et à l'intégration d'outils de conception.

Chapitre 3. Le flot ROSES pour la conception et la validation des systèmes hétérogènes embarqués

Sommaire

3.1. Introduction

Ce chapitre présente le flot de conception ROSES qui implémente une méthodologie de conception avec un ensemble d'outils correspondant. ROSES vise l'intégration des composants logiciel/matériel pour construire des systèmes hétérogènes embarqués. Cette intégration se fait à travers la génération des sous-systèmes d'interface complexes (adaptateurs sophistiqués) pour adapter les différents composants au réseau de communication. La sortie du flot est un modèle architectural pour la synthèse (Figure 11) ou la simulation. Ce flot de conception prend comme entrée un modèle architectural abstrait.

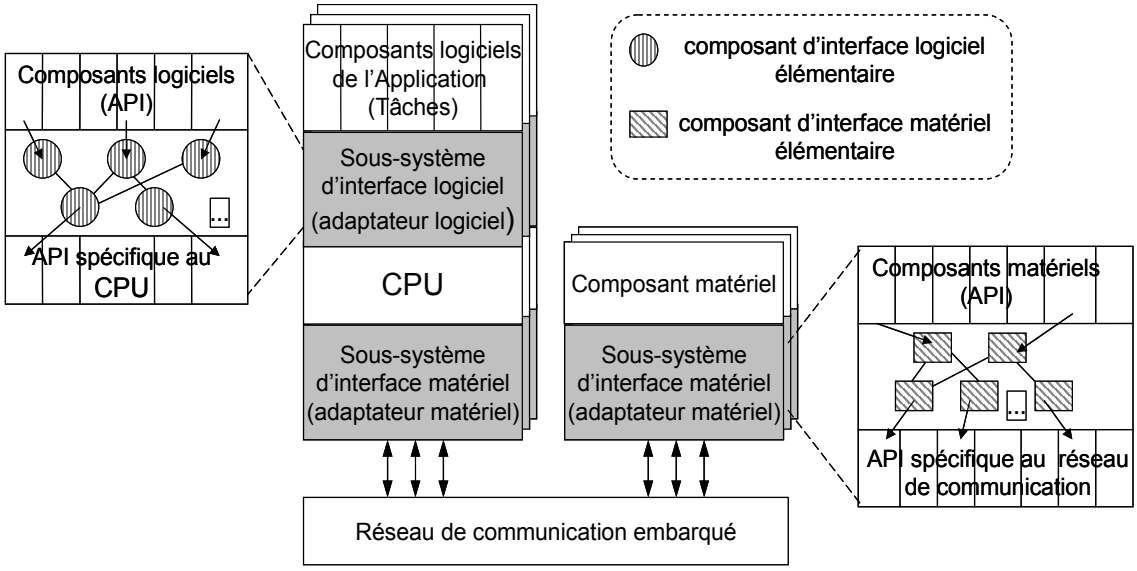

Figure 11. Modèle architectural des SoC

La génération des adaptateurs logiciels consiste à générer des systèmes d'exploitation spécifiques à l'application et aux processeurs. La couche des adaptateurs logiciels offre une API de haut niveau, qui permet d'isoler l'application logicielle du reste du système. La génération des adaptateurs matériels permet l'intégration automatique des composants matériels, et plus précisément les composants ayant un comportement et/ou une structure inconnus. Cependant, l'approche est aussi applicable pour l'intégration des composants, qui suivent un standard d'interface donnée. La méthodologie de ROSES a été conçue pour s'adapter à n'importe quelle structure globale de communication, telle qu'un réseau embarqué (*Network-on-Chip* – NoC) ou un bus.

ROSES est basé sur une définition de trois niveaux d'abstraction de la communication qui sont aussi adoptés par d'autres approches courantes : le niveau système (fonctionnel pure), le niveau macro-architecture, et le niveau micro-architecture (RTL). Ces niveaux constituent

des séparations claires entre les étapes de conception, supportant la réutilisation des composants et des outils pour les tâches de conception à chacun de ces niveaux.

Le reste de ce chapitre est organisé de la manière suivante : la section suivante présente les modèles et les composants définis par la méthodologie de ROSES. La troisième section introduit les différents schémas d'intégration des composants logiciels et matériels proposés par ROSES. La quatrième section décrit l'architecture abstraite (ou encore architecture virtuelle) qui représente le point d'entrée du flot ROSES. La cinquième section présente COLIF qui est un format intermédiaire permettant la description de l'architecture virtuelle. La dernière section introduit les différents outils qui constituent ROSES: l'outil de génération d'adaptateurs logiciels, l'outil de génération d'adaptateurs matériels et enfin l'outil de génération des adaptateurs de co-simulation.

3.2. Les modèles et les composants définis par ROSES

3.2.1. Les modèles

La construction des systèmes hétérogènes embarqués par assemblage de composants préexistants nécessite un modèle architectural de haut niveau qui abstrait tout les détails d'implémentation de bas niveau. L'abstraction est la caractéristique fondamentale dans la plupart des techniques de réutilisation [Kru 92]. ROSES suit l'idée que le modèle architectural peut être implémenté par un assemblage de composants logiciel/matériel réutilisables.

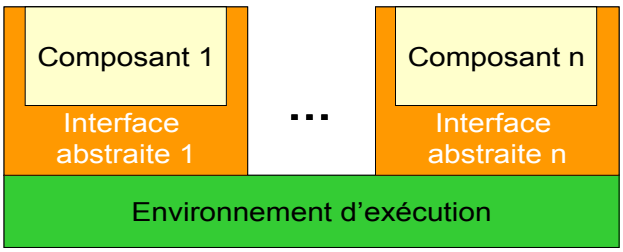

Figure 12. Modèle de composants virtuels

La Figure 12 montre un modèle de composants virtuels (*Virtual Component Model*) qui représente un modèle abstrait du système. Ce modèle est composé de plusieurs composants virtuels et un environnement d'exécution. Un composant virtuel est composé d'un module qui peut être de nature logiciel, matériel ou fonctionnel et une interface abstraite qui inclut plusieurs informations telles que les services requis/fournis, les services de contrôle/synchronisation, des paramètres de conception, etc. L'environnement d'exécution

représente une interconnexion abstraite. Le modèle de composants virtuels permet la manipulation d'objets hétérogènes, la réalisation de schémas d'adaptation différents et très sophistiqués et la génération automatique d'interfaces spécifiques et/ou d'architectures cibles.

Partant du modèle de composants virtuels, un flot de conception des systèmes hétérogènes embarqués produit un modèle architectural au niveau transfert de registres (RTL) qui contient tous les détails au cycle d'horloge prés. Ces détails sont nécessaires pour d'autres étapes de synthèse automatique. Généralement, des adaptateurs logiciel/matériel doivent être générés pour adapter les composants hétérogènes et les interconnecter. Par exemple, les composants logiciels de l'application ont besoin d'être re-compilés sur les processeurs et les systèmes d'exploitation de l'architecture choisie.

3.2.2. Les composants

Dans la méthodologie de ROSES, les composants nécessaires pour la conception et la validation des systèmes hétérogènes embarqués sont classés en trois groupes, (Figure 13) :

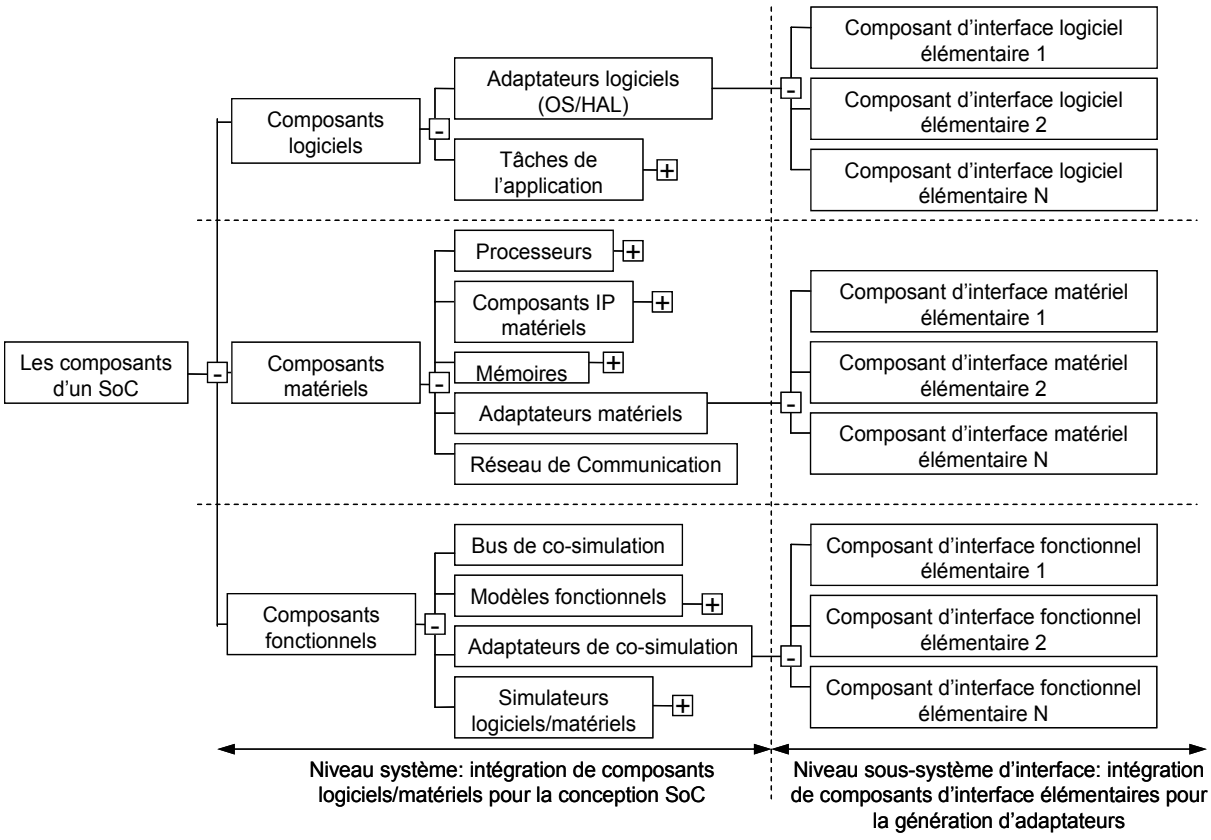

Figure 13. Les composants nécessaires pour la conception des systèmes hétérogènes embarqués

- **les composants logiciels :** ils incluent les tâches de l'application s'exécutant sur un ou plusieurs processeurs cibles en utilisant une API de communication de haut niveau, et

des adaptateurs logiciels qui se composent d'un système d'exploitation (*Operating System* – OS) et d'une couche d'abstraction du matériel (*Hardware Abstraction Layer* – HAL) et qui adaptent les tâches de l'application à la partie matérielle du système. Les adaptateurs logiciels sont construits par un assemblage de composants d'interface logiciels élémentaires. Ces derniers fournissent plusieurs services de haut niveau (par exemple l'ordonnancement des tâches) et plusieurs services de bas niveau (par exemple le changement de contexte) ;

- **les composants matériels :** ils incluent des composants IP, des processeurs, des mémoires, des adaptateurs matériels, et le réseau de communication. Ce dernier est un composant matériel spécial qui implémente la communication globale entre les sous-systèmes de la puce. Les adaptateurs matériels sont construits par un assemblage de composants d'interface matériels élémentaires qui fournissent des services tels que la gestion d'interruption et la conversion de type de données ;
- **les composants fonctionnels :** ils incluent des modèles fonctionnels des interfaces logiciel/matériel, des interfaces de simulateurs, des adaptateurs de co-simulation et un bus de co-simulation. Ce dernier supporte les différentes interactions entre les simulateurs de composants matériels, logiciels et fonctionnels. Les adaptateurs de co-simulation sont utilisés pour adapter les différentes interfaces des simulateurs et des composants au bus de co-simulation.

3.3. Les schémas d'intégration de composants logiciel/matériel

Pour l'intégration des composants logiciels, ROSES utilise un système d'exploitation spécifique à l'application, il contient un ensemble optimal (minimal) de fonctions sollicitées par cette dernière. Initialement, les composants logiciels sont des tâches qui échangent des données entre elles. Ces échanges se font à travers un réseau de communication, dit abstrait, car il abstrait tous les détails de la communication de bas niveau. L'accès à ce réseau est possible en utilisant des fonctions d'accès prédéfinies (API) de haut niveau (Figure 14(a)).

La Figure 14(b) montre la structure en couches de l'adaptation utilisée par ROSES pour l'intégration des composants logiciels. La couche API isole les composants logiciels de l'application de la plateforme matérielle et des autres couches logicielles. Ainsi, l'application logicielle reste complètement indépendante de la plateforme matérielle. La couche service du système d'exploitation contient des services de haut niveau fournis par le système d'exploitation, tels que l'ordonnancement des tâches et des primitives de communication de haut niveau. Enfin, la couche d'abstraction du matériel (HAL) contient tout le logiciel qui

dépend de la plateforme matérielle. Par exemple, le code de démarrage (*boot code*), le code du changement de contexte (*context-switching code*), le code pour la configuration et l'accès aux ressources matérielles (MMU, bus, pont de bus, etc.) appelés pilotes (ou *drivers*). Ces différentes couches logicielles sont construites par un assemblage de composants d'interface logiciels élémentaires et réutilisables disponibles dans une bibliothèque [Gau 01].

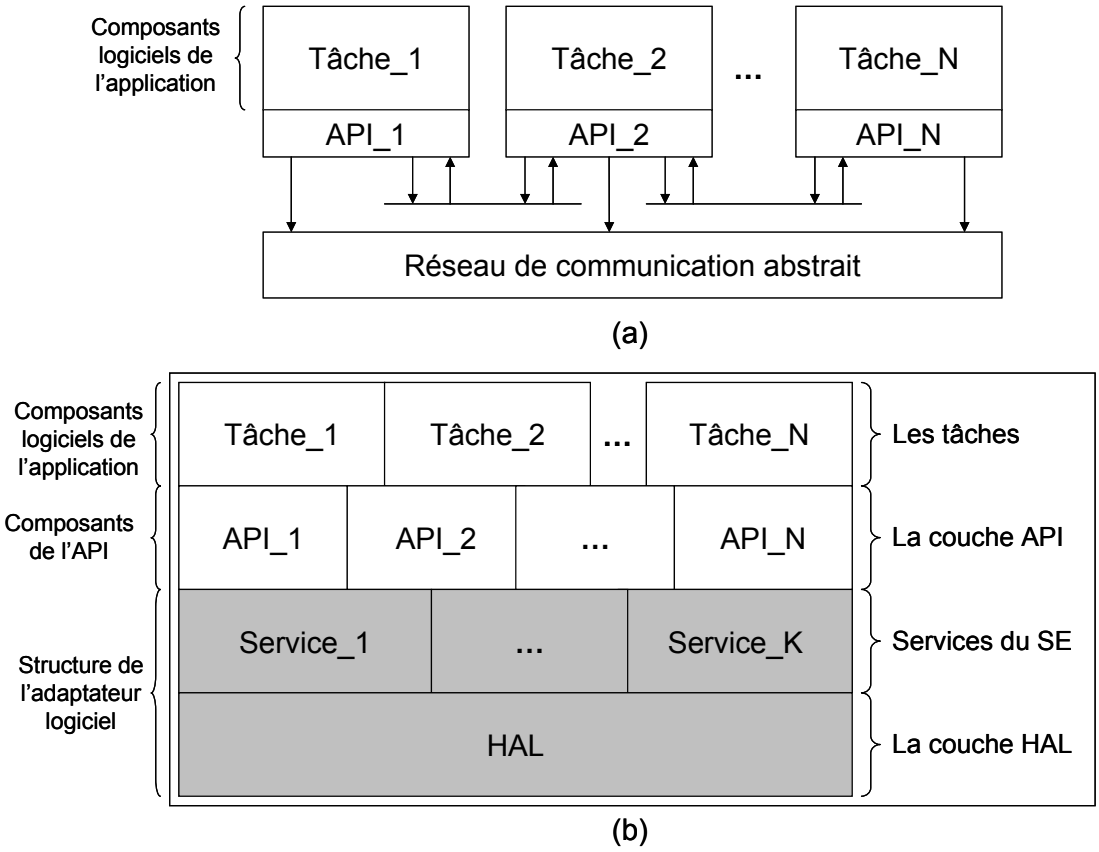

Figure 14. Adaptation logicielle et structure de l'adaptateur logiciel

L'intégration des composants matériels est faite autour d'un réseau de communication générique en utilisant des adaptateurs matériels (Figure 15) qui permettent de connecter chaque processeur/composant IP au réseau de communication embarqué. Ces adaptateurs matériels sont obtenus par un assemblage adéquat de composants d'interface matériels élémentaires et réutilisables [Lyo 01] [Gha 03] : adaptateurs de processeurs/IPs/mémoires et adaptateurs de canaux de communication (*Channel Adapter* – CA).

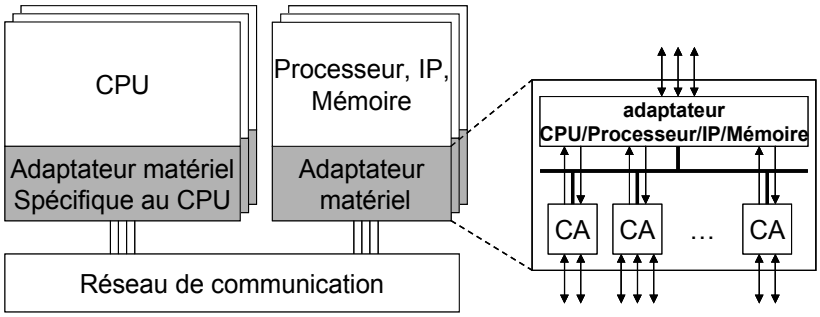

Figure 15. Adaptation matérielle et structure de l'adaptateur matériel

3.4. Architecture virtuelle

ROSES utilise le concept d'architecture virtuelle. Il s'agit d'une architecture abstraite (Figure 16) qui correspond au modèle de composants virtuels présenté dans la section 3.2.1. Cette architecture est composée de modules virtuels, qui correspondent aux nœuds de calcul et mémoires, interconnectés par n'importe quelle structure de communication encapsulée aussi par un module virtuel. Ce modèle d'architecture abstraite sépare clairement l'aspect calcul de l'aspect communication permettant ainsi des chemins d'implémentation indépendants et concurrents pour les composants ainsi que pour la communication. Les modules virtuels qui correspondent aux processeurs peuvent être hiérarchiquement décomposés en sous-modules contentant les tâches logicielles affectées à un processeur donné. Les modules virtuels communiquent à travers des ports virtuels, qui présentent un ensemble de ports internes et externes hiérarchiques à travers lesquels les services sont requis et fournis. La séparation entre les ports internes et externes donne la possibilité d'interconnecter des modules hétérogènes.

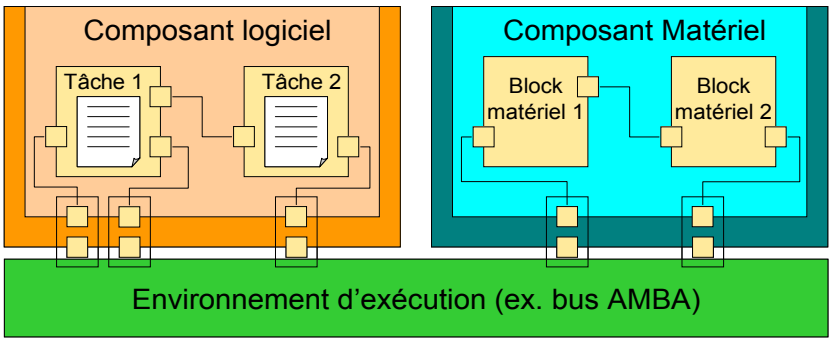

Figure 16. Architecture virtuelle

L'architecture virtuelle est annotée par des paramètres architecturaux qui caractérisent :

- les besoins de l'application logicielle (type de données, taille de la mémoire, services requis/fournis, etc.) ;

- la plateforme matérielle utilisée (type du processeur, type de la mémoire, etc.) ;
- les services de communication nécessaires à l'application. L'architecture virtuelle fixe le producteur et le consommateur de chaque service mais pas le type de réalisation de ces services. Une implémentation logiciel/matériel de ces services correspond à une API, un système d'exploitation et des pilotes d'accès à l'adaptateur matériel.

3.5. COLIF : modèle de représentation pour la spécification des systèmes hétérogènes embarqués

COLIF (*CO-design Language Independent Format*) est un modèle de représentation intermédiaire orienté objet. Il représente le système sous forme d'un ensemble de sous-systèmes qui échangent des données entre eux. Les sous-systèmes sont englobés dans des modules hiérarchiques et le réseau de communication est composé de ports et de canaux (Figure 17). Le modèle COLIF permet une séparation claire entre le comportement et la communication. En effet, il permet de décrire des spécifications multi-langages contenant des modèles de communication de natures différentes et spécifiques à des niveaux d'abstraction différents.

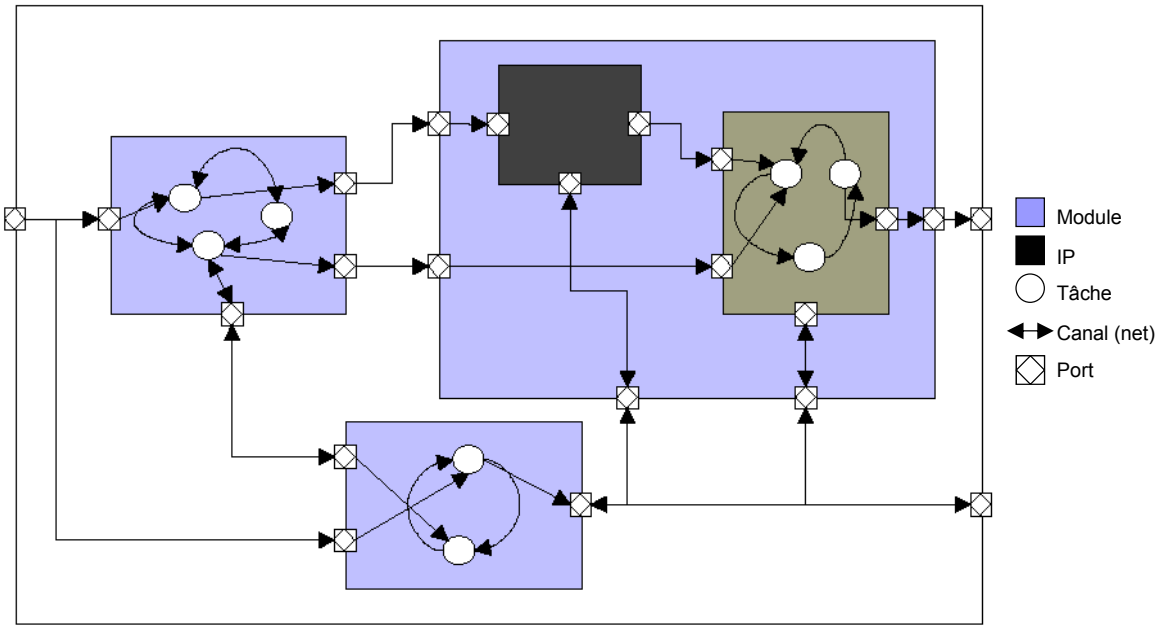

Figure 17. Le modèle de représentation des systèmes hétérogènes embarqués (COLIF)

3.5.1. Les objets de COLIF

COLIF utilise une syntaxe uniforme pour représenter les systèmes hétérogènes décrits à différents niveaux d'abstraction et/ou utilisant différents langages de spécification. Le

système est modélisé sous forme d'un ensemble de modules hiérarchiques qui présente une architecture abstraite. Chaque module est défini par une interface et un contenu qui peut être spécifié ou caché. L'interface est composée d'un ensemble de ports (qui présentent les points de communication du module avec son environnement) et l'ensemble d'opérations possibles réalisées à travers ces ports. Ces opérations (ou méthodes) sont décomposées en deux ensembles complètement différents : un premier ensemble qui présente les opérations internes spécifiques au module qui possède le port et un deuxième ensemble qui présente les opérations externes qui sont spécifiques au reste du système. Le contenu d'un module peut être soit un réseau de sous-modules (une liste de modules) soit des comportements. Si un module décrit seulement la hiérarchie du système alors son contenu est composé d'un réseau de sous-modules. Si le module est une feuille dans la hiérarchie alors son contenu est un réseau de tâches (comportements) ou une référence à un comportement défini à l'extérieur (dans le cas des modules traités en tant que boites noires).

Un *net* (lien) doit être considéré comme une abstraction d'un canal de communication. Par conséquent, un net représente seulement le fait que deux entités dans le système échangent des données d'une façon non spécifiée dans le temps et l'espace. Le net a une sémantique beaucoup plus définie lorsque le modèle abstrait est raffiné et des protocoles de communication spécifiques sont attachés aux nets sous forme de paramètres.

Les trois concepts fondamentaux du modèle COLIF sont : la structure hiérarchique des modules, le port générique et la sémantique de communication attachée aux nets après les étapes de raffinement. Ces concepts caractérisent COLIF comme un modèle de haute flexibilité capable de modéliser des architectures logiciel/matériel à différents niveaux d'abstraction.

La Figure 18 illustre une représentation simplifiée du diagramme des classes COLIF, en utilisant des notations UML (*Unified Modeling Language*). Dans ce diagramme, le modèle d'objets COLIF est divisé en deux parties principales : une partie déclarative et une partie d'instanciation. La partie déclarative représente des objets définissant un modèle ("*template*") réutilisable avec des définitions de propriétés génériques ("*PARAM_DEFS*") et des valeurs par défaut pour certaines propriétés ("*PARAMETER*"). La partie d'instanciation représente les objets effectifs crées à partir de leur déclaration et utilisés pour la construction d'un arbre d'instances.

Comme principe général, chaque concept de base (*MODULE, PORT, et NET*) dans COLIF est divisé en deux parties : une interface (appelée *ENTITY*) et le contenu (*CONTENT*). La partie *ENTITY* a un type (*TYPE*), la structure de base des types valides sont prédéfinis par

COLIF. Le contenu *CONTENT* peut contenir une référence vers un comportement décrit à l'extérieur et une liste d'objets déclarés (*ColifModuleDecl*, *ColifNetDecl*).

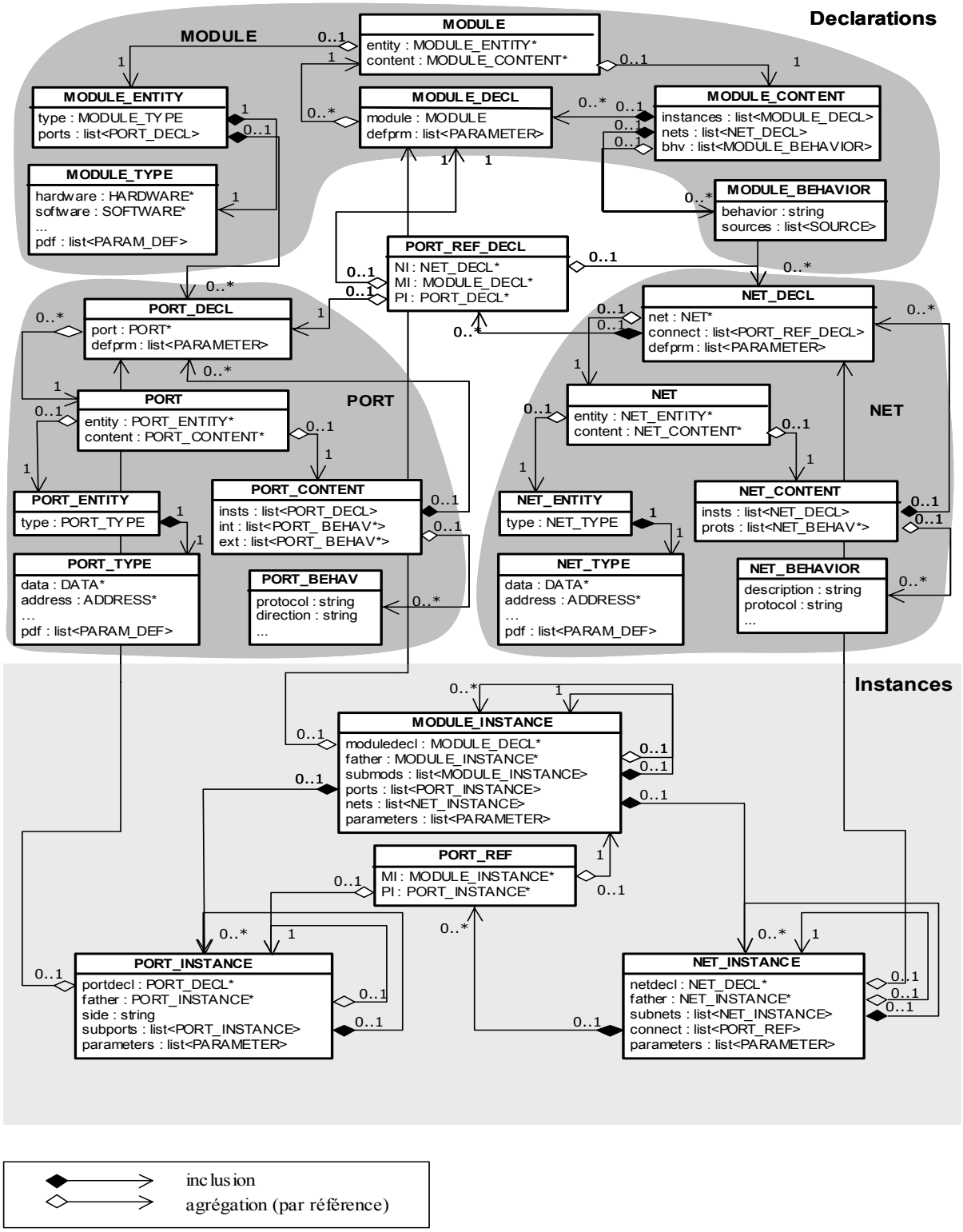

Figure 18. Diagramme de classes de COLIF

3.5.2. Implémentation de COLIF

Pour la réalisation de COLIF, le langage XML (eX*tensible Markup Language*) a été utilisé [XML 00]. XML est un méta-langage standardisé par *W3C*, il a un très grand succès dans le monde de l'informatique grâce à sa souplesse, sa clarté et sa facilité à être analysé. Il facilite l'intégration des outils et l'échange d'informations entre les différents environnements.

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

COLIF est un dérivé de Middle [Gau 01], un méta-langage dérivé de la notation XML; il permet de définir des structures de données complexes. La Figure 19 illustre les couches de langages utilisées pour l'implémentation de COLIF.

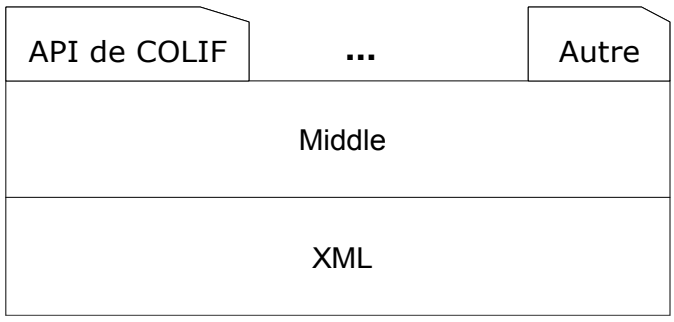

Figure 19. Les couches de langages utilisés pour la réalisation de COLIF

3.5.3. Règles de construction d'un modèle COLIF

La manipulation du modèle COLIF revient à un parcours d'un arbre d'objets instanciés. Les types de ces objets sont bien définis par le langage COLIF. Cependant, pour pouvoir instancier un ensemble d'objets en COLIF, ces derniers doivent être définit et déclarés. Dans une première partie, cette section décrit les différentes règles à respecter (définition, déclaration, instanciation) pour pouvoir construire une représentation syntaxiquement correcte

en COLIF. La deuxième partie met le point sur les règles à respecter pour construire une représentation sémantiquement correcte en COLIF.

3.5.3.1. Règles syntaxiques

Les règles de construction d'un système en format COLIF commencent par la définition des objets de base : *module*, *port* et *net*. En COLIF, ceci correspond respectivement à l'instanciation d'objets de type *ColifModule*, *ColifPort* et *ColifNet*. Chacun de ces objets doit posséder un contenu et une entité. Par conséquent, ils contiennent des références vers d'autres types d'objets qui doivent aussi être créé. En COLIF, ceci correspond :

- pour les modules, à l'instanciation d'objets de type *ColifModuleContent*, et *ColifModuleEntity* ;
- pour les ports, à l'instanciation d'objets de type *ColifPortContent*, et *ColifPortEntity* ;
- pour les nets, à l'instanciation d'objets de type *ColifNetContent*, et *ColifNetEntity*.

L'entité de chaque module, contient :

- une référence vers un objet qui définit le type du module : il faut donc instancier un objet de type *ColifModuleType*. Ce dernier contient une référence vers un objet dont le type est l'un des types prédéfinis par COLIF : un module peut être soit un processeur (*ColifCpu*), une tâche logicielle (*ColifSoftware*), un bloc matériel (*ColifHardware*), une boite noire (*ColifBlackBox*), ou un composé d'autres types de modules (*ColifCompound*) ;
- une référence vers une liste de déclaration de port.

L'entité de chaque port, contient :

- une référence vers un objet qui définit le type du port : il faut donc instancier un objet de type *ColifPortType*. Ce dernier spécifie le type et la direction du port : réception de données (*IN*), émission de données (*OUT*), réception/émission de données (*INOUT*). La direction est inconnu (*UNKNOWN*) pour les ports de type SAP (*Service Access Port*) qui ne sont reliés à aucun canal de communication. L'objet de type *ColifPortType* contient aussi une référence vers une liste de définition de paramètres. La définition d'un paramètre consiste à créer un objet de type *ColifParamDefinition* qui encapsule le nom du paramètre et le type de ses valeurs. Actuellement, les paramètres qui doivent être définit pour le bon fonctionnement du flot ROSES sont décrits dans la deuxième partie de cette section.

L'entité de chaque canal (net), contient :

- une référence vers un objet qui définit le type du net : il faut donc instancier un objet de type *ColifNetType*. Ce dernier contient une référence vers une liste de définition de paramètres.

Le contenu de chaque module contient :

- une référence vers une liste de déclaration de modules (les sous modules). Cette liste est vide lorsqu'il s'agit d'un module feuille dans la hiérarchie. Par exemple un module de type processeur contient un ou plusieurs sous modules de type logiciel (tâches) qui sont des feuilles dans la hiérarchie ;
- une référence vers une liste de déclaration de net. Cette liste est vide pour les modules feuilles dans la hiérarchie. Dans le cas contraire, elle représente l'ensemble des canaux qui connectent les sous modules du module courant ;
- une référence vers une liste d'objets de type *ColifModuleBehaviour* qui représentent le comportement interne du module. Cette liste est vide pour les modules de type processeur ou de type composé. Un objet de type *ColifModuleBehaviour* contient une référence vers une liste d'objets de type *ColifSource* qui encapsulent les liens vers les fichiers de comportement du module et les noms des langages de programmation utilisés.

Le contenu de chaque port contient :

- deux références vers deux listes de déclaration de ports (les sous ports). Dans le cas de ports virtuels, ces listes contiennent l'ensemble des ports internes et externes spécifiques à un port virtuel.

Le contenu de chaque canal (net) contient :

- une référence vers une liste de déclaration de nets (les sous nets). Cette liste contient l'ensemble de sous canaux spécifiques à un canal virtuel donné ;
- une référence vers une liste d'objets de type *ColifNetBehaviour* qui spécifient le nom du protocole utilisé par un canal donné.

Après la définition des différents modules, ports et nets, la deuxième règle consiste à déclarer les modules, les ports et les canaux présents dans le système. En COLIF, ceci revient à créer respectivement des objets de types *ColifModuleDecl*, *ColifPortDecl* et *ColifNetDecl*. L'ensemble de ces objets représente des références vers les définitions de modules, ports et nets qui seront instanciés. En plus de ces déclarations, il faut instancié aussi un objet de type *ColifRoot*. Cet objet représente le module racine de la hiérarchie. Le format COLIF ne supporte qu'un seul et unique objet de ce type (*ColifRoot*).

La dernière règle consiste à instancier les différents objets déclarés. En COLIF, ceci crée un arbre d'instances composé par des objets de type *ColifModuleInstance*, *ColifPortInstance* et *ColifNetInstance*. Le parcours de cet arbre permet un accès aux données de conception à travers la couche API offerte par COLIF.

3.5.3.2. Règles sémantiques

La sémantique dans COLIF est représentée sous forme de paramètres attachés aux modules, ports et net. Par conséquent, chaque module/port/net doit avoir un certain nombre de paramètres contenant les informations de conception nécessaires pour le bon fonctionnement des outils de ROSES. En COLIF, chaque objet de type *ColifModuleInstance*, *ColifPortInstance* et *ColifNetInstance* contient une référence vers une liste d'objets de type *ColifParameter* : ce type d'objet encapsule le nom et la valeur d'un paramètre. Le Tableau 2 résume : (1) l'ensemble de paramètres qui doivent être définis dans chaque modèle COLIF, (2) l'objet auquel chaque paramètre est attaché, (3) la sémantique de chaque paramètre, et (4) les outils pour lesquels chaque paramètre est spécifique.

Paramètres	Attachement	Sémantique	Outil (*)
ADDRESS_INT	Module	Adresse physique du gestionnaire de requêtes d'interruption	HW/SW
CPU	Module	Identifiant d'une architecture locale d'un processeur	HW/SW
DATA_INT_WIDTH	Module	Largeur du bus interne au Coprocesseur de communication	HW
ADRESS_DATA	Port interne	Section pour l'accès au canal	HW/SW
ADRESS_STATE	Port interne	Section pour l'accès au registre d'état du canal	HW/SW
CHAN_NUM	Port interne	Identifiant de canal pour la vecteurisation des requêtes d'interruption	HW/SW
MASK_GET	Port interne	Position du drapeau de lecture	HW/SW
MASK_PUT	Port interne	Position du drapeau d'écriture	HW/SW
SoftPortType	Port interne	Identifiant du pilote	HW/SW
DATA_TYPE	Port interne	Type C des données	HW/SW
IT_LEVEL	Port interne	Priorité d'*irq*	HW/SW
DATA_BIT_WIDTH	Port interne	Longueur native des données en interne au module	HW
DATA_BIT_WIDTH	Port externe	Longueur du média externe inter-modules	HW
CHAN_PRIO	Port externe	Priorités relatives des canaux pour les requêtes d'interruptions	HW
HardPortType	Port externe	Identifiant du protocole RTL	HW
POOL_SIZE	Port externe	Taille en nombre de mots du *buffer*	HW

		interne au adaptateur de canal	
SHB_MASK	Port externe	Masque pour la translation d'adresse entre le plan mémoire locale et le plan mémoire globale	HW
SHB_OFFSET	Port externe	Offset de translation des adresses	HW
SHB_ADRESS	Port externe	Section de mémoire assignée à un port esclave	HW
SHB_PRIO	Port externe	Priorité statique ou rapport cyclique pour l'accès à un bus partagé	HW
Proliferation	Module, Port, et Canal	Réplication de module, port ou canal pour une mise à l'échelle de l'architecture	HW
SystemCType	Port interne/externe, Canal	Type, protocole, niveau d'abstraction d'un port ou d'un canal	Co-Sim
SystemCDataType	Port interne/externe, Canal	Type de données échangées par le canal connecté au port	Co-Sim
(*) Outils de ROSES : HW (outil de génération d'adaptateurs matériels), SW (outil de génération d'adaptateurs logiciels), Co-Sim (outil de génération d'adaptateurs fonctionnels pour la co-simulation)			

Tableau 2. Paramètres de conception attachés aux objets de COLIF

3.6. Les outils de ROSES

Le flot de conception ROSES est basé sur la génération automatique d'adaptateurs logiciels et matériels. Un modèle exécutable est aussi produit afin de valider le système, grâce à la génération automatique d'adaptateurs fonctionnels pour la co-simulation.

La synthèse des adaptateurs est basée sur des bibliothèques qui contiennent des composants d'interface de base à partir desquels les adaptateurs matériels, les adaptateurs fonctionnels et un système d'exploitation dédié (adaptateur logiciel) peuvent être générés. Ces bibliothèques sont étendues avec d'autres éléments nécessaires pour construire des adaptateurs pour les processeurs, les mémoires et pour d'autres composants qui suivent différents standards.

La Figure 20 montre que ROSES propose un ensemble d'outils (générateurs d'adaptateurs) qui permettent la construction des différents adaptateurs. Ces outils communiquent et partagent des données de conception en utilisant le format de représentation intermédiaire COLIF. Grâce à la couche API fournit par COLIF, les outils effectuent des opérations d'import et d'export de données de conception de et vers ce dernier.

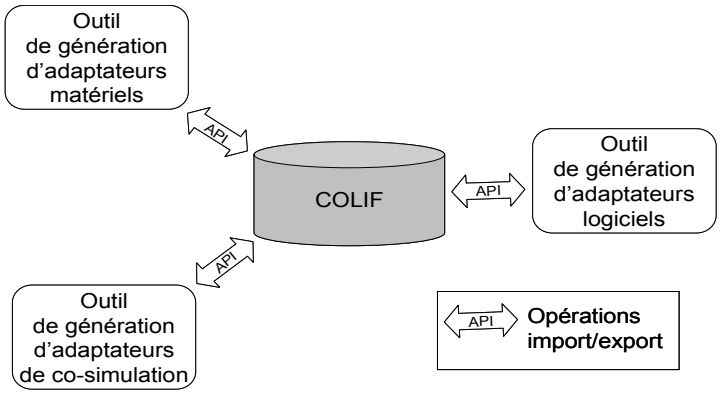

Figure 20. Vue globale de ROSES

3.6.1. Outil de génération des adaptateurs logiciels

L'outil de génération des adaptateurs logiciels est le résultat des travaux d'une thèse effectuée au sein du groupe SLS. Cette section présente seulement les informations pertinentes pour les travaux de cette thèse, plus de détails se trouve dans [Gau 01]. Cet outil génère le code final des systèmes d'exploitation. La fonctionnalité de l'outil consiste à adapter et à assembler des parties de code contenues dans une bibliothèque. L'environnement ainsi que les étapes de l'outil sont détaillés dans les sous sections suivantes.

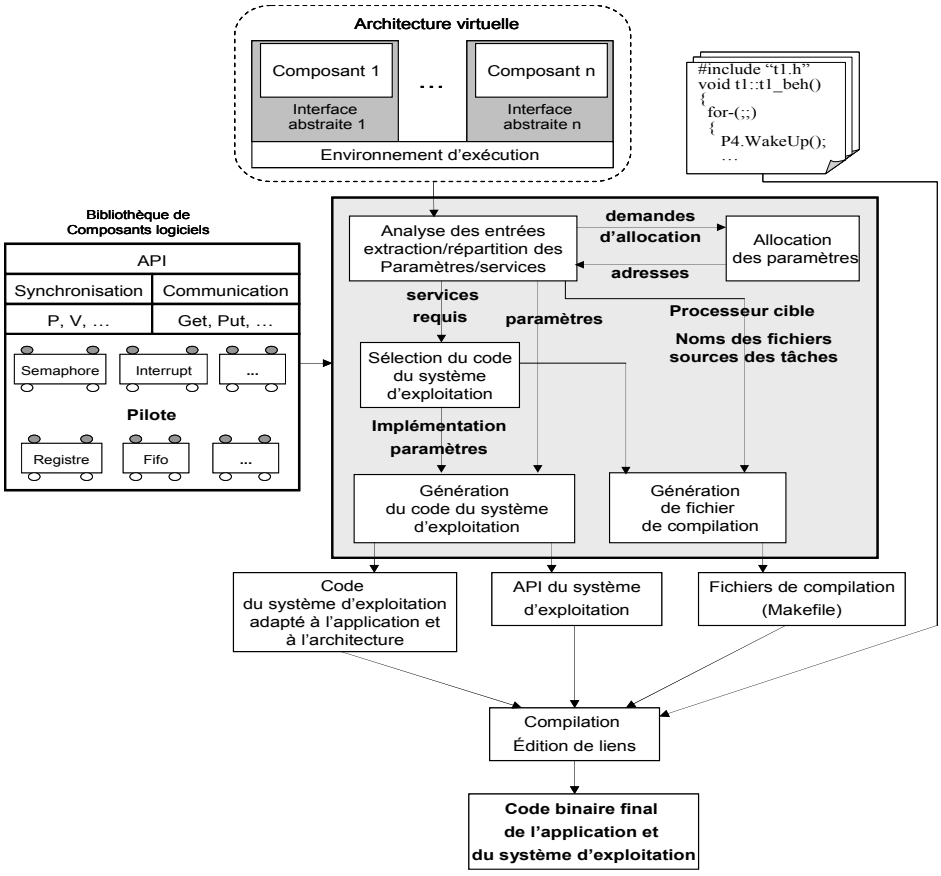

Figure 21. Outil de génération d'adaptateurs logiciels

3.6.1.1. Entrées de l'outil

L'outil de génération des adaptateurs logiciels pour les processeurs programmables prend en entrée deux éléments (Figure 21) :

- une architecture virtuelle qui représente une description annotée de l'architecture matérielle et logicielle ;
- une bibliothèque de composants pour le système d'exploitation.

3.6.1.2. Architecture virtuelle annotée

Une description de l'architecture logicielle et matérielle de l'application est annotée avec des paramètres architecturaux, sous la forme d'un fichier COLIF. Les tâches logicielles sont représentées sous la forme de sous-modules des modules de type processeurs. Les informations et les paramètres contenus dans cette description sont de plusieurs types :

- les informations topologiques provenant de la hiérarchie des modules de la description: il s'agit du nombre de processeurs, du nombre de tâches logicielles, quels processeurs exécutent quelles tâches, par quels ports les tâches communiquent-elles, etc ;
- les informations fournies par les paramètres des modules processeurs : il s'agit des caractéristiques des processeurs tels que les types de processeur (68000, ARM7, etc.), les ressources locales, etc ;
- les informations situées au niveau des ports des processeurs et des canaux : il s'agit des médias utilisés pour les interactions ;
- les informations fournies par les paramètres d'allocation pour les tâches et les divers éléments du système d'exploitation : dans ces paramètres se trouvent les adresses des données en mémoire, les adresses d'interruption, la taille des données, les types de données, les priorités des tâches, etc.

3.6.1.3. Bibliothèque de systèmes d'exploitation

Une bibliothèque contient tous les composants logiciels élémentaires et génériques qui peuvent être spécialisés et assemblés pour obtenir un système d'exploitation complet, ainsi que toutes les informations qui définissent ces composants et leur environnement d'utilisation.

a. Structures et relations utilisées par la bibliothèque : cette bibliothèque est basée sur trois concepts qui donnent chacun une vue différente du système d'exploitation :

- l'élément : représente une partie du système d'exploitation de sorte que la structure d'un système d'exploitation complet est modélisée par un ensemble d'éléments liés ;

- le service : représente une fonctionnalité du système d'exploitation, de telle sorte que le comportement fonctionnel d'un système d'exploitation complet est modélisé par un ensemble de services ;
- l'implémentation : représente une réalisation particulière d'une partie du comportement précis du système d'exploitation, de telle sorte que le code d'un système d'exploitation complet est vu comme un assemblage d'implémentations paramétrées. Une implémentation peut être compatible avec certaines architectures matérielles (notamment le processeur), et incompatible avec d'autres. C'est aux implémentations que les composants logiciels génériques du système d'exploitation sont associés.

Ces trois concepts ne sont pas indépendants, il y a des relations entre les éléments et les services, les éléments et les implémentations, et les services et les implémentations.

- les relations entre les éléments et les services : ce sont des relations de dépendance indirectes entre les éléments basées sur le concept de services requis et de services fournis. Ces relations sont présentées sous forme d'un graphe orienté (Figure 22). Chaque nœud du graphe représente soit un élément, soit un service. Chaque arc orienté relie soit un nœud élément à un nœud service pour modéliser le fait que l'élément requiert le service, soit un nœud service à un nœud élément pour modéliser le fait que l'élément fournit le service.

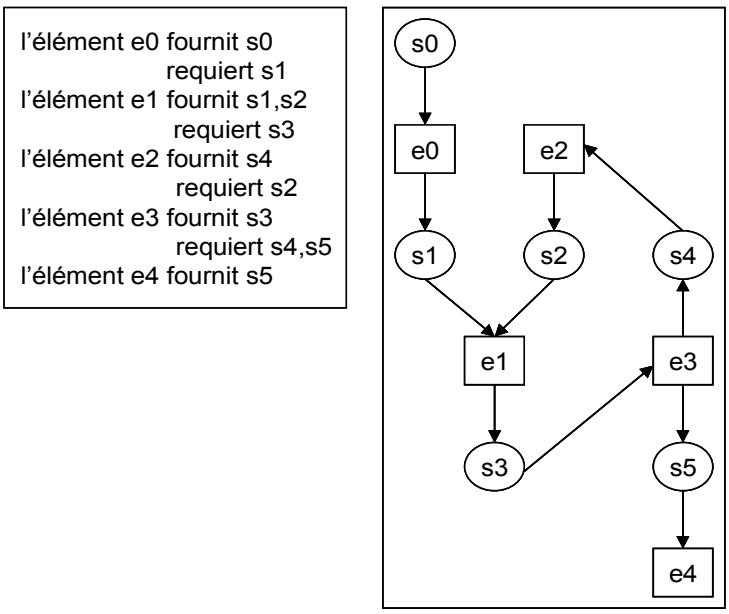

Figure 22. Relations de dépendance entre éléments et services

- les relations entre les éléments et les implémentations : chaque élément possède une ou plusieurs implémentations. Les implémentations d'un élément sont organisées suivant un arbre hiérarchique (Figure 23), les fils d'une implémentation étant toujours compatibles avec moins d'architectures que leur père. Pour décrire complètement un élément compatible avec une architecture donnée, il faut prendre les sources associées à chaque implémentation traversée et parcourir l'arbre en profondeur avec comme critère de choix la compatibilité avec les processeurs et médias. Si aucune feuille n'est atteinte, alors l'implémentation est invalide pour l'architecture visée. Si toutes les implémentations d'un élément sont invalides, alors l'élément est lui aussi invalide. La Figure 23 montre un exemple d'arbre d'implémentations d'un élément compatible avec le processeur ARM7 (les implémentations en gras sont celles compatibles avec le processeur ARM7).

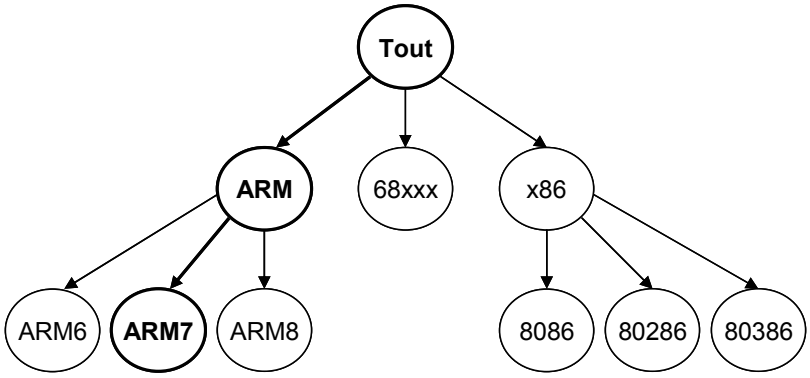

Figure 23. Arbre d'implémentation d'un élément compatible avec le processeur ARM7

- les relations entre les services et les implémentations permettent d'éviter l'inclusion du code d'implémentation d'un service fournis par un élément mais non requis par d'autres éléments dans le système d'exploitation généré.

b. Organisation des services du système d'exploitation : une fonctionnalité de communication est au minimum définie par deux services : un service représentant la fonctionnalité de communication du point de vue de l'application logicielle (c'est-à-dire à haut niveau), et un service la représentant du point de vue du pilote de communication (c'est-à-dire à bas niveau). Les différentes familles de services choisies sont les suivantes :

- *API* : regroupe tous les services représentant l'API du système d'exploitation utilisable par l'application logicielle. Cette famille contient trois sous familles :
  - *IO* : regroupe tous les services liés aux communications utilisables par l'application logicielle ;

  - *Synchronization* : regroupe tous les services liés aux synchronisations utilisables par l'application logicielle (par exemple les sémaphores) ;
  - *Other* : regroupe les services de haut niveau n'entrant pas dans les familles précédentes.
- *Kernel* : regroupe tous les services concernant le noyau du système d'exploitation. Cette famille contient les sous-familles suivantes :
  - *boot* : regroupe tous les services liés au démarrage du système d'exploitation ;
  - *Cxt* : regroupe tous les services liés à la gestion des contextes associés aux tâches. Les éléments fournissant ces services sont toujours spécifiques au processeur cible ;
  - *Schedule* : regroupe tous les services liés à l'ordonnancement des tâches (par exemple le tourniquet, ou les priorités mais aussi le mise en sommeil ou le réveil d'une tâche) ;
  - *Task* : regroupe tous les services liés à la gestion des tâches. En pratique, elle fait le lien entre les autres sous-familles de la famille *kernel*, et contient les tables de tâches.
- *Data* : regroupe tous les services liés aux structures de données (comme les FIFO).
- *Interrupt* : regroupe tous les services liés aux interruptions (par exemple la gestion des fonctions d'interruption, ou les appels système).
- *Synchronization* : regroupe tous les services liés aux mécanismes de synchronisation internes au système d'exploitation.
- *Driver* : regroupe tous les services représentant des gestionnaires de périphériques. Elle peut posséder de nombreuses sous-familles, dont *Driver/IO* pour les services liés aux communications de bas niveau.

c. Représentation du comportement des éléments : une bibliothèque de comportements réalisant les fonctionnalités des divers éléments qui composent le système d'exploitation. Une partie du code d'un système d'exploitation est écrite dans un langage de programmation de haut niveau tel que le langage C. Cependant certaines parties du système d'exploitation, comme le changement de contexte entre deux tâches, sont très spécifiques au processeur et sont écrites en langage d'assemblage du processeur. La génération du code du système d'exploitation consiste en l'assemblage et le paramétrage de parties de code. Cela nécessite de pouvoir modifier le code, c'est pour cela que ces parties de code sont encapsulées

dans des macros. La génération revient à appeler un programme d'expansion de macros avec les bons paramètres.

3.6.1.4. Sortie de l'outil

L'outil de génération des adaptateurs logiciels produit en sortie :

- le code source du système d'exploitation généré, adapté à l'architecture cible et à l'application. Ce code est constitué de plusieurs fichiers pouvant être de divers langages comme le C ou les langages d'assemblage ;
- des entêtes permettant d'adapter le code de l'application au système d'exploitation généré et à l'architecture ;
- des fichiers de compilation, dans le format *Makefile* qui permettent l'automatisation de la compilation ;
- des rapports fournissant des informations sur actions effectuées durant la génération, les erreurs survenues durant ce dernier et les caractéristiques du système d'exploitation généré.

3.6.1.5. Etapes de l'outil

La génération automatique du système d'exploitation est composée de quatre étapes essentielles.

a. Analyse des entrées : l'outil prend en entrée une description de l'architecture logicielle et matérielle annotée sous forme d'un fichier COLIF. Cette étape consiste à récupérer les informations requises par les autres étapes sous un format directement utilisable :

- l'étape de sélection de code a besoin avant tout des services d'API requis par les tâches. Ensuite, pour raffiner la sélection, cette étape a besoin des services de communication utilisés, et des types de processeurs ;
- l'étape de génération du code du système d'exploitation doit connaître les éléments du système d'exploitation ainsi que tous les paramètres qui spécialisent cette génération pour l'architecture et l'application. Ces paramètres sont rassemblés sous le terme de paramètres d'allocation, et correspondent aux adresses des données, tailles des données, types de données, etc ;
- l'étape de génération des fichiers de compilation a besoin de la liste des fichiers à compiler et des types des processeurs cibles. Le premier type d'information est fourni

par l'étape de sélection de code et par l'étape d'adaptation du code de l'application. Le second type est juste une chaîne de caractère représentant le type de processeur.

b. Sélection du code : l'étape de sélection de code prend en entrée des listes de services d'API et de services de communication produits par l'étape d'analyse de la description de l'architecture. Cette étape est divisée en deux étapes :

- la première étape calcule la fermeture transitive des services d'API. Elle construit le graphe de tous les éléments et services valides pour l'architecture en dépendance directe ou indirecte avec les services initiaux ;
- la deuxième étape est appliquée sur le graphe généré par la première étape. Elle construit la liste des éléments nécessaires et suffisants pour fournir tous les services en utilisant les services de communication demandés, c'est-à-dire qu'elle enlève les éléments et les services non utiles pour faire fonctionner l'application sur l'architecture.

c. Génération du code du système d'exploitation : l'étape de génération de code prend en entrée les listes d'éléments provenant de l'étape de sélection de code et les paramètres produits par l'étape d'analyse de la description de l'architecture. Cette étape construit une liste de tous les paramètres associés à chaque élément précédemment sélectionné. Ceci permet de faire appel au programme d'expansion de macros qui permet le paramétrage et l'optimisation des éléments afin de générer leur code spécialisé respectif. Ensuite, ces éléments sont assemblés pour concrétiser sous forme de code les relations entre chaque élément et les services requis et fournis.

d. Génération des fichiers de compilation : le but de l'étape de génération des fichiers de compilation est de produire les fichiers *Makefile* qui permettent l'automatisation de la compilation.

3.6.2. Outil de génération des adaptateurs matériels

L'outil de génération des adaptateurs matériels est le résultat de travaux d'une thèse effectuée au sein du groupe SLS. Cette section présente seulement les informations pertinentes pour les travaux de cette thèse, plus de détails se trouve dans [Lyo 03]. Cet outil génère le code RTL des adaptateurs matériels permettant de connecter les processeurs au réseau de communication. La fonctionnalité de l'outil consiste à assembler les composants d'interface matériels de base qui sont décrits dans un environnement spécifique aux

architectures locales des processeurs. L'environnement ainsi que les étapes de l'outil sont détaillés dans les sous sections suivantes.

3.6.2.1. Entrées de l'outil

L'outil de génération des adaptateurs matériels pour les processeurs prend en entrée trois éléments (Figure 24) :

- une architecture virtuelle annotée ;
- une bibliothèque de modèles décrite en COLIF, appelée aussi bibliothèque d'architecture interne ;
- une bibliothèque de comportement.

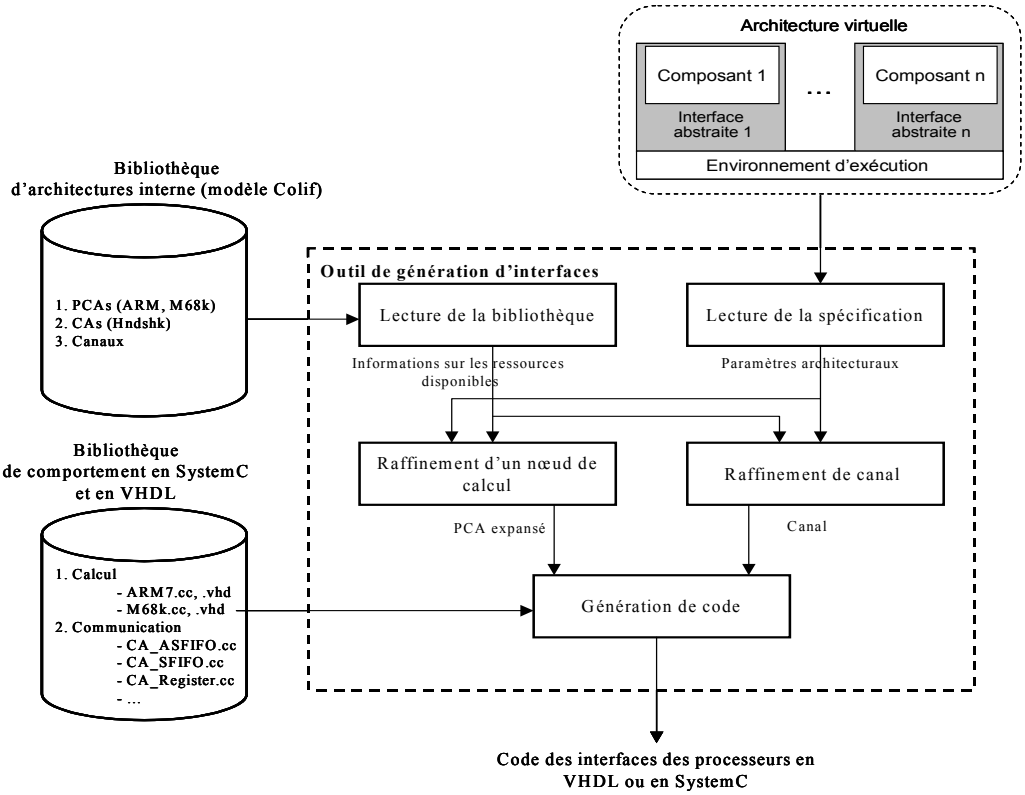

Figure 24. Outil de génération d'adaptateurs matériels

3.6.2.2. Architecture virtuelle annotée

Une description de l'architecture logicielle et matérielle annotée avec des paramètres architecturaux (fichier COLIF). Ces paramètres caractérisent :

- les besoins du logiciel de l'application (type de données, tailles des mémoires, etc.) ;
- la plateforme matérielle utilisée (types des processeurs, types des mémoires, etc.) ;
- les interfaces logiciel/matériel. Plus précisément, les services de communication nécessaires à l'application.

3.6.2.3. Bibliothèque de l'outil

a. Bibliothèque d'architectures internes : une bibliothèque d'architectures internes décrite au format COLIF représente la décomposition hiérarchique et l'interface de chaque module. Cette bibliothèque contient une structure d'un module appelée architecture interne ou PCA (*Processor Centric Architecture*). Ce module représente une architecture générique d'un processeur.

Le PCA (Figure 25) est composé de :

- un module « Proc » qui représente le processeur ;
- un décodeur d'adresse ;
- un gestionnaire d'interruption (IC) ;
- une mémoire locale RAM pour stocker les variables de calcul ;
- une mémoire ROM pour stocker l'image du programme à exécuter ;
- un coprocesseur de communication (CC) qui correspond à une architecture abstraite de l'adaptateur du processeur qui connecte ce dernier au canal de communication. Il est composé d'un adaptateur de module (MA) spécifique au processeur, d'un bus interne et d'un adaptateur de canal virtuel (VCA). Ce dernier est abstrait parce que l'interface, du côté média externe, n'est pas encore définie et parce qu'il ne correspond encore à aucune implémentation.

Cette bibliothèque contient aussi des modèles structurels d'adaptateurs de communication (CA). Les interfaces de ces modèles sont généralement basées sur des protocoles de poignée de main.

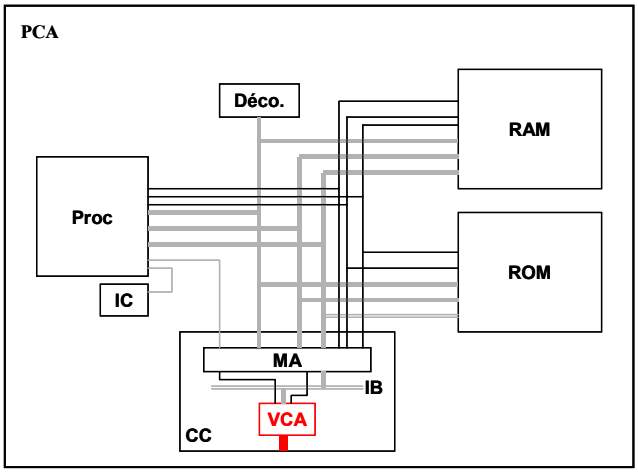

Figure 25. Architecture interne d'un nœud de calcul

b. Bibliothèque de comportement : une bibliothèque de comportement réalisant les fonctionnalités des nœuds de calcul (PCA) et des composants de communication.

3.6.2.4. Sortie de l'outil

La sortie de l'outil de génération des interfaces est le code des adaptateurs matériels du processeur cible. Le code généré est en SystemC ou en VHDL.

3.6.2.5. Etapes de l'outil

La génération automatique des adaptateurs matériels de processeurs est composée de cinq étapes essentielles :

a. Lecture de la spécification : l'outil prend en entrée une description de l'architecture logicielle et matérielle en format COLIF. Cette étape consiste à récupérer les paramètres de configuration qui annotent la spécification d'entrée.

b. Lecture de la bibliothèque : cette étape consiste à charger la bibliothèque architecturale. Cette lecture permet de connaître la disponibilité des ressources architecturales nécessaires pour la réalisation de l'architecture cible. Cette bibliothèque contient différents composants de l'architecture cible. La structure de chaque composant de cette bibliothèque est décrite en COLIF. La fonctionnalité est indiquée sous forme d'un chemin vers l'implémentation fournie par la bibliothèque de comportements.

c. Raffinement d'un nœud de calcul : un nœud de calcul est décrit comme un module virtuel dont l'interface est composée de ports virtuels hiérarchiques regroupant deux types de ports : des ports internes spécifiques à la fonctionnalité du nœud de calcul et des ports externes spécifiques aux canaux de communication. Le raffinement d'un nœud de calcul consiste à :

- choisir et configurer une architecture interne PCA à partir de la bibliothèque d'architectures en COLIF en utilisant les paramètres d'allocation. Cette étape utilise deux paramètres : « CPU » pour identifier le processeur et « ADDRESS_INT » qui correspond à l'adresse physique du gestionnaire d'interruptions ;
- définir la structure de l'adaptateur de canal virtuel (VCA) de l'architecture interne PCA. En effet, selon les paramètres d'allocation attachés aux ports internes et aux ports externes, on fixe le nombre et l'interface des adaptateurs de canaux (CA) qui vont implémenter le VCA. Le choix d'un CA dépend des paramètres liés au protocole

de communication et au bus interne. Un CA peut être dupliqué selon un paramètre d'allocation appelé « proliferation ».

d. Raffinement de canal virtuel : le raffinement d'un canal virtuel est composé de deux parties :

- une analyse de tous les ports liés au canal permet de donner une liste d'attributs caractérisant la spécificité du canal à sélectionner à partir de la bibliothèque ;
- une partie sélection utilise la liste précédente pour choisir un modèle de canal parmi les différents modèles disponibles dans la bibliothèque. Selon le paramètre « proliferation », le canal comme son adaptateur peuvent être également répliqués.

e. Génération du code : à chaque instance d'élément de l'architecture interne PCA (sauf le processeur et les mémoires locales), correspond un modèle d'implémentation générique décrit en macro-langage. Une implémentation générique n'est ni simulable ni synthétisable, car elle n'est pas encore configurée. Le type de données est abstrait, le nombre, la taille et la direction des ports sont encore génériques. Pour une application donnée, on utilise les paramètres d'allocation (type de données, taille du port, etc.) pour configurer le code générique sélectionné à partir de la bibliothèque comportementale. Cette expansion du code génère le code final de l'adaptateur de processeur (*Communication Co-processor* – CC) qui est simulable et synthétisable.

3.6.3. Outil de génération des adaptateurs de co-simulation

L'outil de génération des adaptateurs de co-simulation est l'un des résultats des travaux effectués au sein du groupe SLS. Cette section présente seulement les informations pertinentes pour les travaux de cette thèse, plus de détails se trouve dans [Nic 02]. Cet outil suit le modèle global de co-simulation de la Figure 26, qui utilise différents types de simulateurs (VHDL, ISS, etc.), des composants fonctionnels pour la simulation de comportements indépendamment de leurs implémentations en matériel et/ou en logiciel, des composants représentant des sous-systèmes d'interface (adaptateurs de co-simulation) et un bus de co-simulation. L'outil génère des modèles exécutables pour la co-simulation du système et les adaptateurs de co-simulation permettant de connecter les différents simulateurs au bus de co-simulation. La fonctionnalité de l'outil consiste à assembler les composants d'interface de base. L'environnement ainsi que les étapes de l'outil sont détaillés dans les sous sections suivantes.

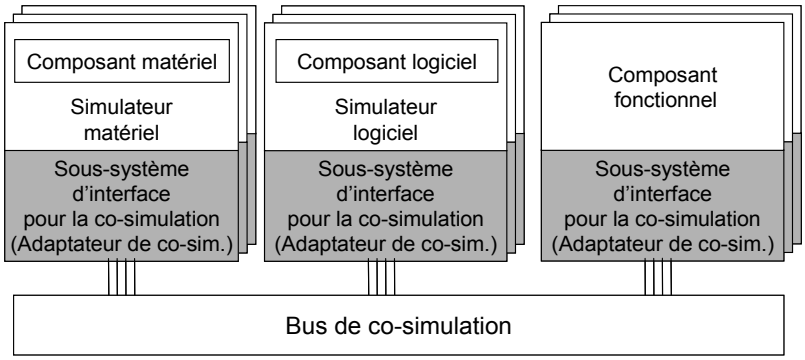

Figure 26. Modèle architectural utilisé pour la co-simulation

3.6.3.1. Entrée de l'outil

L'outil de génération des adaptateurs de co-simulation prend en entrée deux éléments :

- une architecture virtuelle qui représente une description annotée de l'architecture matérielle et logicielle ;
- une bibliothèque de composants d'interface (objets *template*) pour la génération d'adaptateurs de co-simulation.

3.6.3.2. Bibliothèque de l'outil

La bibliothèque du générateur d'adaptateur de co-simulation est composée de composants de base de l'interface de communication (les adaptateurs de ports, les adaptateurs de canaux et le média de communication interne), des éléments pour la génération des interfaces du simulateur, et d'un bus de co-simulation. Les éléments de la bibliothèque ont été implémentés à l'aide de SystemC [Sys 00]. Nous allons introduire brièvement quelques concepts de base du langage SystemC. En fait, le système est modélisé comme étant une collection de modules qui contiennent des processus, et qui communiquent en utilisant des ports, des canaux, et des interfaces. Les processus définissent le comportement des modules. Une interface définit un ensemble de méthodes, mais ne les implémente pas. Un canal implémente un ou plusieurs méthodes d'une interface [Sys 01]. Un port permet à un module, et désormais ses processus, d'accéder à un canal. Le port est aussi définit en terme d'interface, c'est-à-dire qu'il ne peut être utilisé qu'avec les canaux qui implémentent cette interface. Le concept d'interface permet l'utilisation d'un schéma de communication appelé en anglais (*Interface Method Call – IMC*), qui s'applique à un processus appelant une méthode de l'interface implémentée au niveau d'un canal.

a. Les interfaces de communication : les interfaces de communication sont composées d'adaptateurs de module, d'adaptateurs de canal et d'un média de communication interne:

- l'adaptateur de module est en charge d'effectuer les différentes conversions suivantes : conversions de données, conversions des protocoles, arbitrage. Il encapsule deux ports : le premier port sert pour accéder au module et doit donc avoir un type compatible avec ce module (type opposé (en direction) de celui du module), le deuxième port est connecté au média de communication interne ;
- l'adaptateur est un simple processus SystemC ;
- le média de communication interne utilise des éléments de communication fournis par la bibliothèque SystemC : la communication par des RPC (*Remote Procedure Call*), les événements, les signaux physiques.

Les éléments de la bibliothèque sont des modèles (objets *templates*) SystemC, ils peuvent donc être configurés. Parmi les paramètres de configuration, on peut citer par exemple les types des données transférées, la taille des données ou les caractéristiques du protocole de communication.

b. Les interfaces du simulateur : concernant les éléments de bibliothèque pour les interfaces des simulateurs, ils sont basés sur la bibliothèque SystemC et la bibliothèque Unix pour la communication inter-process, IPC (*Inter-Process Communication*). Chaque simulateur participant à la simulation est en fait un processus fils d'un processus SystemC avec lequel il communique par des IPC (utilisant les mémoires partagées et les sémaphores). Pour la communication des différents simulateurs par des IPC nous avons utilisé les différentes bibliothèques fournies par les simulateurs en vue de la communication avec un environnement extérieur.

c. Le bus de co-simulation : le bus de co-simulation doit pouvoir modéliser la communication de niveau macro-architecture ainsi que la communication de niveau micro-architecture. Pour modéliser la communication de niveau micro-architecture, le bus de co-simulation utilise les signaux standard fournis par SystemC. Pour la communication au niveau macro-architecture, il utilise des méthodes implémentées dans les canaux spécifiques au protocole de communication.

3.6.3.3. Sortie de l'outil

La sortie de l'outil de génération des adaptateurs de co-simulation est un modèle de simulation de l'architecture logiciel/matériel, et le code des adaptateurs de co-simulation.

3.6.3.4. Etapes de l'outil

a. Analyse de la description de l'architecture : l'étape d'analyse de la description de l'architecture logicielle et matérielle revient à l'analyse de la représentation de l'architecture virtuelle décrite en COLIF. Cette étape consiste à récupérer et traduire les informations dans une structure de données internes qui sera transmise aux étapes suivantes :

- l'étape de construction des interfaces de simulateur a besoin des informations attachées aux ports sous forme de paramètres et représentants les types des données et la direction ;
- l'étape de construction des interfaces de communication a besoin de différents informations attachés aux modules et aux ports sous forme de paramètres, tels que le niveau d'abstraction des modules, les types des modules (ex. logiciel/matériel), les types des protocoles de communication, les types de données et les caractéristiques du protocole de communication (ex. taille de FIFO), etc.

b. Construction des interfaces de co-simulation : l'étape de construction des interfaces de co-simulation inclut la construction des interfaces du simulateur et de l'interface de communication.

Etant donné que le bus de co-simulation est implémenté en SystemC, une interface de simulateur sera instanciée pour chaque module qui n'est pas spécifié en SystemC. Pour la construction des interfaces de communication, cette étape commence par parcourir la liste des ports internes et externes de chaque module pour tester la nécessité d'une telle interface. Dans le cas où le module nécessite une interface de communication, une analyse de l'enveloppe du module est faite afin d'obtenir les informations nécessaire pour pouvoir sélectionner des éléments de la bibliothèque. Ainsi, l'analyse du module et de ses ports internes indiquera les adaptateurs de ports nécessaires et leurs paramètres. Le niveau d'abstraction du module, son type et le type de protocole de communication indiqueront le type de l'adaptateur de port. Les types de données et les caractéristiques du protocole de communication fournissent les paramètres nécessaires pour la configuration de l'adaptateur de port. De même, l'analyse du réseau de communication et des ports externes de chaque module indiquera les types des adaptateurs de canal nécessaires et leurs paramètres : le niveau d'abstraction et le protocole de

communication du canal virtuel indiquent le type d'adaptateurs de canal ; les types de données et les caractéristiques du protocole de communication donneront des paramètres pour la configuration des adaptateurs de canal. Une fois que les types des adaptateurs de canal et des adaptateurs de port nécessaires ont été décidés, une table d'information sur la topologie de l'interface de communication et la configuration des éléments est conçue. Cette table sera l'entrée de l'étape suivante du flot de génération des modèles de simulation. L'analyse des ports virtuels nous donnera les paramètres de configuration du média de communication interne.

c. Construction du bus de co-simulation : cette étape réalise l'instanciation du bus de co-simulation et l'intégration des interfaces de communications dans le modèle global du système. Ceci correspond à la création d'un composant SystemC où les différents adaptateurs de canal et adaptateurs de port sont instanciés avec les paramètres nécessaires et interconnectés avec les composantes du système.

d. Génération de code : cette étape permet la génération des fichiers de code SystemC et d'un fichier de compilation.

3.7. Analyse

Initialement, chaque outil du flot ROSES est le fruit des travaux d'une thèse. Chaque outil a été réalisé pour répondre à un problème bien spécifique : intégration des composants matériels (adaptation matérielle), intégration des composants logiciels (adaptation logicielle) et la validation du système (adaptation fonctionnelle pour la co-simulation). Par conséquent et grâce au format COLIF, chaque outil a été construit indépendamment des autres :

- chaque outil possède sa propre bibliothèque de composants, utilise une certaine granularité des composants et une certaine structure de description de la bibliothèque. Par exemple, la structure de la bibliothèque du générateur des systèmes d'exploitation est décrite sous forme d'un graphe de dépendance entre les différents composants d'interface logiciels en terme de services requis/fournis alors que la structure de la bibliothèque du générateur d'adaptateurs matériels est décrite en utilisant COLIF. Le générateur des adaptateurs de co-simulation n'utilise aucun modèle pour décrire la structure de sa bibliothèque. Cet outil admet l'existence des différents composants d'interface nécessaires sous des différentes bibliothèques UNIX ;
- chaque outil a ses propres étapes de composition de composants ;

- chaque outil a sa propre technique de composition. Par exemple, la technique de composition du générateur des systèmes d'exploitation est basée sur les services requis/fournis par les composants alors que la technique de composition du générateur d'adaptateurs matériels est basée sur une architecture générique des adaptateurs matériels. L'utilisation de cette architecture générique présente des limites surtout lors de l'exploration de différents partitionnement de la communication logiciel/matériel.

Bien que le principe fondamental de ces outils soit l'assemblage de composants, chacun d'entre eux utilise sa propre bibliothèque, ses propres étapes de composition, et sa propre technique de composition. Ceci cause une grande difficulté lors de la maintenance de l'environnement ROSES. Une unification des étapes de composition, des techniques de composition et des bibliothèques de composants présente une solution potentielle pour construire un environnement ouvert à l'intégration des composants logiciel/matériel.

De plus, l'analyseur (*parser*) du format intermédiaire COLIF ne vérifie pas la bonne formation de ce dernier. Un mécanisme de vérification de la bonne formation et de la cohérence du modèle COLIF est une solution nécessaire pour assurer le bon fonctionnement des outils de ROSES et assurer une interopérabilité entre les différents outils existants et les outils externes à intégrer.

Cela nécessite une réorganisation du flot de conception, des outils de génération d'adaptateurs existants et la mise en place d'un mécanisme de vérification du modèle COLIF afin que ROSES représente un environnement de conception unifié et ouvert à l'intégration de composants logiciel/matériel et à l'intégration d'outils. Cette réorganisation fera l'objet du chapitre suivant.

3.8. Conclusion

Le flot de conception d'architectures ROSES, suit une approche permettant aux concepteurs une réutilisation effective d'une grande variété de composants logiciel/matériel. Ceci est fait par une intégration de ces composants dans un modèle global du système, grâce à la génération automatique des adaptateurs logiciel/matériel. Le flot ROSES permet aussi la génération d'un modèle exécutable afin de valider le système, grâce à la génération automatique des adaptateurs de co-simulation.

Ce chapitre a présenté les modèles et les composants définis par le flot ROSES. Les différents schémas d'intégration des composants logiciel/matériel et le concept d'architecture virtuelle définis et utilisés par ROSES, ont été aussi présentés. L'environnement logiciel du format de représentation intermédiaire COLIF a été présenté. COLIF permet la représentation

des systèmes hétérogènes embarqués, une séparation claire entre le comportement et la communication, et représente le modèle de référence des différents outils de ROSES. Une présentation des outils fournis par ROSES a été faite tout en mettant l'accent sur les informations pertinentes et nécessaires pour les travaux de cette thèse. Une analyse des différentes outils, nous a permis d'envisager une réorganisation du flot de conception ROSES afin d'en faire un environnement ouvert à l'intégration de composants logiciel/matériel et à l'intégration d'outils de conception. Cette nouvelle approche fera l'objet du Chapitre 4.

Chapitre 4. Evolution de ROSES en vue d'en faire une méthodologie d'intégration de composants logiciel/matériel et un modèle d'intégration d'outils

Sommaire

4.1. Introduction

Ce chapitre décrit l'évolution du flot ROSES vers un environnement ouvert pour l'intégration des composants logiciel/matériel et pour l'intégration d'outils de conception.

Pour l'intégration de composants, on définit une approche à deux niveaux : le niveau système et le niveau interface. En effet, cette intégration nécessite un travail d'adaptation qui consiste à construire des sous-systèmes d'interface très sophistiqués (adaptateurs). Ces sous-systèmes d'interface peuvent être de différents types : (1) logiciels, pour adapter les composants de l'application logicielle à la partie matérielle du système ; (2) matériels, pour adapter les interfaces des composants matériels à l'interface du réseau de communication embarqué ; (3) virtuels (ou encore fonctionnels), pour adapter les interfaces des composants et/ou les interfaces des différents simulateurs au bus de communication afin de simuler le système à différents niveaux d'abstraction. Au moment de l'intégration au niveau système, ces différents types de sous-systèmes d'interface peuvent être manipulés en tant que composants qui servent à adapter les composants logiciel/matériel au réseau de communication. Au niveau interface, ces sous-systèmes d'interface sont construits par un assemblage de composants d'interface logiciel/matériel élémentaires. ROSES en tant que tel peut accomplir la tâche d'intégration au niveau interface.

Pour l'intégration d'outils, on définit une approche autour d'un format intermédiaire neutre permettant la description des systèmes hétérogènes (COLIF présenté dans la section 3.5 du Chapitre 3). Cette approche utilise des processus de traduction qui permettent la traduction des données et des modèles de conception, produits par les outils externes, vers le format intermédiaire. En plus, un outil de vérification additionnel est nécessaire pour s'assurer de la bonne formation du format intermédiaire et que l'échange de données de conception au cours du processus de traduction est en accord avec les exigences du format intermédiaire et des outils consommateurs (c'est-à-dire les outils de ROSES).

Ce chapitre est organisé en deux grandes parties : dans la première partie, nous proposons une méthodologie unifiée qui permet : (1) au niveau système, l'intégration des composants logiciel/matériel afin de générer des modèles architecturaux pour la synthèse et des modèles exécutables pour la co-simulation; et (2) au niveau interface, l'assemblage des composants d'interface logiciel/matériel élémentaires afin de générer les différents types de sous-systèmes d'interface. Dans la deuxième partie, nous proposons un modèle d'intégration d'outils qui s'articule autour du format intermédiaire COLIF. Un outil de vérification de la bonne formation de COLIF est aussi présenté. Il s'agit de : (1) Coral, un langage développé au

cours des travaux de cette thèse pour pouvoir spécifier des règles syntaxiques et sémantiques spécifiques au modèle COLIF et aux outils de ROSES et (2) COLIF-*Checker*, un moteur de vérification qui permet de prouver la bonne formation du modèle COLIF. Cette vérification se base sur l'ensemble des règles spécifiées à l'aide de Coral.

4.2. Méthodologie unifiée pour l'intégration de composants logiciel/matériel

Cette section introduit une méthodologie unifiée pour l'intégration automatique de composants logiciel/matériel. Cette méthodologie nous permet de générer des modèles raffinés de systèmes hétérogènes embarqués pour la synthèse et la co-simulation. Selon la nature du modèle à générer et des composants manipulés, la méthodologie unifiée est utilisée à deux niveaux d'intégration différents :

- *au niveau système*, l'intégration des composants est utilisée pour obtenir des modèles globaux de systèmes sur puce pour la synthèse et la co-simulation à partir de composants hétérogènes : matériel, logiciel, fonctionnel et des adaptateurs de communication (sous-systèmes d'interface complexes) ;
- *au niveau interface*, l'intégration des composants est utilisée pour obtenir des sous-systèmes d'interface sophistiqués à partir d'un assemblage de composants d'interface de base homogènes : des blocs matériels, des fonctions logicielles et des modèles de simulation.

4.2.1. Flot unifié pour l'intégration des composants

Afin de manipuler la composition de plusieurs composants logiciel/matériel ayant des granularités différentes, nous avons défini un flot unifié dédié à l'intégration des composants. Ce flot est illustré par la Figure 27.

L'entrée de ce flot est une architecture abstraite qui représente le système en terme de composants interconnectés. Chaque composant fournit et requiert un ensemble de services. L'interconnexion entre les composants est basée sur le principe des services requis/fournis. L'architecture abstraite est annotée avec des paramètres de conception qui guideront le processus d'intégration.

La sortie du flot est une architecture raffinée qui représente le système après le processus d'intégration. Elle peut être une architecture globale de système embarqué (prête pour la phase de prototypage ou pour la co-simulation) ou une architecture de sous-systèmes d'interface (adaptateurs logiciels, matériels ou fonctionnels pour la co-simulation).

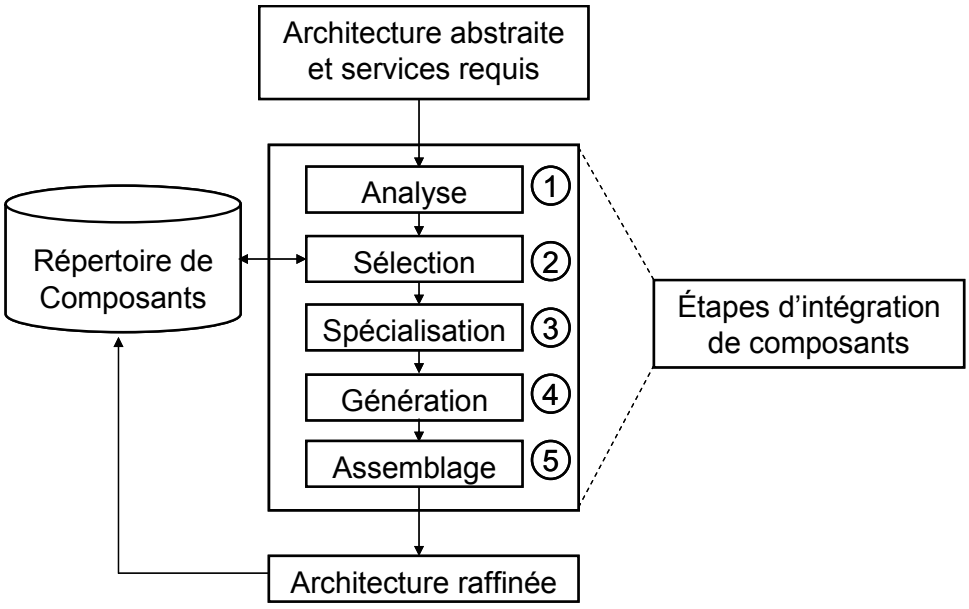

Figure 27. Flot unifié dédié à l'intégration des composants logiciel/matériel

La bibliothèque de composants est indispensable pour le processus d'intégration. Elle contient des composants configurables (paramétrables) utilisés pour la composition. Ces composants possèdent des granularités différentes : ils peuvent être des composants d'interface de base ou des sous-systèmes complexes. Chaque composant communique avec l'environnement extérieur (le reste du système) via une interface de communication qui encapsule une liste de services fournis/requis par ce composant.

Le processus d'intégration de composants se compose de cinq étapes élémentaires :

- ***l'étape d'analyse* :** cette étape permet l'extraction d'information relative au système à partir de l'architecture abstraite. Par exemple, les informations telles que les types de modules, leur niveau d'abstraction, leur interface de communication (services requis et protocoles), les types de données et leur taille, les priorités des tâches, etc. Ceci permet de guider le processus d'intégration au cours des étapes suivantes ;
- ***l'étape de sélection* :** cette étape localise, compare, et sélectionne les composants qui fournissent les services requis par le système à partir d'une bibliothèque. Ceci est fait selon les informations de conception extraites précédemment. Cette étape doit être exécutée récursivement jusqu'à ce qu'un ensemble convenable (optimal) de composants qui fournissent tous les services requis soit identifié ;
- ***l'étape de spécialisation* :** cette étape permet la personnalisation des composants, sélectionnés au cours de l'étape précédente, pour satisfaire les exigences des services à fournir (par exemple : les protocoles, les types de données, la taille du bus, etc.). Elle permet aussi de vérifier la compatibilité de ces composants. Les valeurs des

paramètres finaux sont stockées dans des fichiers de configuration utilisés pour la génération automatique au cours des étapes suivantes ;

- ***l'étape de génération*** **:** cette étape a besoin de deux types de fichiers : (1) les fichiers contenant le macro code des composants configurables et (2) les fichiers contenant les paramètres de configuration. Elle permet de générer des fichiers contenant le code source spécialisé correspondant au comportement de chaque composant sélectionné ;
- ***l'étape d'assemblage*** **:** au cours de cette étape les composants produits par les étapes précédentes sont assemblés pour fournir une architecture complète et raffinée.

4.2.2. Intégration au niveau système et au niveau interface

La Figure 28 montre un flot de conception des systèmes hétérogènes embarqués basé sur l'intégration de composants. Ce flot prend en entrée une architecture abstraite du système. Les éléments principaux du flot sont constitués d'un ensemble de générateurs ((a), (b), (c), (d) et (e)) et d'un ensemble de composants logiciel/matériel de différentes granularités (stockés dans des bibliothèques différentes). Chaque générateur utilise une instance du flot de composition unifié présenté dans la section précédente. En plus, étant donné que les étapes définis par ce flot unifié sont toujours les mêmes, on peut concevoir une réutilisation des modules logiciels qui implémentent les étapes 2, 3, et 4. En effet, les étapes 1 et 5 restent spécifiques à chaque générateur. Selon le niveau d'intégration, les différents générateurs sont classés en deux groupes :

- le groupe de générateurs {(a), (b) et (c)} : il permet l'intégration des composants au niveau interface ;
- le groupe de générateurs {(d) et (e)} : il permet l'intégration des composants au niveau système.

Selon le niveau d'intégration, les modèles d'architectures produits par ce flot sont aussi classés en deux groupes :

- au niveau interface : il s'agit du groupe d'architectures qui représente les sous-systèmes d'interface (adaptateurs logiciels, matériels et fonctionnels) ;
- au niveau système : il s'agit du groupe d'architectures SoC qui inclut deux modèles d'architecture, le premier pour la synthèse et le second pour la co-simulation.

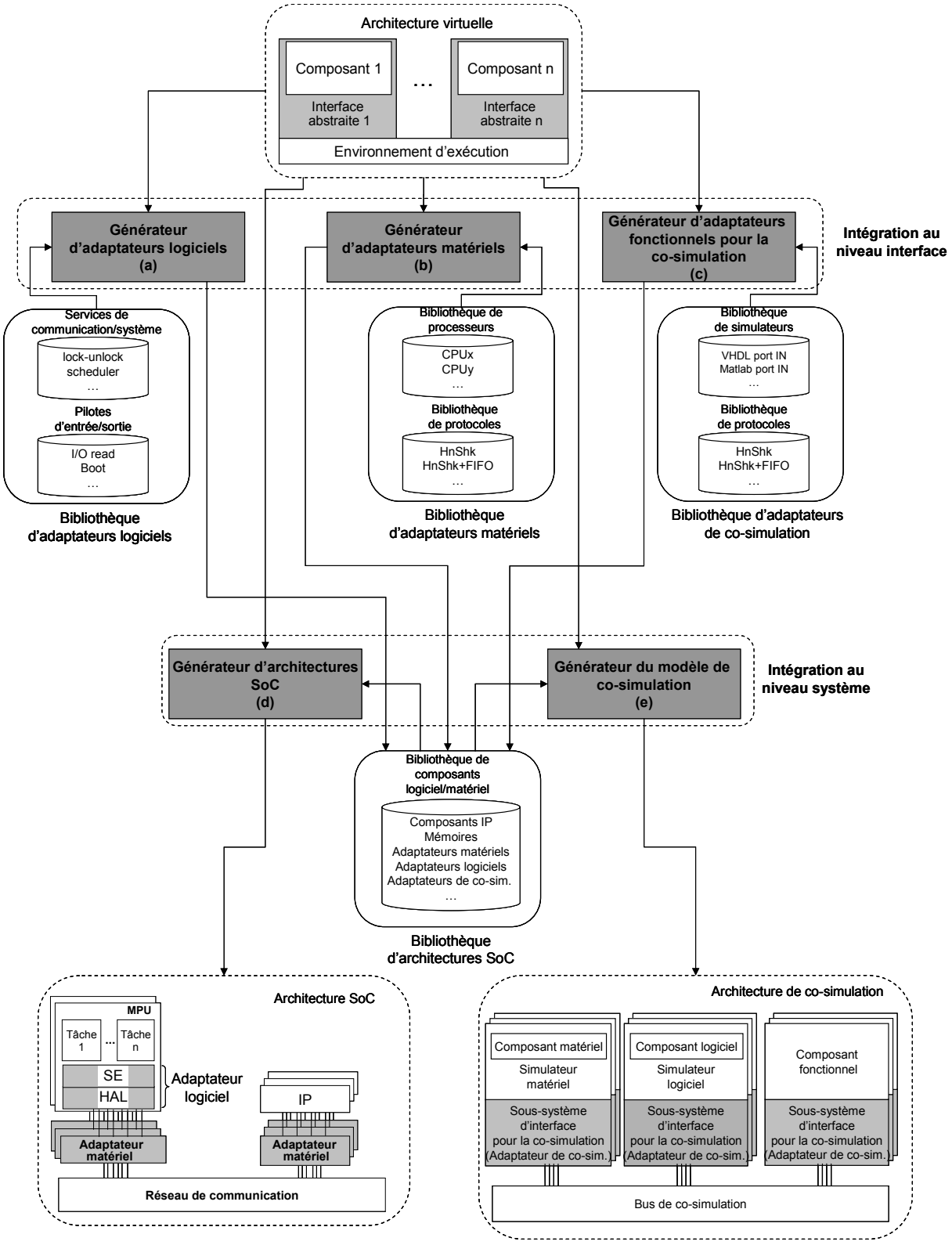

Figure 28. Flot de conception des MPSoC basé sur l'assemblage des composants

4.2.2.1. Intégration au niveau interface

Les générateurs ((a), (b) et (c)) sont des outils de conception permettant l'intégration automatique des composants au niveau interface. Les composants manipulés à ce niveau sont des composants d'interface de base.

Pour la partie logicielle, le flot intègre les composants logiciels (tâches de l'application) qui s'exécutent sur un ou plusieurs processeurs de l'architecture (Figure 28 (a)). Cette intégration nécessite la construction d'un adaptateur logiciel spécifique à chaque processeur de l'architecture choisie. L'adaptateur logiciel — un système d'exploitation et une couche HAL taillés sur mesure — est généré à partir des composants d'interface logiciels de base. Son rôle est de permettre aux composants logiciels de l'application d'accéder à la partie matérielle du système. La compilation des composants logiciels de l'application avec le système d'exploitation généré permet d'avoir un logiciel exécutable. Ce dernier est spécifique à une architecture locale de chaque processeur de l'architecture du système.

Pour la partie matérielle, le flot intègre les composants IP et les architectures locales de chaque processeur de l'architecture (Figure 28 (b)). Cette intégration nécessite la construction d'un adaptateur matériel spécifique pour chaque architecture locale et pour chaque composant IP de l'architecture choisie. Un adaptateur matériel est généré à partir des composants d'interface matériels de base. Le rôle des adaptateurs matériels est d'adapter les architectures locales de chaque processeur et les composant IP au réseau de communication.

Pour la partie fonctionnelle, le flot intègre les composants IP (logiciels, matériels, et fonctionnels) de l'architecture (Figure 28 (c)). Cette intégration nécessite la construction d'un adaptateur fonctionnel spécifique pour chaque composant IP choisi. Un adaptateur fonctionnel est généré à partir des composants d'interface fonctionnels de base. Le rôle des adaptateurs fonctionnels est d'adapter les interfaces de communication des différents composants IP et les interfaces de communication des différents simulateurs au bus de co-simulation.

4.2.2.2. Intégration au niveau système

Les générateurs (d) et (e) sont des outils de conception permettant l'intégration automatique des composants au niveau système. Les composants manipulés à ce niveau sont des composants IP (logiciels, matériels et fonctionnels), des architectures locales spécifiques à des processeurs donnés, des sous-systèmes d'interface (logiciels, matériels et fonctionnels) et un réseau de communication. Les architectures locales et les sous-systèmes d'interface sont

manipulés, à ce niveau, en tant que composants. Les différents composants sont stockés dans la bibliothèque respective de chaque générateur.

Le générateur (d) permet la génération d'un modèle d'architecture SoC pour la synthèse. Ce modèle inclut des composants IP matériels, une architecture locale spécifique à chaque processeur, un sous-système d'interface logiciel (adaptateur logiciel) spécifique à chaque architecture locale, un sous-système d'interface matériel (adaptateur matériel) spécifique à chaque architecture locale/composant IP matériel et un réseau de communication.

Le générateur (e) permet la génération d'un modèle d'architecture SoC pour la co-simulation. Ce modèle est indispensable pour la validation globale du système. Il inclut des modèle de simulateurs spécifiques à chaque composant logiciel/matériel, des composants fonctionnels (ce sont des composants dont les concepteurs n'ont pas encore – au moment de la simulation – décider de leur implémentation en logiciel ou en matériel), des sous-systèmes d'interface fonctionnels (adaptateurs pour la co-simulation) spécifiques à chaque simulateur/composant fonctionnel et un bus de co-simulation.

4.3. Utilisation de ROSES pour l'intégration des composants

Cette section propose une évolution du flot de conception ROSES afin de suivre les différents concepts présentés dans les sections précédentes de ce chapitre. Pour ce faire, cette section est organisée en trois sous-sections : la première sous-section décrit les modifications au niveau de la structure des bibliothèques de composants, la deuxième sous-section décrit les modifications pour l'intégration au niveau interface, et la dernière décrit les modifications pour l'intégration au niveau système.

4.3.1. Structure des bibliothèques de composants

D'après la méthodologie unifiée, qui ressemble le plus à la méthodologie de l'outil ASOG, la sélection des composants se base sur le principe de services requis/fournis. En fait, un composant communique avec son environnement via une interface qui inclut l'ensemble des services requis/fournis par ce composant. Le composant lui-même propose une implémentation de l'ensemble des services fournis. Cette implémentation peut être en matériel, dans le cas d'un composant matériel. Elle peut être en logiciel, dans le cas d'un composant logiciel. Par conséquent, deux types d'implémentation (en logiciel et en matériel) sont possibles pour un ensemble de services fournis par un même composant.

Pour permettre la sélection d'un composant selon l'ensemble de ses services requis/fournis, nous proposons l'unification de la structure des différentes bibliothèques

utilisées par les outils de ROSES. Pour ce faire, nous proposons l'utilisation de la structure de la bibliothèque de système d'exploitation (présentée dans la section 3.6.1.3 du Chapitre 3) : la bibliothèque des adaptateurs matériels ainsi que la bibliothèque des adaptateurs fonctionnels doivent suivre la même structure proposée par la bibliothèque du générateur de systèmes d'exploitation (c'est-à-dire un graphe de dépendance entre éléments, services et implémentations). Pour pouvoir raffiner le modèle COLIF au cours du processus de conception, nous proposons d'associer à chaque composant de la bibliothèque un fichier qui représente sa description en COLIF. Des fichiers décrits par un macro langage sont aussi associés à chaque composant, dit configurable. Après leur configuration, ces fichiers constituent l'implémentation (ou encore comportement) final du composant.

L'unification de la structure des bibliothèques permet la sélection d'un composant selon ses services requis/fournis. En plus, différentes implémentations en logiciel ou en matériel du même composant peuvent être utilisées. Ceci permet une exploration plus rapide de l'espace des solutions architecturales pour la communication entre logiciel et matériel, c'est-à-dire générer automatiquement la partie qui sera implémentée en logiciel et la partie qui sera implémentée en matériel de la couche de communication.

Les sections suivantes présentent les deux niveaux d'intégration définis par la méthodologie unifiée. Ces deux niveaux sépare clairement l'intégration des composants d'interface de base et l'intégration des composants complexes (adaptateurs, composants logiciel/matériel, etc.) en vue d'une stratégie de réutilisation des adaptateurs au niveau système. Actuellement, les outils de ROSES permettent une intégration des composants d'interface pour générer les adaptateurs et une intégration de ces derniers dans le système globale pour générer des architectures SoC, sans prévoir une éventuelle réutilisation des adaptateurs générés.

4.3.2. Intégration au niveau interface

Au niveau interface, ROSES permet actuellement la génération automatique des différents sous-systèmes d'interface. Par contre, ces sous-systèmes d'interface ne sont pas ajoutés aux différentes bibliothèques de composants. Pour permettre une réutilisation de ces sous-systèmes d'interface au niveau système, nous proposons : (1) l'identification d'un ensemble de services fournis/requis spécifique à chaque sous-système d'interface, (2) l'ajout de chaque sous-système d'interface à la bibliothèque unifiée en tant que composant qui fournis/requiert des services et (3) l'identification d'un ensemble de propriétés, de chaque sous-système d'interface, qui peut être paramétrable (par exemple, les priorités des tâches

pour le sous-système d'interface logiciel). Ceci permet une configuration de chaque sous-système d'interface selon les exigences du système.

Actuellement, ROSES utilise deux architectures locales qui s'articule autour de deux instances de processeurs (ARM7 et 68000 de Motorola). Les composants de ces architectures locales (mémoires, périphériques, bus interne, etc.) sont assemblés et décrits manuellement sous forme de modèles COLIF. Pour permettre une réutilisation plus effective des composants et une génération automatique de différentes architectures locales, nous proposons : (1) la construction d'un générateur qui produit automatiquement des architectures locales selon des paramètres annotés par l'architecture abstraite du système, (2) le respect des étapes du flot unifié lors de la construction de ce générateur, (3) l'ajout des composants indispensables pour la construction d'une architecture locale à la bibliothèque unifiée en respectant la structure de cette dernière (c'est-à-dire un graphe de dépendance entre éléments, services et implémentations). Les avantages d'un tel générateur sont : (1) avoir la possibilité d'explorer plus rapidement plusieurs solutions architecturales en utilisant différentes architectures locales compatibles avec le reste de l'architecture globale du système, et (2) libérer le concepteur d'un travail manuel – qui consiste à écrire manuellement les descriptions des architectures locales en COLIF – très fastidieux et source d'erreur.

4.3.3. Intégration au niveau système

Au niveau système, ROSES permet actuellement une intégration automatique des différents adaptateurs générés au niveau interface dans des modèles d'architectures SoC (pour la synthèse ou pour la co-simulation). Cette intégration est assurée par les mêmes générateurs utilisés au niveau interface. En plus, la réutilisation des adaptateurs au niveau système n'est pas encore supportée. Par conséquent, nous proposons la construction de deux générateurs : le premier permet la génération de modèles d'architectures SoC pour la synthèse et le deuxième permet la génération de modèles de co-simulation. Ces deux générateurs suivent les mêmes étapes du flot unifié et manipulent les différents sous-systèmes d'interface en tant que composants qui fournissent/requiert des services. Les entrées de chaque générateur sont : (1) une architecture abstraite du système (l'architecture virtuelle) et (2) une bibliothèque de composants. Dans le cas où le générateur n'arrive pas à trouver un adaptateur dans la bibliothèque de composants, ce dernier peut être généré automatiquement en utilisant les générateurs au niveau interface.

4.4. Modèle d'intégration d'outils

4.4.1. Principes

Le deuxième objectif encadrant les travaux de cette thèse est la construction d'un environnement ouvert à l'intégration d'outils pour la conception des systèmes hétérogènes embarqués. Pour ce faire, nous proposons l'utilisation d'un format intermédiaire dédié à la représentation de ces systèmes comme un modèle de référence pour tous les outils participant au processus de conception (Figure 29). Ce format intermédiaire joue le rôle d'un standard ou un vocabulaire commun à travers lequel les différents outils peuvent communiquer et échanger des données de conception. L'échange de données se fait par des processus de traduction. Ces processus encapsulent des opérations de type « import/export » entre deux modèles de représentation de données différents : le modèle interne de représentation de données de chaque outil externe et le format intermédiaire. Grâce à l'utilisation de ce format intermédiaire, le nombre de traducteurs nécessaires à l'échange de données entre les outils est réduit. Il est linéairement proportionnel au nombre d'outils intégrés. Cependant, une vérification du bon déroulement des échanges de données entre chaque outil externe intégré et le format intermédiaire est toujours nécessaire pour garantir une interopérabilité entre les outils.

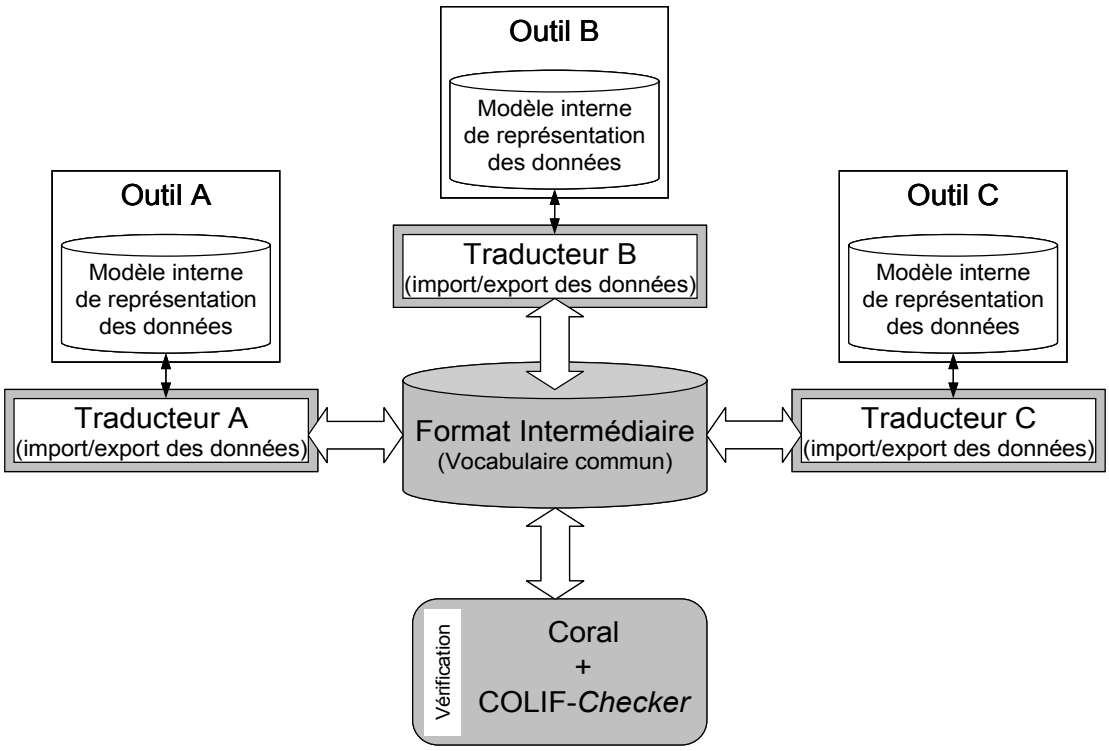

Figure 29. Modèle d'intégration d'outils autour d'un format intermédiaire

4.4.2. Le format intermédiaire

Le format intermédiaire utilisé doit être neutre, c'est-à-dire indépendant des outils, des méthodes et des processus de conception. Il doit être aussi suffisamment flexible pour maintenir les sémantiques entre les outils, garantir l'intégrité du système, préserver la structure du système et les décisions fondamentales dont dépend sa performance.

Le format intermédiaire COLIF, développé par de groupe SLS et présenté dans la section 3.5 du chapitre précédent, rempli les caractéristiques d'un standard d'échange de données. Dans ce chapitre, COLIF est utilisé comme un format de référence flexible et ouvert à l'intégration d'outils de conception externes dans l'environnement ROSES.

4.4.3. Le processus de traduction

Généralement, un traducteur spécifique à chaque outil est construit pour achever l'étape d'intégration d'outils dans un environnement de conception donné sans l'intervention des concepteurs. Dans le modèle d'intégration d'outils que nous proposons, les avantages de la construction des traducteurs sont : (1) assurer un lien automatique entre les outils de conception d'une part et le modèle COLIF d'autre part, (2) pour les systèmes complexes de taille importante, ce processus est moins coûteux qu'un travail manuel pour traduire les données et (3) libérer les concepteurs ou les intégrateurs d'un travail manuel très fastidieux et qui est une source d'erreurs dans le processus de conception. Ceci permet aussi de se concentrer sur d'autres étapes de conception liées par exemple à l'exploration d'architectures. Cependant, la construction des traducteurs est une tâche importante. En effet, elle nécessite une maîtrise des modèles en entrée/sortie du traducteur, un travail de modélisation pour identifier les correspondances entre les deux modèles et proposer les restrictions nécessaires, et de bonnes compétences en programmation. Le processus de traduction est très dépendant de l'outil à intégrer :

- si le code source de l'outil est disponible alors le principe d'intégration de type « *white-box* » est utilisé (voir la section 2.6.1 du Chapitre 2). Dans ce cas, la traduction consiste à développer des bibliothèques qui étendent le code source de l'outil pour rejoindre le modèle désiré. Par exemple dans le cas de SystemC [Sys 00], une extension de son code source est faite (VADeL – Virtual Architecture Description Language) pour mettre en évidence les concepts de module virtuel, port virtuel, canal virtuel et les différentes annotations (paramètres) requises par l'architecture virtuelle et le modèle COLIF ;

- si le code source de l'outil n'est pas disponible, alors on procède plutôt à une intégration de type « *black-box* ». Dans ce cas l'outil est manipulé en tant que boite noir. Par exemple dans le cas de l'outil VCC [Cad 01], les données concernant le modèle produit par cet outil sont stockées dans une base de données disponible. Cette base de données se repose sur un répertoire de travail contenant des fichiers qu'on peut visualiser à l'aide d'un simple éditeur de texte. L'intégration de VCC en utilisant notre modèle d'intégration d'outils se fait à travers un processus de traduction qui permet l'extraction des données de conception à partir de la base de données de VCC, de traiter ces données et enfin de produire le modèle COLIF.

Cependant, à cause de sa sémantique extensible et dynamique (basée sur les paramètres attachés aux modules, ports et canaux), le modèle COLIF est différent d'un langage dans le sens où il ne peut pas être exécuté. En plus, COLIF obéit à une syntaxe uniforme afin de représenter les systèmes hétérogènes décrits à différents niveaux d'abstraction et/ou utilisant différents langages de spécification. La bonne formation du modèle COLIF est indispensable pour garantir une interopérabilité entre les outils. Dans notre modèle d'intégration d'outils, le modèle COLIF est obtenu suite à un processus de traduction automatique assuré par un traducteur spécifique à chaque outil. Ces traducteurs peuvent être construits par des développeurs différents. Le problème réside dans le fait que les outils de ROSES admettent que le modèle COLIF est bien formé et contient toutes les données indispensables pour leur bon fonctionnement, ce qui n'est pas toujours évident suite à un processus de traduction automatique. Notre contribution pour résoudre ce problème est de mettre en place un processus de vérification.

4.5. Vérification des échanges entre outils

Notre modèle d'intégration d'outils repose sur le fait que le format intermédiaire COLIF est un modèle d'échange entre tous les outils participant au processus de conception. Les outils, accédant directement au modèle COLIF tels que les outils de ROSES, admettent qu'il est cohérent et complet. Par conséquent, certaines normes de bonne formation doivent être respectées au moment de la traduction telles que la syntaxe et la sémantique. La vérification de la bonne formation du modèle COLIF revient à vérifier un ensemble de règles syntaxiques et sémantiques spécifiques à ce modèle et aux outils de ROSES. Pour ce faire, un outil a été développé pour assurer une vérification automatique de la syntaxe et la sémantique du modèle COLIF. Ce processus de vérification permet d'éviter la perte d'un temps de conception important causée par une mauvaise génération du modèle COLIF (par exemple, non respect

de la syntaxe ou/et manque de données nécessaires pour le bon fonctionnement du processus de conception).

4.5.1. Architecture de l'outil COLIF-*Checker*

La Figure 30 montre l'architecture de l'outil COLIF-*Checker* développé au cours des travaux de cette thèse. Cet outil permet de vérifier d'une façon automatique la bonne formation du modèle COLIF produit par les outils externes. Cette vérification est faite en deux étapes : la première étape consiste à vérifier la structure syntaxique de COLIF. La deuxième étape consiste à vérifier la sémantique qui est spécifique à COLIF et à chaque outil de ROSES. Les entrées de l'outil sont :

- le modèle COLIF produit par les outils externes, sous forme d'un fichier XML ;
- une base de données contenant un ensemble de règles syntaxiques et sémantiques spécifiques à COLIF et à chaque outil de ROSES. Les constructeurs et/ou les intégrateurs d'outils peuvent enrichir cette base de données avec leurs propres règles. Un langage de spécification de règles a aussi été développé.

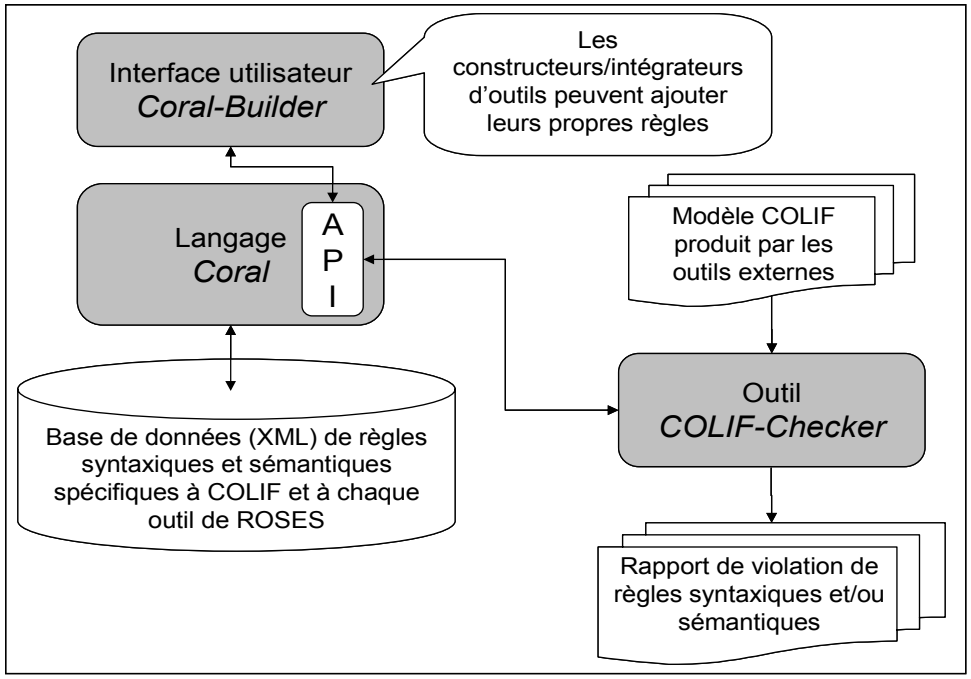

Figure 30. Architecture de l'outil COLIF-*Checker*

L'outil COLIF-*Checker* produit un fichier texte qui représente un rapport de violation des règles syntaxiques et sémantiques. Ce rapport fournit aux concepteurs des informations concernant l'ensemble des règles violées, ce qui leur permet de localiser très rapidement

l'origine de l'erreur dans le processus de la traduction. Dans le cas où le modèle COLIF est bien formé, le rapport confirme la vérification et la validation de toutes les règles.

4.5.2. Le langage Coral

Coral est un langage pour la spécification des règles syntaxiques et sémantiques spécifiques au modèle COLIF et aux outils de ROSES. La construction de ce langage a été abordée par couches (Figure 31). Ces couches sont :

- une couche d'interfaçage du langage Coral (en langage C++) avec l'outil de construction des règles (Coral-*Builder*) et l'outil de vérification de ces derniers (COLIF-*Checker*): API Coral. Elle fournit un jeu de primitives permettant de découpler l'utilisation par un outil des structures de données (règles), de l'implémentation de ces structures et de leur gestion. ;
- une couche d'implémentation du langage Coral. Elle définit les structures des données permettant la représentation des règles ;
- un langage et son API permettant la définition de type des structures de données et l'utilisation de ces structures : Middle et API Middle. La conception de ce langage est une contribution de [Gau 01], il a aussi été utilisé pour construire COLIF ;
- le langage XML et son analyseur (*parser*).

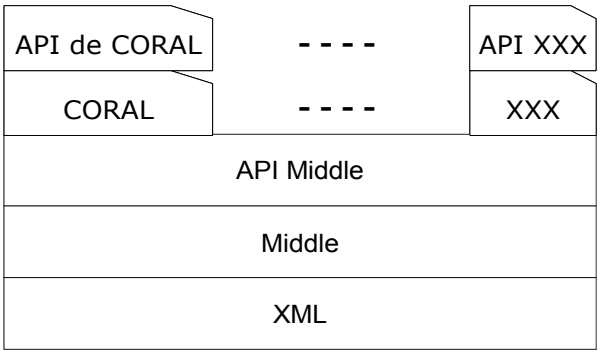

Figure 31. Couches logicielles du langage Coral

La couche Middle (*Mark-up Internal Data Description Language Extension*) est une grammaire dérivée de XML et offre le pouvoir d'exprimer des structures de données complexes pour les différents objets du langage Coral.

4.5.2.1. Les objets de base de Coral

La construction d'une règle en Coral consiste à créer un ensemble d'objets. Ces objets sont prédéfinis par Coral et leur création se fait à partir d'un ensemble de classes implémentées par ce dernier. Chaque objet créé encapsule un ensemble d'informations bien

particulières qui servent à décrire une partie de la règle. Une fois que tous ces objets, nécessaires pour la construction d'une règle, sont créés la description de la règle est dite complète. Par conséquence, elle peut être sauvegardée dans la base des règles afin d'être manipulée par la suite selon des scénarios de vérification par l'outil COLIF-*Checker*. La Figure 32 illustre le digramme UML des classes de Coral.

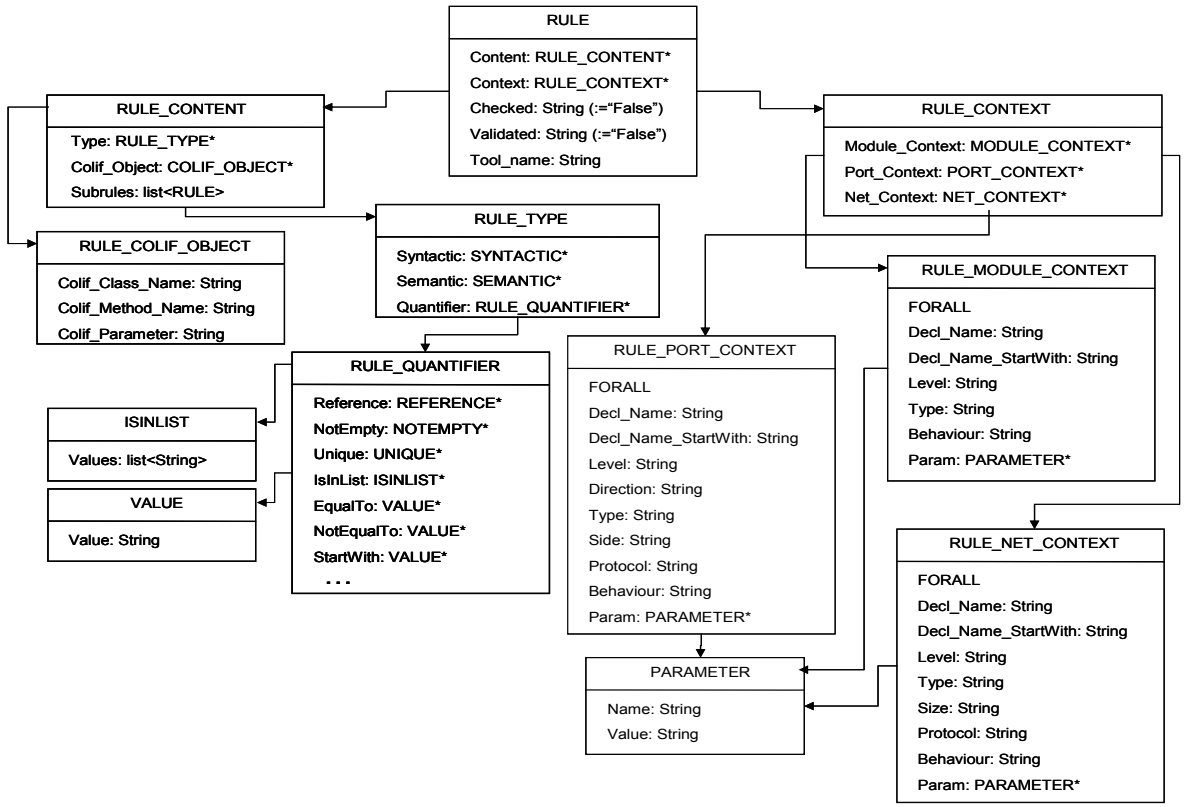

Figure 32. Diagramme UML des classes Coral

La sémantique des classes principales de Coral est brièvement introduite par le Tableau 3.

Classe	
Responsabilité	
Membre	Signification
RULE	
Modélise une règle	
Description	Brève description de la règle
Content	Référence vers une description du contenu de la règle
Context	Référence vers une description du contexte de la règle
Checked	Drapeau indiquant si une règle est vérifiée ou pas
Validated	Drapeau indiquant si une règle est validée ou pas
tool_name	Le nom de l'outil auquel appartient la règle
RULE_CONTENT	
Modélise le contenu d'une règle	

description	Brève description du contenu de la règle
Type	Référence vers le type de la règle
colif_object	Référence vers un objet COLIF
colif_parameter	Le nom d'un paramètre COLIF
subrules	Liste de règles qui doivent être vérifiées et validées avant la vérification de la règle courante

RULE_CONTEXT	
Modélise le contexte d'exécution d'une règle	
description	Brève description du contexte de la règle
module_context	Référence vers un contexte de module
port_context	Référence vers un contexte de port
net_context	Référence vers un contexte de canal

RULE_TYPE	
Modélise le type d'une règle	
description	Brève description du type de la règle
syntactic	Indique qu'il s'agit d'une règle syntaxique
semantic	Indique qu'il s'agit d'une règle sémantique
quantifier	Référence vers un quantificateur de la règle

RULE_COLIF_OBJECT	
Modélise un objet COLIF	
description	Brève description de l'objet COLIF sur lequel la règle est appliquée
colif_class_name	Le nom de la classe qui représente le type d'un objet COLIF sur lequel la règle est appliquée
colif_method_name	Le nom de la méthode implémentée par la classe sous-jacente et utilisée pour récupérer les références d'autres objets COLIF. Généralement c'est une méthode de type get*Property*.

RULE_QUANTIFIER	
Modélise la liste des quantificateurs possibles pour une règle	
description	Brève description du quantificateur de la règle
Reference	La règle vérifie qu'une référence vers un autre objet COLIF existe (c'est-à-dire qu'elle est créée et instanciée)
notEmpty	La règle vérifie que le contenu d'une liste de référence est différent du vide
unique	La règle vérifie qu'un objet COLIF est unique. Pour l'instant ce quantificateur est construit spécialement pour tester l'unicité de l'objet ColifRoot.
isInList	La règle vérifie l'existence d'une valeur donnée dans une liste de valeurs possibles
EqualTo	La règle vérifie l'égalité entre une entité spécifiée par la règle et une autre d'un objet COLIF
NotEqualTo	La règle vérifie l'inégalité entre une entité spécifiée par la règle et une autre d'un objet COLIF
StartWith	La règle vérifie si une chaîne de caractères commence par une sous chaîne indiquée par ce quantificateur

Tableau 3. La sémantique des classes Coral

4.5.2.2. Syntaxe des règles

La syntaxe d'une règle en Coral est décrite par la Figure 33. Une règle possède un contenu, un contexte, et elle est spécifique à un outil bien particulier.

Le contenu d'une règle est composé par :

- un objet COLIF bien déterminé : il peut être soit l'un des objets définis par le modèle COLIF, soit un objet de type paramètre. Cette séparation est choisie principalement pour pouvoir décrire des règles syntaxiques spécifiques aux objets COLIF et des règles sémantiques spécifiques aux paramètres. Un objet COLIF est identifié par son type. Chaque propriété (*Property*) d'un objet COLIF peut être récupérée en utilisant des méthodes de type « get*Property*…». La valeur d'une propriété d'un objet COLIF peut être soit une chaîne de caractères, ou une référence vers un autre objet COLIF, soit une référence vers une liste d'objets COLIF. L'indication de ce type de méthodes dans la spécification d'une règle permet de connaître la nature de l'information recherchée. Un paramètre est identifié par le type *PARAMETER* et par un nom ;
- un quantificateur : il détermine comment la règle doit agir. Un ensemble de quantificateurs possibles a été aussi implémenté (par exemple « *reference* » est un quantificateur qui indique que la règle est spécialement construite pour vérifier l'existence d'une référence vers un objet COLIF) ;
- un ensemble de règles enfants : c'est un ensemble de règles dont dépend la règle courante (règle mère). Par conséquent, lors de la vérification de la règle mère il faut s'assurer de la validation des règles enfants. Dans le cas où ces dernières ne sont pas tous validées, la règle mère ne peut pas être vérifiée. La vérification et la validation des règles par rapport à un modèle COLIF donné est accompli en utilisant l'outil COLIF-*Cheker*.

Le contexte d'une règle représente son contexte d'exécution. Puisque une règle est spécifique au modèle COLIF, son contexte d'exécution est alors déterminé par les objets module, port, et net. Par conséquent, ce contexte peut être soit par rapport à un objet « module » (*Module_context*), soit par rapport à une combinaison d'un objet « module » et d'un objet « port » (*Port_context*) ou d'un objet « net » (*Net_contexte*). Le contexte des différents objets est déterminé par la donnée d'une information qui caractérise l'objet (par exemple, dans le cas d'un contexte de module le mot clé *FORALL* caractérise tous les modules qui constituent le système : la règle sera alors appliquée à tous les modules sans aucune exception. La donnée d'un type de module Type= «*XXX*» et d'un nom de port DeclName= «*YYY*» signifie que la règle sera appliquée à tous les ports qui possèdent le nom

«*YYY*» et qui appartiennent aux modules de type «*XXX*». Dans ce cas nous avons réduit considérablement le contexte d'exécution de la règle).

Les règles supportées par Coral et par l'outil COLIF-*Checker* sont spécifiques à l'ensemble des outils offert par l'environnement ROSES : COLIF, le générateur d'adaptateurs matériels (ASAG), le générateur d'adaptateurs logiciels (ASOG), et le générateur d'adaptateurs fonctionnels pour la co-simulation (CosimX). L'indication du nom de l'outil au niveau de la spécification de la règle permet à un outil donné d'exécuter ou de vérifier l'ensemble de règles qui lui sont spécifiques.

<Rule> ::= <Rule_Content> <Rule_Context> <Tool>

<Rule_content> ::= <Rule_Type> <Colif_Object> <{Sub_Rules}>

<Rule_Type> ::= <{SYNTACTIC | SEMANTIC}><Rule_Quantifier>

<Colif_Object> ::= <ColifObjectType, [MethodName, [ParameterName]]>

<Rule_Quantifier> ::= {Reference | NotEmpty | Unique | isInList | EqualTo | NotEqualTo | StartWith | …}

<Sub_Rules> ::= {Null | Rule, […]}

<Rule_Context> ::= <Module_Context>< Port_Context | Net_Context | Null>

<Module_Context> :: = {FORALL | DeclName | DeclNameStartWith | Level | Type | Behaviour | Param}

<Port_Context> :: = {FORALL | DeclName | DeclNameStartWith | Level | Direction | Type | Side | Protocol | Behaviour | Param}

<Net_Context> :: = {FORALL | DeclName | DeclNameStartWith | Level | Type | Side | Protocol | Behaviour | Param}

<Tool> :: = {COLIF | ASAG | ASOG | CosimX}

Figure 33. Syntaxe des règles décrites en Coral

4.5.2.3. Exemples de règles

Cette section présente quelques règles à titre d'exemple.

La règle ***<R1>::=<SYNTACTIC><Unique><ROOT><COLIF>*** obéit à la syntaxe des règles définie par Coral. Cette règle est syntaxique et exprime le fait qu'un seul et unique objet de type *ColifRoot* doit exister dans le modèle COLIF (Par analogie, un document XML syntaxiquement bien formé doit avoir un seul nœud racine. L'objet de type *ColifRoot* représente le module *sommet* de la hiérarchie). Pour ce faire la règle indique le type de l'objet (ici *ROOT*), le quantificateur (ici <*Unique*>) permettant d'exprimer le fait qu'on cherche à prouver l'unicité de ce type d'objet. Cette règle est une règle syntaxique spécifique au modèle COLIF et elle est la première règle à vérifier, par défaut, par l'outil COLIF-*Checker*.

Remarque : les types des objets COLIF sont prédéfinis au niveau de la couche Middle. Le Tableau 4 illustre les différents objets COLIF et leur correspondance en Middle. Le type

prédéfini dans Middle peut être obtenu à partir des objets COLIF en utilisant une méthode (***isA( )***) implémentée au niveau de chaque classe COLIF.

Objet COLIF	Type prédéfini dans Middle
ColifModule	MODULE
ColifModuleEntity	MODULE_ENTITY
ColifModuleContent	MODULE_CONTENT
ColifModuleInstance	MODULE_INSTANCE
ColifModuleDecl	MODULE_DECL
ColifModuleType	MODULE_TYPE
ColifModuleBehaviour	MODULE_BEHAVIOUR
ColifRoot	ROOT
ColifPort	PORT
ColifPortEntity	PORT_ENTITY
ColifPortContent	PORT_CONTENT
ColifPortInstance	PORT_INSTANCE
ColifPortDecl	PORT_DECL
ColifPortRef	PORT_REF
ColifPortRefDecl	PORT_REF_DECL
ColifPortType	PORT_TYPE
ColifPortBehaviour	PORT_BEHAVIOUR
ColifNet	NET
ColifNetEntity	NET_ENTITY
ColifNetContent	NET_CONTENT
ColifNetInstance	NET_INSTANCE
ColifNetDecl	NET_DECL
ColifNetType	NET_TYPE
ColifNetBehaviour	NET_BEHAVIOUR
ColifSource	SOURCE
ColifCpu	CPU
ColifBlackbox	BLACKBOX
ColifHardware	HARDWARE
ColifSoftware	SOFTWARE
ColifCompound	COMPOUND
ColifParameter	PARAMETER
ColifParamDefinition	PARAM_DEFINITION

Tableau 4. Les types d'objets COLIF et leur équivalent en Middle

Dans l'environnement ROSES les niveaux d'abstraction de la communication sont : le niveau système (*System Level* – SL), le niveau message (*Message Level* – ML), le niveau transaction (*Driver Level* – DL), et le niveau RTL. La règle suivante :

<R2> ::= < SEMANTIC > < isInList {"", "SL", "ML", "DL", "RTL"} > < MODULE, GET_LEVEL> < moduleCxt (FORALL) > < COLIF > obéit à la syntaxe des règles définie par Coral. Cette règle est sémantique et exprime le fait que, en COLIF, le niveau d'abstraction d'un module doit être l'une des valeurs situés dans la liste attachée au quantificateur

(*isInList*). Pour ce faire la règle indique le quantificateur (ici <*isInList*>) permettant de fixer une liste de valeurs possibles d'un niveau d'abstraction (les valeurs possibles sont : une chaîne de caractères vide, "SL", "ML", "DL", et "RTL"), le type de l'objet (*MODULE*), le nom de la méthode permettant la récupération du niveau d'abstraction d'un objet module/port/net (*GET_LEVEL*). Le contexte d'exécution de la règle est fixé à la valeur (*FORALL*) pour couvrir tous les modules du modèle COLIF. Lors de l'application de cette règle, tous les objets de type *MODULE* sont récupérés à partir de l'arbre d'objets COLIF. Ensuite, le niveau d'abstraction de chaque module est récupéré grâce à une méthode de type *GET_LEVEL* et la valeur est comparée à celles proposées dans la liste. La règle est dite validée dès que les niveaux d'abstraction de tous les modules correspondent à ceux qui se trouvent dans la liste. Cette règle est une règle sémantique spécifique aux outils de ROSES.

La règle :

<R3> ::= < SEMANTIC > < NotEqualTo {" "} > < PARAMETER, GET_VALUE, SoftPortType > < moduleCxt (Type (software)), portCxt (FORALL) > < ASOG > obéit à la syntaxe des règles définie par Coral. Cette règle est sémantique et exprime le fait que la valeur du paramètre *SoftPortType* attaché aux ports de tous les modules logiciels (tâches logicielles) ne doit pas avoir la valeur *vide* (c'est-à-dire que la valeur doit être spécifiée impérativement). Pour ce faire, la règle indique le type de l'objet et le nom du paramètre (ici *PARAMETER* et *SoftPortType*), le quantificateur (ici <*NotEqualTo*>) permettant d'exprimer le fait qu'on cherche à prouver l'inégalité de la valeur du paramètre à une valeur fixer par l'utilisateur. Cette règle est une règle sémantique spécifique à l'outil de génération d'adaptateurs logiciels ASOG.

4.5.2.4 Limitation de Coral

Le langage Coral est limité à : (1) la description des règles syntaxiques et sémantiques, et (2) un ensemble de quantificateurs prédéfinis.

4.5.3. Les outils utilisant Coral

4.5.3.1. Interface utilisateur : Coral-*Builder*

La couche API offerte par Coral permet aux concepteurs et aux intégrateurs d'outils de manipuler les règles. Un autre moyen est aussi offert à travers une interface utilisateur appelée Coral-*Builder*. Cet outil a été développé essentiellement pour faciliter la manipulation des règles. Coral-*Builder* représente une couche logicielle construite au dessus de la couche API de Coral. Par conséquent, il permet de masquer l'utilisation de l'API de Coral. L'interface

utilisateur Coral-*Builder* est basée sur un interpréteur de commande, inspiré du langage de requête SQL. Les commandes existantes sont :

- la commande (**USE *nom_fichier_coral.xml ;***) : permet la lecture du fichier Coral contenant les règles syntaxiques et sémantiques. Elle permet aussi de charger les structures utilisées pour décrire ces règles en mémoire ;
- la commande (**SHOW RULES ;**) : permet de visualiser, sous forme d'une table, les informations relatives à toutes les règles stockées dans le fichier XML (nom, type, quantificateur, contexte, sous-ensemble de règles, etc.) ;
- la commande (**CREATE {SYNTACTIC | SEMANTIC} RULE '*rule_name*' –q [Quantifier] –o [ColifObjectType] –m [MethodName] [-p [parameterName]] –s [SubRules] –mc [moduleContext] [-pc [portContext] –nc [netContext]] –t [Tool] ;**) : permet de créer une nouvelle règle et de l'insérer dans la liste des règles. Par exemple, pour créer la règle R2 décrite dans la section précédente la commande suivante est utilisée : (CREATE SEMANTIC RULE 'R2' –q isInList (" ", "SL", "ML", "DL", "RTL") –o MODULE –m GET_LEVEL –cm FORALL –t COLIF ;). Pour créer la règle R3, la commande suivante est utilisée : (CREATE SEMANTIC RULE 'R3' –q NotEqualTo(" ") –o PARAMETER –m GET_VALUE –p SoftPortType –mc (Type(software)) –pc FORALL –t ASOG ;) ;
- la commande (**DELETE RULE 'rule_name' ;**) : permet de supprimer une règle de la liste des règles ;
- la commande (**SAVE RULES ;**) : permet de sauvegarder dans un fichier XML la liste des règles ;
- la commande (**HELP ;**) : permet d'afficher le mode d'utilisation des commandes implémentées par Coral-*Builder* ;
- la commande (**EXIT ;**) : permet de quitter l'interpréteur de commande.

Si la syntaxe d'une commande n'est pas respectée, une exception est levée et un message d'erreur s'affiche. Dans le cas contraire, un message indiquant la fin de l'exécution de la commande est affiché.

4.5.3.2. Format de représentation des règles

Les règles sont stockées dans un fichier XML. L'analyseur (*parser*) Middle permet la lecture du fichier XML. Ensuite, il crée une structure en mémoire (arbre) des différents objets permettant la description des différentes règles et retourne un objet de type *CoralParseTree* (équivalent à l'objet *Document* dans le *Document Object Model – DOM* [DOM] créé par le

Consortium *W3C* pour manipuler les documents XML). L'objet de type *CoralParseTree* représente la racine de la structure créée en mémoire et permet un accès aux différents objets de l'arbre. Les différentes opérations (création, suppression, etc.) appliquées sur les règles sont exécutées directement sur la structure en mémoire en utilisant l'API fournit par Coral.

4.5.3.3. L'outil COLIF-*Checker*

L'outil COLIF-C*hecker* est un composant logiciel qui permet d'interpréter un ensemble de règles syntaxiques et sémantiques, spécifiées en utilisant le langage Coral, et de les appliquer sur un modèle COLIF issu d'un processus de traduction automatique. Cet outil prend en entrée :

- le modèle COLIF ;
- un ensemble de règles syntaxiques et sémantiques spécifiques au modèle COLIF et aux outils de ROSES.

La sortie de l'outil dépend de deux cas :

- si l'ensemble de règles à vérifier par rapport au modèle COLIF est complètement validé alors le *Checker* affiche un message indiquant la validation de cet ensemble de règles ;
- dans le cas contraire, c'est-à-dire si l'ensemble de règles à vérifier par rapport au modèle COLIF n'est pas complètement validé, alors le *Checker* génère un rapport de violation de règles. Ce rapport indique les noms des règles violées et les noms (s'il existent) des objets COLIF présentant les origines des violations. Techniquement, le rapport est un fichier texte simple qui peut être visualisé à l'aide d'un simple éditeur de texte.

Les différentes étapes et composants de l'outil COLIF-*Checker* sont :

- ***lecture de la description de l'application* :** la description du système électronique à traiter, vient sous la forme d'un modèle COLIF tel l'exemple de la Figure 17 présenté dans la section 3.5. Lors de cette étape les deux opérations suivantes sont chronologiquement faites : l'analyse de la syntaxe Middle et l'extraction de la définition de COLIF ; et la construction d'un arbre contenant tous les objets définis (instances, déclaration et définitions des modules, ports et nets). Pour des raisons de performances, deux optimisations sont utilisées. L'usage d'un analyseur de syntaxe XML associé au DTD Middle est abandonné au profit d'un analyseur de syntaxe Middle (parseur câblé). En imposant la définition des structures de données COLIF en

début de fichier XML, les deux opérations précédentes sont successivement réalisées en une unique lecture de fichier ;

- ***chargement des règles* :** cette étape est identique à celle de la lecture d'un modèle COLIF. La base des règles se trouve sous forme d'un fichier XML ayant pour définition de document (DTD) le langage Middle. Lors de cette étape, les deux opérations suivantes sont chronologiquement faites : l'analyse de la syntaxe Middle et extraction de la définition de Coral ; et la construction d'un arbre contenant tous les objets définis (Rule, Rule_Context, Rule_Content, etc.) ;
- ***le moteur de vérification* :** ce moteur représente un processus qui commence par la récupération d'une liste de toutes les règles produite par l'étape précédente. Ceci ne veut pas dire que toutes les règles seront vérifiées par le *moteur*. En effet, il est possible de réduire cette liste à la demande de l'utilisateur, par exemple pour la vérification des règles syntaxiques seulement, la vérification des règles sémantiques seulement, la vérification d'un sous-ensemble de règles spécifiques à un outil donnée ou la vérification d'un sous-ensemble de règles indiquées par leur nom. Cependant, dans le cas où aucune restriction sur l'ensemble des règles n'est imposée explicitement par l'utilisateur, le *moteur* prendra en compte toutes les règles se trouvant dans la base.

Dans tous les cas, la première règle à vérifier par le *moteur* est l'unicité de l'objet *ColifRoot* : un modèle COLIF doit avoir un seul objet de type *ColifRoot*. Les autres règles sont vérifiées selon l'ordre de leur apparition dans la liste complète ou réduite.

Dans le cas où une règle possède un sous-ensemble de règles filles, le *moteur* doit s'assurer d'abord de la validation de ce sous-ensemble avant de vérifier la règle mère. Si l'une des règles filles n'est pas validée, le *moteur* ignore la règle mère et passe à la règle suivante dans la liste principale. Pour éviter de vérifier une même règle plusieurs fois, le *moteur* marque chaque règle atteinte au cours du processus de vérification en utilisant un drapeau (*verified* = *true*). En plus, si la règle est validée alors un deuxième marquage est effectué en utilisant un deuxième drapeau (*validated* = *true*).

Le *moteur* est basé sur l'algorithme suivant :

Début_Algorithme
L_R: la liste des règles
Pour R_i **dans** la liste L_R, **faire**
Procédure (R_i)
L_{DR} : la liste des règles *filles* de R_i
Si (L_{DR} n'est pas vide) **faire**
Pour R_j **dans** L_{DR} **faire**
Si (R_j n'est pas vérifiée) **faire** Procedure (R_j)
Sinon Si (R_j est validée) **faire** continue
Sinon sortir de cette boucle
Fin_Pour
Si (toutes les R_j dans L_{DR} sont validées) **faire** vérifier R_i
Sinon vérifier R_i
Fin_Procédure
Fin_Pour
Fin_Algorithme

Lors de la vérification d'une règle feuille dans la hiérarchie, cette règle est marquée « *verified* ». Ensuite le *moteur* extrait les différentes informations relatives à cette dernière : son contenu, son contexte d'exécution. Puis, il applique la règle sur tous les objets COLIF qui vérifient le contexte d'exécution de la règle.

- ***interface utilisateur* :** l'interface utilisateur dédié à l'utilisation de l'outil *Checker* set très simple. Cette interface consiste à exécuter le *Checker* à partir de la ligne de commandes en lui spécifiant quelques arguments :

Checker **–c** *colif_file.xml* **–r** *rule_files.xml* **–v** *report_file.txt* [**-t** toolName] [**–y** RuleType] [**-s** {ruleName, [...]}]

Par exemple, pour vérifier l'ensemble des règles sémantiques spécifiques à l'outil CosimX, il faut taper la commande suivante :

Checker **–c** *vdsl.xml* **–r** *rules.xml* **–v** *reportVDSL.txt* **-t** COSIMX **–y** SEMANTIC

- *limitation de* COLIF-*Checker* : l'outil COLIF-*Checker* est limité par : (1) l'ensemble d'objets du modèle COLIF actuel et (2) l'ensemble des quantificateurs définis actuellement par Coral.

La Figure 34, présente une vue générale de notre modèle dédié à l'intégration d'outils :

- l'entrée du modèle représente les différents outils de conception externes. Généralement, ces outils génèrent des architectures logiciel/matériel prêtes pour la phase de l'implémentation ;
- un traducteur spécifique à chaque outil externe, permet la translation automatique du modèle produit par cet l'outil vers un modèle unique, qui est COLIF. L'avantage

d'utiliser COLIF est que tous les outils peuvent lire et écrire dans le même modèle durant les différentes étapes du processus de conception ;

- COLIF qui représente le modèle d'entrée pour l'environnement ROSES ;
- l'outil COLIF-*Checker* construit spécialement pour la vérification syntaxique et sémantique du modèle COLIF par rapport à un ensemble de règles ;
- l'ensemble d'outils fourni par l'environnement de conception ROSES.

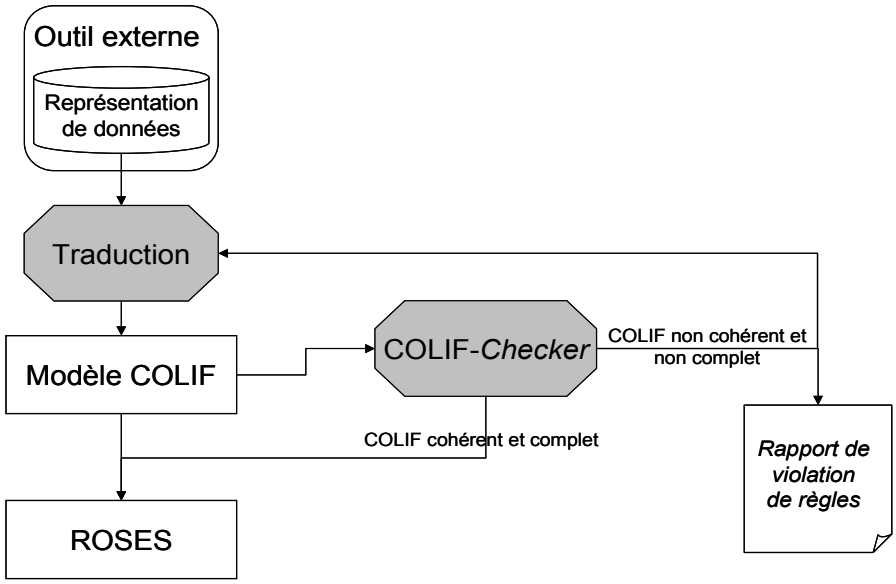

Figure 34. Modèle d'intégration d'outils autour de COLIF

4.6. Conclusion

Dans ce chapitre, nous avons proposé une méthodologie unifiée qui vise l'intégration de composants logiciel/matériel pour la conception et la validation des systèmes hétérogènes embarqués. Cette méthodologie permet une intégration de composants à deux niveaux différents : niveau système et niveau interface. La granularité des composants manipulés varie selon le niveau d'intégration. Au niveau interface, des composants d'interface logiciel/matériel de base sont assemblés pour générer des sous-systèmes d'interface complexes (adaptateurs logiciels, matériels, et fonctionnels pour la co-simulation). Les concepteurs ne sont donc pas obligés d'écrire manuellement les détails d'interface de bas niveau. Au niveau système, ces sous-systèmes d'interface sont manipulés en tant que composants qui permettent l'adaptation des différents composants IP logiciel/matériel au réseau de communication. Nous proposons aussi une réorganisation du flot de conception ROSES afin d'obéir à la méthodologie unifiée sous-jacente. Ainsi, le flot ROSES implémente

une solution généralisée et efficace au problème d'intégration de composants IP et présente une combinaison de caractéristiques unique :

- il implémente une approche unifiée pour l'intégration de composants qui permet l'intégration automatique des composants hétérogènes pour la conception et la validation au niveau système, et il permet aussi d'intégrer automatiquement des composants d'interface élémentaires et homogènes pour la génération des différents adaptateurs au niveau interface ;
- il fournit un environnement capable d'accommoder les composants ayant des interfaces et des granularités différentes et de manipuler des sous-systèmes d'interface complexes en tant que composants ;
- il fournit un processus automatique pour la réutilisation et l'intégration de composants, de façon à ce que les concepteurs se débarrassent d'un travail manuel très fastidieux et source d'erreur lors de la génération d'adaptateurs pour chaque nouvelle solution architecturale. Ces concepteurs peuvent donc se concentrer sur des problèmes de conception plus critiques ayant un rapport avec l'exploration de l'espace architecturale.

Dans une deuxième partie, ce chapitre a montré qu'un flot de conception système complet nécessite le support et la collaboration de différents outils existants dans un même environnement de conception. Une interopérabilité entre ces outils doit être assurée pour améliorer la productivité et garantir l'intégrité du système lors d'échange d'information de conception entre les outils. Pour cela, nous avons proposé un modèle d'intégration d'outils qui s'articule autour du format intermédiaire COLIF. Ce format présente un modèle d'échange de données de conception pour les différents outils : les outils de ROSES d'une part et les outils externes d'autre part. L'intégration d'un outil de conception revient donc à construire un traducteur entre le modèle produit par cet outil et le modèle COLIF. Ce processus de traduction présente plusieurs avantages : il assure un lien automatique entre les modèles fournis par les outils de conception d'une part et le modèle COLIF d'autre part ; il coûte moins qu'un travail manuel pour translater les données vers COLIF surtout pour les systèmes complexes et de taille important ; et il débarrasse les concepteurs ou les intégrateurs d'un travail très fastidieux et source d'erreurs dans le processus de conception. Cependant pour s'assurer de la bonne formation et de la cohérence du modèle COLIF à la suite du processus de traduction, nous avons proposé un outil appelé COLIF-*Checker* qui utilise une base de données de règles syntaxiques et sémantiques extensible et spécifiques au modèle COLIF. Cet outil permet la vérification de ces règles qui sont décrites en utilisant un langage

appelé Coral. Ce langage a été développé au cours des travaux de cette thèse pour pouvoir spécifier des règles syntaxiques et sémantiques spécifiques au modèle COLIF et aux outils de ROSES.

Chapitre 5. Utilisation de l'environnement ROSES pour l'intégration d'outils et de composants

Sommaire

5.1. Intégration de l'outil VCC dans l'environnement ROSES

L'objectif de cette expérimentation est de construire un chemin d'implémentation complet pour les systèmes hétérogènes embarqués, partant d'une spécification fonctionnelle du système jusqu'à obtenir une architecture RTL. Pour construire un tel chemin, nous proposons la combinaison d'un outil d'exploration d'architectures appelé VCC et le flot de conception d'architectures ROSES.

Dans une première étape, nous proposons une analyse des méthodologies de conception de ces deux environnements (VCC et ROSES) pour déterminer les compatibilités et les différences entre eux. Plus précisément, le modèle architectural produit par VCC et le modèle architectural en entrée de ROSES ont été comparés. Cette analyse permet aussi de vérifier que la méthodologie ROSES est appropriée pour construire un flot d'implémentation complet à partir de l'environnement VCC. Dans une deuxième étape, nous proposons une définition d'un lien automatique entre les deux environnements (VCC et ROSES). Ce lien a nécessité la mise au point d'une démarche de conception, de modèles de composants et la construction d'un traducteur. Dans la dernière étape, nous présentons la conception d'un modem VDSL (*Very-High-data-rate Digital Subscriber Line*) en utilisant le chemin d'implémentation sous-jacent. Cette étape consiste à générer et valider un modèle RTL à partir du modèle architectural logiciel/matériel fourni par VCC en utilisant les techniques de composition de composants implémentés par l'environnement ROSES.

Les sections suivantes détaillent les différents travaux faits au cours de cette expérience. L'application à concevoir est également présentée.

5.1.1. L'environnement VCC

5.1.1.1. Méthodologie de conception VCC

L'environnement de conception VCC utilisée pour l'exploration de l'espace des solutions architecturales fait l'analyse des alternatives de partitionnements matériel/logiciel et des raffinements de la communication et permet aussi d'évaluer les performances. Après avoir trouvé une architecture appropriée, VCC utilise des bibliothèques existantes afin de trouver des modèles de composants virtuels pour la simulation de l'architecture.

La méthodologie part d'une spécification fonctionnelle, d'une architecture basée sur des modèles de composants IP (processeurs, IP matériels, mémoires, bus). Outre les composants qui correspondent aux IP effectifs dans le système à concevoir, un modèle d'un système d'exploitation temps réel doit être rattaché à chaque processeur. Dans ce cas-ci, on parle d'une architecture abstraite contenant seulement la topologie de base sans détails sur les

interconnexions. Ensuite, les blocs fonctionnels (fonctions du comportement) sont affectés aux composants architecturaux définissant ainsi le partitionnement logiciel/matériel.

L'affectation est un processus de raffinement continu, pendant lequel des décisions de conception sont prises. Les schémas de communication peuvent être raffinés et prendre en considération des protocoles, des mémoires partagées pour la communication, des mécanismes d'interruption, des types de données concrets et l'adressage de bus et de mémoires. Ils peuvent également inclure des DMA et des composants de mémoire cache.

5.1.1.2. Modèle d'architecture VCC

L'architecture résultat du flot de conception VCC contient des composants IP qui correspondent aux fonctions de calcul, tels que processeurs et IP matériels, et à celles du réseau de communication tels que bus et adaptateurs (bridge). L'architecture doit contenir un système d'exploitation temps réel ou un ordonnanceur rattaché à chaque processeur. Eventuellement, elle pourrait également avoir des composants spécifiquement rajoutés pour l'implémentation du schéma de communication, tels que des mémoires partagées, des contrôleurs DMA, des mémoires cache et des contrôleurs d'interruption.

5.1.2. L'environnement ROSES

5.1.2.1. Méthodologie de conception ROSES

Le processus de synthèse ROSES est guidé par les décisions de conception antérieures, par exemple les résultats donnés par un outil d'exploration d'architecture qui détermine le partitionnement logiciel/matériel, les protocoles de communication, les types de processeurs, les adresses de bus et mémoires, les niveaux d'interruptions et les types de données de communication.

5.1.2.2. Modèle d'architecture ROSES

L'architecture initiale du flot ROSES doit contenir des modules virtuels (MV) correspondant à tous les modules de calcul et de mémoire du système embarqué finale. Toute la structure de communication, y compris un ou plusieurs bus, doit être modelée comme un seul et unique réseau de communication considérer comme une boîte noire. Les différents IP du système sont connectés au réseau de communication en utilisant des connexions « point à point ». Les composants relatifs à la conversion des protocoles ou à l'adaptation des composants seront automatiquement synthétisés par ROSES.

L'architecture doit être complétée par des informations concernant les décisions de conception, dont quelques-unes concernent le raffinement de communication. Ainsi, l'architecture initiale est annotée par des paramètres tels que des protocoles rattachés aux ports et/ou réseaux, des types de processeurs et des types de données. Il faut également définir les paramètres d'allocation concernant l'affectation des mémoires, l'adressage des bus et mémoires et les niveaux d'interruptions.

5.1.3. Lien entre VCC et ROSES

VCC et ROSES visent des objectifs complémentaires et couvrent différents niveaux d'abstraction du flot de conception des systèmes embraqués. Alors que VCC est dédié à l'exploration de l'espace des solutions architecturales allant du niveau purement comportemental au niveau architecture, ROSES est dédié à la synthèse de la communication à partir du niveau architecture abstraite pour atteindre une architecture RTL.

Le processus de synthèse de ROSES peut prendre comme entrée le modèle architectural, résultant du flot de conception VCC. Les deux modèles architecturaux comportent les modèles abstraits des IP de l'architecture cible. Cependant, l'architecture VCC peut aussi inclure des modèles d'IP qui ont été introduits par le raffinement du schéma de communication tels que des mémoires partagées, des mémoires cache, des contrôleurs DMA ainsi que des ressources architecturales qui détaillent le réseau de communication telles que des ponts entre les bus et même des programmes de conversion de protocoles. Il est donc nécessaire de définir un style de modélisation qui rejoint le plus possible le modèle architectural de ROSES, dans lequel le réseau de communication est modélisé comme un seul composant virtuel.

De plus, VCC couvre une grande variété de niveaux d'abstraction allant du niveau purement comportemental au niveau architecture concrète dépendant des choix de conception effectués par l'utilisateur. Il est impératif que l'architecture finale dans le flot de conception VCC corresponde exactement au niveau d'abstraction attendu à l'entrée du processus de synthèse de ROSES suivant des décisions de conception précises dont plusieurs concernent le raffinement de communication. Comme exemples de décisions de conception prises au niveau du flot VCC, on peut citer le choix des types de bus et des mécanismes d'arbitrage, choix des protocoles, choix des mécanismes d'ordonnancement pour les tâches logicielles, insertion de mémoires partagées pour la mise en œuvre de la communication, définition de l'adressage des bus et mémoires et choix des types de données.

Finalement, il faut que les modèles des composants de VCC et de ROSES soient compatibles. VCC fournit quelques modèles intégrés pour des processeurs génériques, des IP matériels, des mémoires, des bus et des schémas de communication. ROSES génère des adaptateurs matériels et logiciels réalisant ainsi un ensemble de schémas de communication donnés. VCC et ROSES peuvent être étendus par des modèles spécialement dédiés à l'application.

5.1.4. Mise en œuvre du lien entre VCC et ROSES

Un lien entre les deux méthodologies de conception doit bénéficier le plus possibles des avantages spécifiques de chacune d'elles (l'exploration de l'espace des solutions architecturales avec simulation des performances pour VCC et la synthèse de la communication pour ROSES) tout en réduisant au maximum les efforts d'implémentation. Suivant l'analyse de la section précédente, les principales étapes sont :

- la définition du style de modélisation VCC qui doit correspondre le plus possible au modèle architectural ROSES dans lequel le réseau de communication est modélisé comme un module virtuel explicite ;
- la définition du niveau d'abstraction que doit atteindre le flot de conception VCC, spécifiant exactement quelles décisions de conception doivent être prises. La description de l'architecture finale VCC doit contenir toutes les informations requises pour guider le processus de synthèse ROSES ;
- l'identification de l'ensemble des ressources architecturales requises pour les applications souhaitées et l'identification/développement des modèles VCC et ROSES correspondants. Le même ensemble de ressources doit être disponible dans les deux environnements afin qu'un flot de conception complet puisse être réalisé, de la validation fonctionnelle initiale et du raffinement de conception en VCC jusqu'à la génération de la micro-architecture en ROSES ;
- un processus de traduction assure un lien automatique entre l'outil VCC et le flot ROSES. Autrement dit, le traducteur est un outil qui prend en entrée les fichiers stockés dans la base de données de VCC, effectue quelques transformations, et produit un fichier COLIF. Plusieurs étapes sont nécessaires pour accomplir ce processus de traduction: initialement, le traducteur analyse le diagramme d'affectation des blocs fonctionnels aux composants architecturaux afin d'extraire les décisions de partitionnement logiciel/matériel. L'étape suivante consiste à extraire les informations

concernant les différents blocs fonctionnels à partir du diagramme fonctionnel. Le traducteur analyse chaque bloc fonctionnel et extrait la structure de communication entre les différents blocs fonctionnels, les paramètres attachés aux ports, et le chemin vers le code source de chaque bloc fonctionnel. L'analyse du diagramme architectural se fait de la même manière que celle du diagramme fonctionnel, mais la différence est que les blocs représentent les composants architecturaux et ne référent aucun code source. Puisque, le réseau de communication utilise par le diagramme architectural de VCC (Figure 35(a)) est considéré comme une modélisation abstraite de la communication, la structure de communication de ROSES est déterminée par le réseau de communication du diagramme fonctionnel (Figure 35(b)) et les décisions de partitionnement logiciel/matériel :

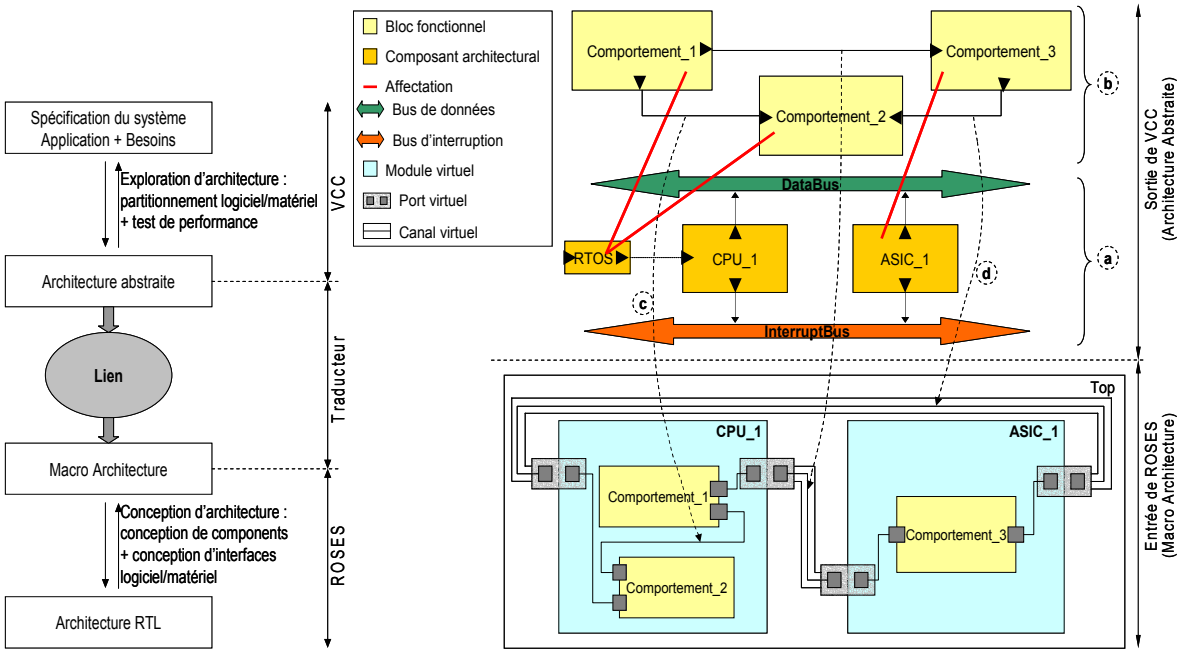

Figure 35. Intégration de VCC dans ROSES et Schéma de correspondance entre leurs objets respectifs

- si un canal connecte deux blocs fonctionnels, affectés à un même composant architectural, la connexion est gardée exactement la même. Dans ce cas, un canal est créé entre les deux ports impliqués par la connexion (Figure 35(c)) ;
- si la connexion est entre deux blocs fonctionnels affectés à deux composants architecturaux différents, un port virtuel (ayant un coté interne et un coté externe) est attaché à chaque module virtuel et trois connexions sont établies : la première connexion est entre les deux modules virtuels (qui représentent les

composants architecturaux) à travers les cotés externes des ports virtuels. Les deux autres connexions sont entre chaque module fonctionnel et le module virtuel à travers les cotés internes des ports virtuels (Figure 35(d)).

En fin, l'architecture virtuelle de ROSES est complétée avec les paramètres de conception à partir du model architecturale de VCC (par exemple les priorités des tâches). Les différentes étapes décrites ci-dessus et qui ont permis la mise en œuvre du lien entre VCC et ROSES, ont nécessité trois à quatre mois de travail.

5.1.5. Chemin d'implémentation complet du modem VDSL

5.1.5.1. Présentation générale du modem VDSL

Le VDSL est un protocole de communication utilisant les lignes téléphoniques qui fait partie des techniques xDSL (*Digital Subscriber Line*). L'application que nous avons utilisée pour notre expérimentation est une partie d'un modem VDSL. Le point de départ a été la réalisation du modem VDSL en utilisant des composants discrets par [Dia 01]. Le schéma bloc de ce prototype est illustré dans la Figure 36(a), Dans ce schéma le DSP, l'ASIC et le FPGA sont utilisés pour le traitement du signal et la MCU pour le control du modem et l'interfaçage avec le PC hôte. La partie que nous avons prise pour notre expérimentation est hachurée dans la Figure 36, dans le reste de ce chapitre elle est appelée DFU (*Deframing/Framing Unit*). Nous avons reparti la fonctionnalité du sous-ensemble hachurée sur deux processeurs ARM7 et un bloc matériel réalisant la chaîne de transmission. Ce partitionnement nous a été suggéré par l'équipe qui a réalisé le prototype du modem VDSL [Dia 01] [Mes 00].

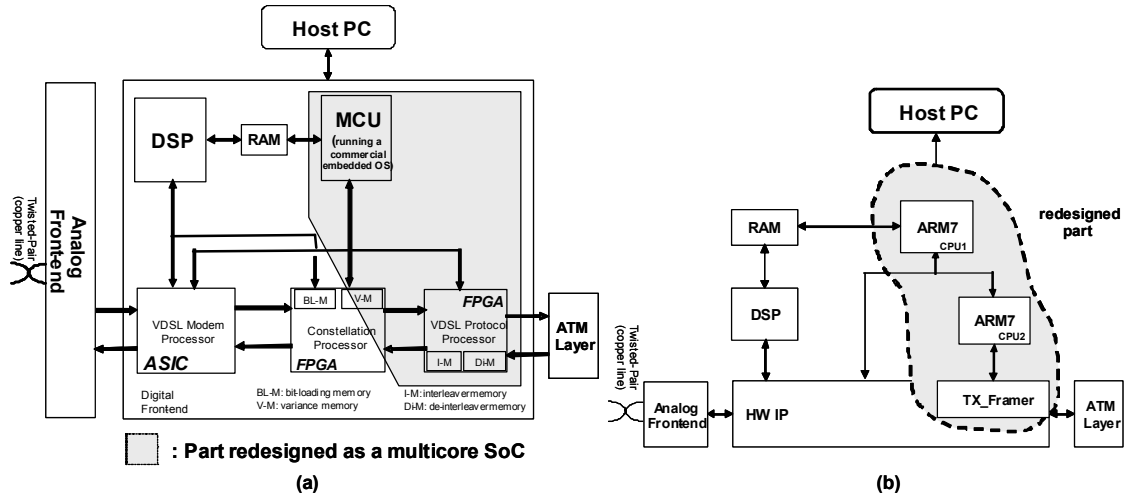

Figure 36. Le modem VDSL- schéma bloc (a) et architecture à re-concevoir DFU (b)

5.1.5.2. Conception du DFU

Cette section illustre les différentes étapes pratiques suivies au cours du flot d'implémentation du DFU

a. VCC pour la spécification et l'exploration d'architecture du DFU : le modèle VCC (Figure 37) inclut deux diagrammes : le premier présente le diagramme comportemental qui décrit le sous-ensemble DFU du modem VDSL d'un point de vue fonctionnel – (Figure 37(a)). Le deuxième présente le diagramme architectural (Figure 37(b)), qui inclut les composants architecturaux composés par deux processeurs et un ASIC. Ce dernier représente le composant IP TX_Framer (traité ici comme une boite noire). Enfin les flèches entre les deux diagrammes correspondent au partitionnement logiciel/matériel suggéré.

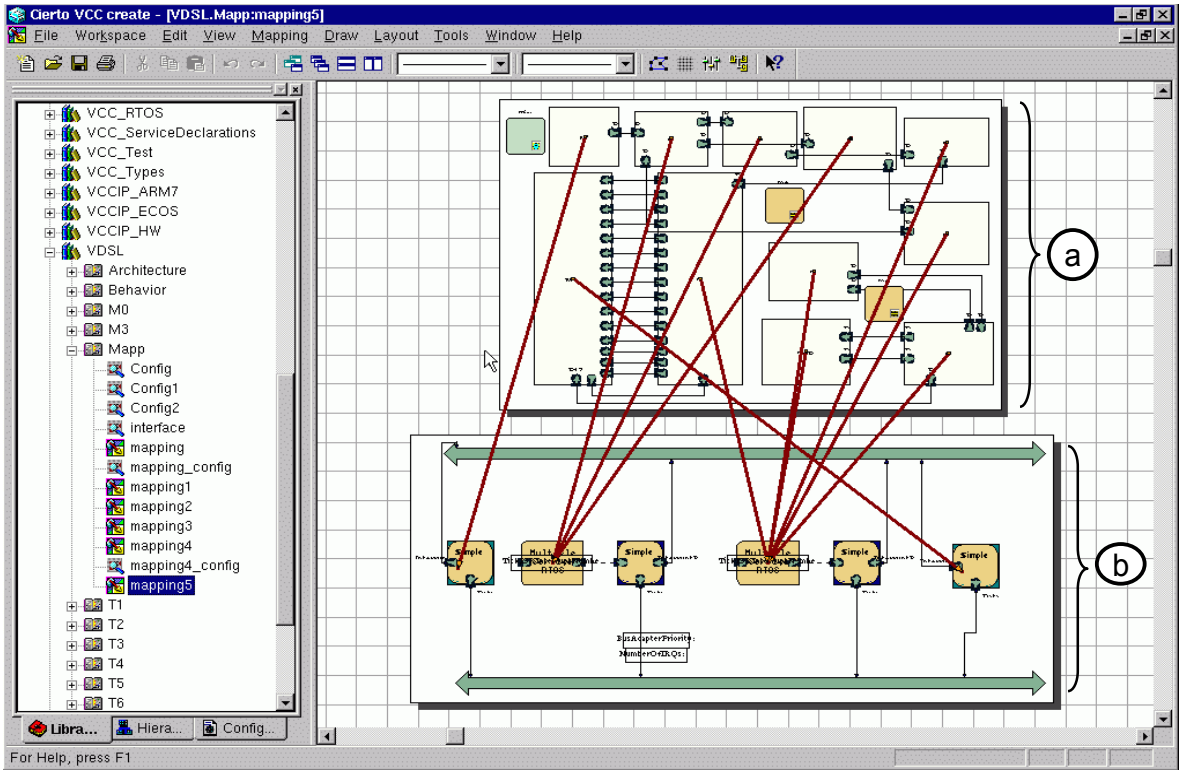

Figure 37. Le modèle VCC du DFU

b. Lien entre VCC et ROSES : à partir des différents diagrammes (fonctionnel et architectural) de l'outil VCC et les décisions de partitionnement logiciel/matériel prises, l'architecture virtuelle réclamée à l'entrée du flot ROSES est générée à l'aide du processus de traduction décrit dans la section 5.1.4. Le model généré est basé sur des connections « point à point » pour se marier avec les capacités de synthèse actuelles du flot ROSES. La Figure 38 montre la totalité de l'architecture virtuelle générée par le traducteur. Cette architecture est

composée de trois modules virtuels : deux modules logiciels qui seront implémentés par deux processeurs ARM et un module matériel qui sera implémenté par le composants IP TX_Framer.

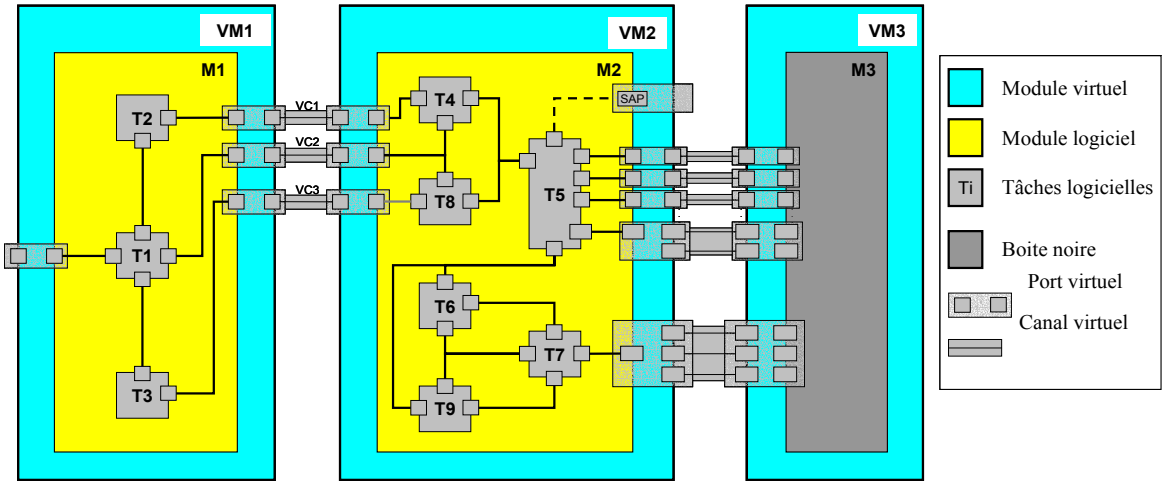

Figure 38. Le modèle d'architecture virtuelle du DFU

Après l'étape de traduction, le langage Coral et l'outil COLIF-*Checker* ont joués un rôle très important. Le premier a été utilisé pour enrichir l'ensemble des règles existantes à travers l'interface utilisateur Coral-*Builder*. Le deuxième a permis une vérification de la bonne formation du modèle COLIF généré par le traducteur et qui décrit l'architecture virtuelle du DFU. Pour la vérification de cette architecture une cinquantaine de règles sémantiques et syntaxiques ont été définis et utilisées. L'outil COLIF-*Checker* prend en entrée le fichier COLIF généré par le traducteur et l'ensemble des règles déjà définis. La tâche de cet outil consiste à interpréter l'ensemble des règles et les appliquées sur le modèle COLIF généré. A la fin du processus de vérification et si le modèle COLIF ne respecte pas toutes les règles, un rapport de violation de règles est généré automatiquement et fournis les informations nécessaires pour une éventuelle correction manuelle des bugs au niveau du traducteur.

La section suivante détaille l'étape de la conception d'architecture faite en utilisant le flot ROSES.

c. ROSES pour la conception d'architecture et la validation du DFU : l'architecture virtuelle générée à la suite du processus de traduction automatique est vérifiée en utilisant l'outil COLIF-*Checker* afin de s'assurer qu'elle constitue bien une entrée valide du flot ROSES. Pour l'implémentation du DFU, trois composants IP matériels sont sélectionnés : deux processeurs ARM7 (VM1 et VM2) et le TX_Framer (VM3). L'application logicielle est construite par une réutilisation de plusieurs composants IP logiciels existants qui implémentent les tâches T_i (i=1..9) à exécuter par les deux processeurs (Figure 38). Le

Tableau 5, résume les fonctions réalisées par chaque composant IP et leur interface. Le modèle d'architecture virtuelle capte la spécification du DFU avec des connexions « point à point » entre les trois composants IP matériels. Ce modèle peut être transposé sur des architectures différentes, cela dépend des composants IP sélectionnés, des performances désirées, et des contraintes sur la surface et sur la consommation d'énergie. Par exemple, les trois connexions « point à point » (VC1, VC2, et VC3) entre VM1 et VM2 peuvent être transposées sur un bus ou sur une mémoire partagées.

Composants IP		Fonctions	Interfaces
Logiciel	T1	Interface avec le PC hôte	registre/signal
	T2/T3	Générateur de nombres aléatoires	signal/FIFO
	T4/T8	Transmission de données (FIFO multipoint)	FIFO
	T5	Configuration des registres	registre/FIFO/SHM
	T6/T9	Synchronisation (mémoire partagée)	SHM/signal
	T7	Flot de données (FIFO)	SHM/FIFO
Matériel	ARM7	Processeur	ARM7 pins
	TX_Framer	Construction de paquetages de données	17 registres, 1 FIFO

Tableau 5. Les caractéristiques des composants IP logiciel/matériel utilisés

Le Tableau 6 liste les fonctions API disponibles pour différents types d'interfaces de composants IP logiciels. Par exemple, l'interface de la mémoire partagée (SHM) fournit des fonctions « bloquer/débloquer » et « lecture/écriture ».

Interface de composants IP logiciels		Fonctions API
logiciel/logiciel	Registre	Put/Get
	Signal	Sleep/Wakeup
	FIFO	Put/Get
	Mémoire partagée (SHM)	Lock/Unlock-Read/Write
logiciel/matériel	Registre	Put/Get
	FIFO	Put/Get
	Contrôleur de temps (Timer)	Set/Wait

Tableau 6. Interface de communication des composants IP logiciels

ROSES procède au raffinement de l'architecture virtuelle à travers les deux niveaux d'intégration (niveau système et niveau sous-système d'interface). Au niveau sous-système d'interface trois générateurs d'adaptateurs sont impliqués pour la synthèse de la communication logiciel/matériel et pour la validation du système global.

Pour la synthèse, deux générateurs sont impliqués. Ils permettent de générer les différents adaptateurs logiciel/matériel nécessaires : l'outil de génération de sous-système

d'interface logiciel appelé ASOG [Gau 01] est appliqué aux deux modules virtuels VM1 et VM2, qui corresponds aux processeurs ARM7, pour synthétiser la communication logicielle associée à chaque module. Cet outil génère pour chaque processeur un système d'exploitation personnalisé. L'outil de génération de sous-système d'interface matériel appelé ASAG [Lyo 03] synthétise, à son tour, la communication matérielle entre les trois modules (les deux processeurs ARM7 et le bloc TX_Framer). La Figure 39 montre l'architecture RTL générée du DFU. Il comprend une architecture locale d'un processeur ARM7 et un co-processeur de communication (HW wrapper dans la Figure) représentant une interface de communication matérielle très sophistiquée pour gérer les communications multipoints.

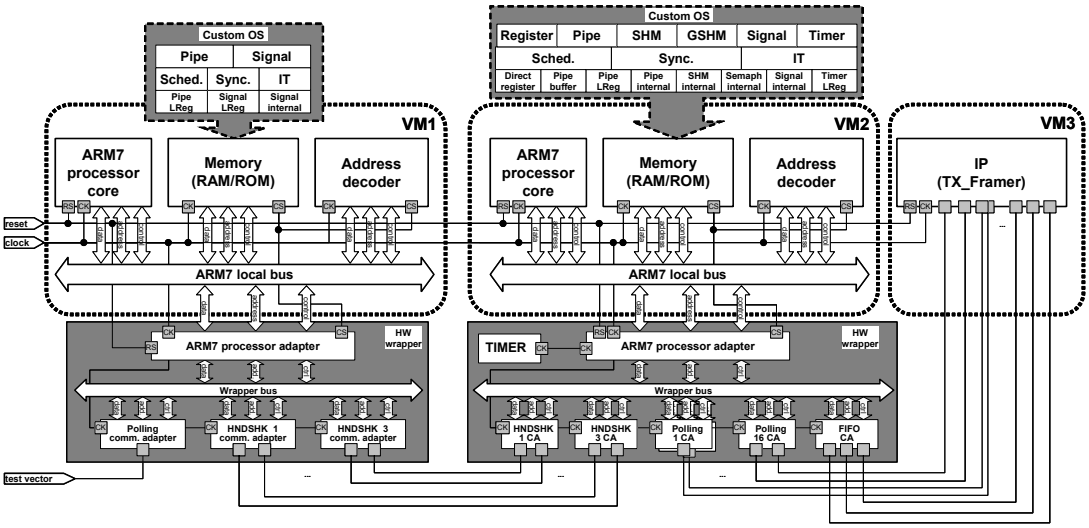

Figure 39. L'architecture RTL générée

Il est important de noter qu'à partir d'une architecture abstraite contenant deux processeurs ARM7 abstraits, ROSES génère automatiquement deux architectures locales ARM7 concrètes contenant des composants IP supplémentaires qui implémentent les mémoires locales, bus locales, et des décodeurs d'adresses.

Chaque adaptateur logiciel (système d'exploitation et la couche HAL) est personnalisé pour l'ensemble des composants IP logiciels exécutés par chaque processeur. Par exemple, les composants IP logiciels s'exécutant sur VM2 accèdent au système d'exploitation en utilisant une API : Put/Get est utilisée pour lire/écrire de/dans les registres de configuration/statut se trouvant dans le bloc TX_Framer. Une mémoire partagée (*Shared Memory – SHM*) est utilisée pour la gestion des communications basées sur une mémoire partagée. Chaque système d'exploitation contient un ordonnanceur (*Scheduler – Sched*) et des services synchronisation/interruption (Sync/IT) pour la gestion des ressources. La couche des pilotes

(HAL) contient un code de bas niveau pour accéder aux adaptateurs de canaux (CA), et d'autres routines de bas niveau spécifique au processeur ARM7.

Chaque adaptateur matériel contient un adaptateur spécifique au processeur ARM7 qui relie le bus local du processeur ARM7 aux adaptateurs de canaux. Il y a un adaptateur de canal pour chaque canal virtuel dans la spécification de l'architecture abstraite.

Le générateur des adaptateurs de co-simulation appelé CosimX [Nic 02] [Yoo 01] est utilisé pour la validation de l'architecture RTL générée. En utilisant CosimX deux modèles de simulation ont été générés :

- un modèle de simulation permettant la simulation native du logiciel (l'application et la couche du système d'exploitation) ;
- un modèle de simulation pour l'exécution du logiciel sur le simulateur ISS (*Instruction Set Simulator*). Dans le cas de notre expérimentation, le simulateur pour le processeur ARM.

La différence entre ces deux modèles se situe au niveau des temps de simulation. La simulation native du logiciel a fournit un gain en vitesse de 2 ordres de grandeur : la co-simulation avec le logiciel en natif a été 100 fois plus rapide que la co-simulation à base d'ISS.

5.1.6. Evaluation et résultats

La conception manuelle du modem VDSL complet nécessite plus qu'une dizaine d'hommes/année ; celle du DFU présenté est estimée à plus que cinq hommes/année d'effort. Cette expérience à montrer qu'en utilisant ROSES ce temps est réduit à six mois pour l'implémentation d'une solution architecturale donnée. L'exploration de l'espace architecturale pour avoir un partitionnement logiciel/matériel consomme une partie importante du temps de conception. En utilisant VCC, le temps nécessaire pour explorer l'espace de conception d'une application spécifique peut être réduit considérablement. Ca prend seulement quelques semaines pour explorer une grande variété de solutions. En combinent VCC et ROSES, il est possible de synthétiser plusieurs solutions pendant une semaine. Les résultats de l'expérience montrent que cette combinaison fourni un gain de temps de conception important, permet simultanément l'exploration de plus de solutions par une génération et une évaluation rapide des solutions architecturales et la fourniture d'un chemin complet allant de l'exploration de l'espace de conception jusqu'à la synthèse au cycle d'horloge prés.

La conception du DFU a nécessitée seulement une personne pour une période de quatre mois (sans compter l'effort nécessaire pour développer les éléments de bibliothèque et pour le déboguage des outils de conception). Ceci correspond à 15 fois plus de réduction des efforts de conception par rapport à l'estimation de cinq hommes/année pour la conception manuelle.

Le Tableau 7 présente les résultats concernant les systèmes d'exploitation générés. Une partie du système d'exploitation est générée en assembleur et inclut quelques routines de bas niveau telles que le changement de contexte et le démarrage du processeur, qui sont spécifiques à chaque processeur.

résultats du SE	# de lignes C	# de lignes Assembleur	Taille du code (bytes)	Data size (bytes)
VM1	968	281	3829	500
VM2	1872	281	6684	1020
Changement de contexte (cycles)				36
Latence pour les traitements d'interruption (cycles)				59(OS) + 28(ARM7)
Latence pour les appels système (cycles)				50
Reprise d'exécution des tâches (cycles)				26

Tableau 7. Les résultats de génération de système d'exploitation

Si on compare les chiffres présentés dans le Tableau 7 avec un système d'exploitation configurable commerciale, les résultats sont acceptables. Généralement, la taille minimale des systèmes d'exploitation commerciales est environ 4 kbytes, mais avec cette taille peu d'entre eux fournissent les fonctionnalités nécessaires. La performance est aussi acceptable : le changement de contexte prend 36 cycles, la latence pour les interruptions matérielles est de 59 cycles (plus 4 à 28 cycles nécessaires au processeur ARM7 pour réagir), la latence pour les appels système est 50 cycles, et la réactivation des tâches prend 26 cycles.

Interfaces matérielles	# de portes	Délai du chemin critique (ns)	Fréq. Max. (MHz)
VM1	3284	5.95	168
VM2	3795	6.16	162
Latence pour une opération de lecture (cycles d'horloge)			6
Latence pour une opération d'écriture (cycles d'horloge)			2
Nombre de lignes de code (RTL VHDL)			2168

Tableau 8. Les résultats de génération d'adaptateurs matériels

Le Tableau 8 montre les résultats de génération d'adaptateurs matériels en utilisant une technologie CMOS 0.35 µm. Les résultats sont acceptables parce que l'adaptateur compte moins que 5% de la surface du cœur ARM7 et possède un chemin critique qui correspond à

moins que 15% du cycle d'horloge pour une fréquence 25MHz des processeurs ARM7 utilisés dans cette expérimentation.

5.2. Intégration d'un IP de communication de haut niveau dans ROSES

L'objectif de cette deuxième expérimentation est d'intégrer un nouveau IP de communication et d'un environnement de simulation correspondant dans l'environnement ROSES. L'IP de communication permet l'échange de données entre les différents modules du système sous forme de transaction : il s'agit d'une modélisation des systèmes embarqués au niveau transaction (*Transaction Level Modeling* – TLM). Cette intégration permet la génération d'une architecture au niveau RTL à partir d'une modélisation du système au niveau transaction (modèle TLM).

Dans une première étape, nous proposons une présentation du concept de la modélisation au niveau transaction, les caractéristiques du modèle TLM, le domaine d'utilisation de ce modèle, et l'IP de communication au niveau transaction (IP TLM). Dans une deuxième étape, nous présentons le processus d'intégration de l'IP TLM. Nous proposons aussi une présentation des différents enrichissements de la bibliothèque des systèmes d'exploitation de l'environnement ROSES afin de supporter la nouvelle API utilisée par l'IP TLM.

Les sections suivantes détaillent les différents travaux faits au cours de cette expérience.

5.2.1. Le concept de la modélisation au niveau transaction

La modélisation au niveau transaction se reporte à l'approche utilisant des messages comme un mécanisme d'échange de données entre les composants d'un système. Cette approche est très connue et largement utilisée dans le domaine de l'ingénierie du logiciel. Dans le contexte de la modélisation du matériel, la modélisation au niveau transaction (TLM) est devenue un mot clé pour désigner les approches de modélisation qui ont une tendance à définir des niveaux d'abstraction plus haut que le niveau RTL. Les principales raisons de cette tendance sont :

- atteindre une vitesse élevée pour permettre le développement du logiciel embarqué sur des modèles de simulation ;
- un développement rapide des spécifications exécutables ;
- l'analyse et l'exploration d'architecture des systèmes embarqués.

Le modèle TLM utilisé, dans cette expérimentation, est basé sur le langage SystemC.

5.2.2. L'IP de communication au niveau transaction : TAC

Dans le modèle TLM utilisé, la communication est assurée par un IP de communication appelé « *Transaction Accurate Channel* – TAC » [Clo 02] [Pas 02]. Cet IP TLM supporte des adresses mémoires, des types de données arbitraires, et la connexion d'un nombre arbitraire de modules. Les modules peuvent être soit des initiateurs de transaction, soit des cibles de transaction, ou les deux en même temps. Un initiateur de transaction (ou encore maître) est un module qui lance des transactions avec le système, et il est connecté au canal TAC à travers des ports maîtres. Une cible de transaction (ou encore esclave) est un module qui permet de servir les requêtes qui viennent du canal TAC, et il est connecté à ce dernier à travers des ports esclaves. Les transactions sont lancées en utilisant une API d'écriture « *Write* » vers le canal TAC et interceptées en utilisant une API de lecture « *Read* » vers le canal TAC.

5.2.3. L'utilisation de TLM

Le domaine d'utilisation du modèle TLM est le développement au plus tôt du logiciel, avant l'élaboration de l'architecture RTL. Puisque les systèmes embarqués récents contiennent au moins un processeurs, le logiciel est devenu une partie essentielle de ces systèmes [Roa 01]. Le modèle TLM facilite le développement du logiciel en permettant une simulation rapide des modèles développés [Pas 02]. Une autre utilisation de ce modèle est la vérification du matériel des systèmes embarqués. Par exemple, un module fonctionnel peut lancer des transactions à travers le canal TAC pour vérifier le bon fonctionnement d'autres modules tels que les mémoires. Les deux lignes suivantes montrent deux exemples de prototypes des API TLM « *Read/Write* » :

```
Read (ADDR_TYPE addr, DATA_TYPE data, unsigned int number, unsigned int byte_enable, access_option_t access_option);
Write (ADDR_TYPE addr, DATA_TYPE data, unsigned int number, unsigned int byte_enable, access_option_t access_option);
```

Les paramètres qui figurent sur les deux prototypes d'API « *Read/Write* » sont respectivement :

- addr : l'adresse à laquelle les données sont stockées ;
- data : les données à stocker à l'adresse addr ;
- number : le nombre d'éléments impliqués dans la transaction, la valeur par défaut est 1 ;
- byte_enable : le nombre de bits utilisés, 32 bits, 16 bits ou 8 bits ;
- access_option : le type d'accès à un canal, accès simple ou accès bloquant.

D'après la Figure 40, cette expérimentation propose le raffinement des modules logiciels à travers la génération d'un système d'exploitation personnalisé avec une API TLM.

La génération des adaptateurs matériels fera l'objet de travaux futur, par conséquent elle n'est pas abordée dans cette présentation. Le générateur des systèmes d'exploitation prend en entrée une description de l'architecture logiciel/matériel en COLIF. Par conséquent, le traducteur SystemC-COLIF a été étendu pour assurer une transformation automatique du modèle TLM vers le modèle COLIF. En plus, les nouvelles fonctionnalités ont été ajoutées à la bibliothèque du générateur des systèmes d'exploitation pour les nouvelles API TML « *Read*/*Write* ».

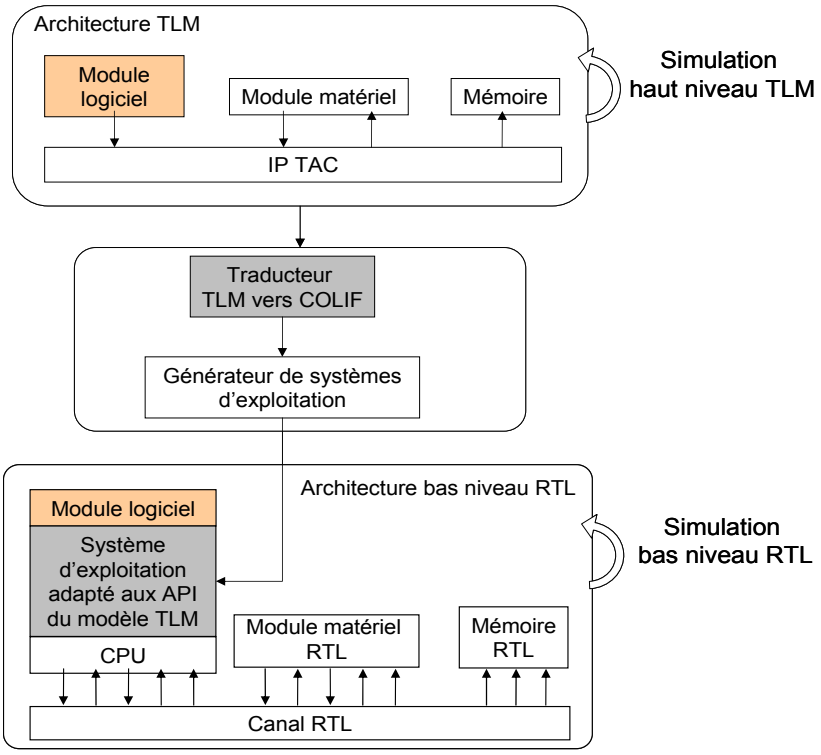

Figure 40. Intégration du modèle TLM dans ROSES

Dans les deux sections suivantes, nous présentons le processus de traduction du modèle TLM vers le modèle COLIF, ensuite nous décrirons les différentes fonctionnalités ajoutées à la bibliothèque du système d'exploitation.

5.2.4. Traduction du modèle TLM vers COLIF

L'intégration du modèle TLM dans le flot ROSES nécessite la construction d'un traducteur entre ce modèle et COLIF. Ce traducteur assure une transformation automatique du modèle TLM vers le modèle COLIF, ce qui permet aux concepteurs d'éviter un travail manuel très fastidieux et source d'erreur. Cette section est consacrée à la présentation de ce traducteur.

Puisque le modèle TLM est basé sur le langage SystemC, le traducteur doit donc supporter ce langage. Les étapes suivantes décrivent les travaux effectués pour construire le traducteur. L'effort nécessaire pour accomplir ces travaux était de trois à quatre mois.

- Maîtrise des concepts et de la structure interne des bibliothèques du langage SystemC.
- annotation des objets SystemC par des paramètres : pour pouvoir annoté tous les objets SystemC (module, port, canal) par des paramètres de conception, une nouvelle méthode est ajouté au code source d'une classe mère appelée *sc_object*. Le rôle de cette méthode est de récupérer les identificateurs Id des objets SystemC et de leur affecter les paramètres (un paramètre est caractérisé par un nom et une valeur). Pour chaque objet, l'identificateur et l'ensemble des paramètres correspondants sont stockés dans une liste. Cette liste est utilisée pour annoter les différents objets COLIF lors de leur création à partir des objets SystemC. La création des différents objets de COLIF se fait en utilisant la couche API offerte par ce dernier ;
- extraction d'information à partir de SystemC : pour pouvoir simulé le système, le simulateur de SystemC utilise des structures de données en interne. Ces structures de données sont récupérables grâce à un ensemble de classes fournis par SystemC. De cette façon, nous pouvons extraire toutes les informations et les objets SystemC : les modules, le nombre de port attachés à chaque module, les canaux connectés à chaque ports, etc. Cependant, les noms des ports, leur type de donnée, et leur interface ne peuvent pas être extraits en utilisant la méthode sous-jacente. Ils sont plus tôt obtenus par un parcours des fichiers d'entête de chaque module. Les objets COLIF sont créés au fur et à mesure que l'identification des différents objets SystemC. Dans le but d'effectuer des recherches plus rapide, les différents objets sont mis dans des dictionnaires qui permettent de garder une trace entre chaque objet COLIF crée et l'objet SystemC correspondant ;
- identification des objets TLM : lors de l'extraction des objets SystemC, les objets TLM sont aussi identifiés. Les ports maîtres/esclaves utilisant l'API TLM « *Read/Write* » sont transformés, dans COLIF, en ports SAP_TLM (*Service Access Port* TLM). Un port SAP_TLM maître offre le service « *Write* », et un port SAP_TLM esclave offre le service « *Read* ». Le canal TAC est représenté, dans COLIF, sous forme d'un module de type « *tac_channel* ».

Après l'étape de traduction, le langage Coral et l'outil COLIF-*Checker* ont joués un rôle très important. Le premier a été utilisé pour enrichir l'ensemble des règles existantes à travers l'interface utilisateur Coral-*Builder*. Le deuxième a permis une vérification de la bonne

formation du modèle COLIF généré par le traducteur. Pour la vérification du modèle COLIF généré par le traducteur, une soixantaine de règles sémantiques et syntaxiques ont été définis et utilisées. L'outil COLIF-*Checker* prend en entrée le fichier COLIF généré par le traducteur et l'ensemble des règles déjà définis. La tâche de cet outil consiste à interpréter l'ensemble des règles et les appliquées sur le modèle COLIF. A la fin du processus de vérification et si le modèle COLIF ne respecte pas toutes les règles, un rapport de violation de règles est généré automatiquement et fournis les informations nécessaires pour une éventuelle correction manuelle des bugs au niveau du traducteur.

5.2.5. Les fonctionnalités ajoutées à la bibliothèque de systèmes d'exploitation

Cette section est consacrée à la description de l'enrichissement de la bibliothèque de systèmes d'exploitation avec les primitives de communication « *Read/Write* » du modèle TLM. Cette tâche est accomplie en plusieurs étapes :

- un nouvel élément est ajouté à la description de la bibliothèque de système d'exploitation. Cet élément s'appelle TLM et offre deux services de haut niveau : HighIO/TLM/Write et HighIO/TLM/Read ;
- pour ce nouvel élément TLM, des fichiers sont écrits en utilisant un langage de macro (IO/High/TLM/tlm_master.h.gen et IO/High/TLM/tlm_master.c.gen). Ces fichiers de macro code sont expansés lorsque les services « HighIO/TLM/Write et HighIO/TLM/Read » sont réclamés par l'application logicielle ;
- les primitives de haut niveau « *Read* » et « *Write* » nécessitent un élément de bas niveau (dans la couche HAL) pour pouvoir accéder à la partie matérielle du système. Cet élément est alors ajouté à la description de la bibliothèque du système d'exploitation, il est appelé TLMLow, et il offre des services Input/Output de bas niveau : LowIO/TLMPut et LowIO/TLMGet ;
- pour ce nouvel élément TLMLow, des fichiers sont écrits en utilisant un langage de macro (IO/High/TLM/tlm_master.h.gen et IO/High/TLM/tlm_master.c.gen). Ces fichiers de macro code sont expansés lorsque les services « HighIO/TLM/Write et HighIO/TLM/Read » sont réclamés par l'application logicielle.

Le raffinement des modules logiciels est maintenant possible en utilisant le système d'exploitation généré, puisque la bibliothèque de ce dernier reconnaît l'API TLM « *Read*/*Write* ».

5.3. Conclusion

Ce chapitre a présenté l'application des concepts proposés sur des applications complexes : l'intégration de l'outil VCC dans l'environnement ROSES et l'utilisation de ce dernier pour la conception d'un système embarqué multiprocesseur : la partie DFU du modem VDSL – l'intégration d'un IP de communication au niveau transaction (canal TAC), permettant de modéliser le système au niveau transaction (modèle TLM), dans l'environnement ROSES. Les objectifs ont été : (1) de prouver l 'efficacité et la flexibilité du modèle COLIF, (2) montrer que COLIF permet à ROSES de présenter un environnement ouvert à l'intégration d'outils et de composants IP, et (3) de montrer les capacités d'intégration de composants logiciel/matériel offertes par ROSES.

Les expérimentations ont montrées que l'utilisation de l'environnement ROSES pour l'intégration d'outils et des composants IP logiciel/matériel a prouvé un gain de temps de conception très important. En plus l'utilisation du modèle d'intégration d'outils a permit une soudure entre les niveaux d'abstraction de la communication. Par exemple, le lien entre VCC et ROSES permet une soudure entre le niveau macro-architecture et le niveau RTL. Le lien entre le modèle TLM et ROSES permet une soudure entre le niveau purement fonctionnel et le niveau RTL. L'utilité de la vérification du modèle COLIF a été aussi montrée.

Conclusion et perspectives

Conclusion

La tendance très répandue dans la conception des systèmes sur puce est l'assemblage des composants existants. La difficulté d'une telle conception provient du fait que : (1) les composants sont hétérogènes sur plusieurs aspect : en termes de protocoles de communication, niveaux d'abstraction, langages de spécification et granularités et (2) un outil de conception complet n'existe toujours pas. Dans cette étude nous avons abordé deux principaux problèmes liés à la conception des systèmes embarqués hétérogènes : l'intégration des composants logiciel/matériel et l'intégration d'outils de conception.

Le chapitre 2 a présenté une étude sur les méthodologies de réutilisation de composants logiciel/matériel et de réutilisation d'outils de conception. La première partie a présentée le cycle de réutilisation des composants IP et a défini les différents styles de conception des systèmes embarqués. A la lumière de ces concepts, une étude des solutions proposées pour la réutilisation et l'intégration des composants logiciel/matériel a été entreprise. Ces solutions proposent l'utilisation des standards existant, mais le problème est que plusieurs standards co-existent ce qui rend leur intégration sur la même puce très difficile. La construction d'adaptateurs logiciel/matériel pour adapter les différents composants logiciel/matériel au réseau de communication s'avère une solution nécessaire. La deuxième partie de ce chapitre a été consacrée à la réutilisation d'outils de conception existants. Dans un premier temps, nous avons abordé le problème d'interopérabilité entre les outils. Différents mécanismes d'échange de données de conception entre les outils ont été présentés. En montrant qu'aucun outil de conception ne convient parfaitement pour couvrir toutes les étapes de conception, nous avons effectué une présentation des outils les plus connue existants. Cette présentation montre que la plupart des outils sont spécifiques à un domaine d'application donnée et à une étape de conception bien déterminée. Les outils qui essayent de couvrir toutes les étapes de conception proposent des solutions très limitées. La tendance très récente consiste à combiner et intégrer plusieurs outils dans un même flot de conception dans le but de construire des flots de

conception complet. L'utilisation d'un modèle de référence (ou encore format standard) s'avère une solution attractive pour assurer l'échange de données entre les différents outils de conception. Il est intéressant que ce modèle soit capable d'évoluer au cours du processus de raffinement à travers plusieurs niveaux d'abstraction. Par conséquent, il peut être utilisé par une grande variété d'outils de conception à différentes étapes de conception.

Dans le chapitre 3, nous avons proposé une présentation du flot de conception des systèmes hétérogènes embarqués ROSES développé par l'équipe SLS. Ce flot est basé sur l'assemblage des composants IP logiciel/matériel existants et permet de générer automatiquement une architecture RTL à partir d'un modèle d'architecture abstraite. Ce chapitre commence par présenter les modèles et les composants définis par ROSES pour la conception des SoC. Une présentation des schémas d'intégration des composants logiciel/matériel et de l'architecture abstraite (architecture virtuelle) définis par ROSES a été faite. Le flot ROSES utilise un modèle de représentation pour la spécification des systèmes hétérogènes. Ce format intermédiaire couvre les différents niveaux d'abstraction et permet la description des systèmes hétérogènes où les différents composants sont décrits en des langages de spécification différents et/ou aux niveaux d'abstraction différents. Le concept central de ce format est la séparation entre la communication et le comportement. Pour ce faire, le modèle de représentation présenté propose l'enveloppement de chaque composant hétérogène dans une enveloppe qui sépare la communication et le comportement en abstrayant l'interconnexion d'un composant du reste du système. Un composant et son enveloppe forment un composant virtuel. Ainsi, les systèmes hétérogènes sont représentés comme un ensemble de composants virtuels interconnectés, dans une architecture virtuelle. Dans une deuxième partie, ce chapitre a présenté les différents outils offerts par ROSES. Ces outils permettent la génération automatique des adaptateurs logiciels, matériels, et de co-simulation. Enfin, une analyse des outils de ROSES a été faites afin d'identifier leur problèmes. Bien que ces outils proposent l'assemblage de composants d'interface de base pour construire des composants plus complexes (adaptateurs), chaque outil correspond à un sous-flot de conception indépendant des autres outils de ROSES (la coordination entre les outils de ROSES est accomplie grâce au format intermédiaire COLIF). Par conséquent, chaque outil utilise sa propre technique de composition, ses propres étapes de composition et sa propre structure de bibliothèque de composants. Une unification des étapes de composition et des bibliothèques de composants s'avère une solution attractive pour résoudre les problèmes liés aux outils de ROSES.

Le chapitre 4 a présenté, dans un premier temps, une méthodologie unifiée pour l'intégration des composants logiciel/matériel pour la conception et la validation par co-simulation des systèmes hétérogènes embarqués. L'idée de base est la définition d'un flot générique de composition des composants indépendamment de leur nature et leur granularité. Ce flot générique permet, selon le niveau d'intégration, de générer des architectures raffinée à partir du modèle architecturale abstrait (présenté dans le chapitre 3) qui défini les besoins du système en terme de services requis. Ce flot agit à deux niveau d'intégration différents : au niveau système, il génère des architectures SoC pour la synthèse ou la co-simulation. A ce niveau les adaptateurs sont utilisés en tant que composants. Au niveau interface, ce flot génère des sous-systèmes d'interface très sophistiqués (adaptateurs) : logiciels, matériels, et de co-simulation. Ces sous-systèmes d'interface sont construits par un assemblage des composants d'interface de base et sont utilisés en tant composants au niveau système pour adapter les différents composants logiciel/matériel au réseau de communication. La technique de composition est basée sur les services requis/fournis par les différents composants. Une présentation du flot ROSES a été proposée selon la méthodologie unifiée sous-jacente. Dans une deuxième partie, ce chapitre a présenté un modèle d'intégration d'outils autour du format intermédiaire présenté dans le chapitre 3. Ce format est utilisé comme un modèle de référence pour assurer l'échange de données de conception entre les différents outils. L'intégration d'outils nécessite la construction de traducteurs vers ce format intermédiaire. Une vérification du format généré à partir du processus de traduction est un étape essentielle pour s'assurer de la bonne formation et de la cohérence de ce format et afin d'assurer une interopérabilité entre les outils. Pour ce faire un outil de vérification a été mis en place. Cet outil est basé sur une base de données de règles syntaxiques et sémantiques spécifiques à ce format intermédiaire. Un langage permettant la spécification de ces règles a été aussi développé.

Le chapitre 5 a présenté l'application des concepts proposés sur des problèmes complexes : (1) l'intégration de l'outil VCC de cadence dans l'environnement ROSES et l'utilisation de cette combinaison pour la conception d'un système embarqué multiprocesseurs - le DFU ; (2) l'utilisation de l'environnement ROSES pour l'intégration d'un IP de communication TLM (canal TAC) et son environnement de simulation. Les objectifs ont été : (1) de prouver l'utilité et l'efficacité du modèle d'intégration d'outils, (2) d'appliquer la méthodologie unifiée suivie par ROSES pour la conception et la validation des SoC sur des applications complexes réelles et (3) de montrer les avantages de l'approche d'intégration de composants logiciel/matériel offerte par ROSES. Le format intermédiaire s'est avéré un repère commun très efficace pour les différents outils internes de ROSES et

également pour les outils externes. Le processus de vérification (accompli par l'outil COLIF-Checker et en utilisant le langage Coral) permet un gain très important de temps de conception en s'assurant de la bonne formation et la cohérence du format intermédiaire. Concernant la méthodologie d'intégration d'outils, l'utilisation de l'environnement ROSES a permis de tirer des bénéfices offerts par ce dernier et les performances des outils existants tel que VCC pour l'exploration d'architecture. En plus du gain important de temps de conception prouvé par cette combinaison, elle permet aussi une soudure entre l'étape d'exploration d'architecture et l'étape de conception d'architecture. L'intégration du canal TAC et de son environnement de simulation dans l'environnement ROSES permet de profiter des bénéfices offerts par les deux environnements. En plus, ça permet une modélisation des systèmes au niveau transaction (TLM) et un passage automatique au niveau RTL.

Perspectives

Les travaux de cette thèse ont été consacrés à deux aspects très importants pour la conception des systèmes hétérogènes embarqués : l'intégration de composants logiciel/matériel et l'intégration d'outils de conception. En ce qui concerne l'intégration de composants, une étude du flot ROSES actuel a été faite. Suite à cette étude, un travail de généralisation a été proposé afin de créer un environnement ouvert à l'intégration de composants logiciel/matériel. Il est important de poursuive la conceptualisation de ce travail de généralisation afin de pouvoir passer à la phase d'implémentation d'un nouveau flot ROSES. Il est intéressant aussi de constater les bénéfices d'une telle généralisation sur le partitionnement de la communication logiciel/matériel. En ce qui concerne l'intégration d'outils et à la suite d'une étude du format intermédiaire COLIF, un travail qui vise la vérification d'échange de données de conception a été proposé. La vérification se base sur une base de données de règles syntaxiques et sémantiques spécifiques à COLIF. Il est intéressant d'enrichir cette base de règles afin de couvrir toutes les utilisations possibles du flot ROSES. Une évolution de l'outil COLIF-*Checker* peut être aussi envisagé pour permettre une vérification du modèle COLIF au niveau RTL. En effet, le modèle COLIF généré à partir des processus de traduction représente le modèle d'entrée du flot ROSES (c'est-à-dire qui respecte l'architecture virtuelle). Actuellement, ce modèle est raffiné par l'outil ASAG afin d'inclure les détailles de communication matérielle (c'est-à-dire les informations concernant les adaptateurs matériels). Le modèle COLIF raffiné (au niveau RTL) est le modèle d'entrée de l'outil CosimX qui permet la génération d'un modèle de co-simulation au niveau RTL. Par

conséquent, une vérification du modèle COLIF au niveau RTL est nécessaire pour le bon fonctionnement de l'outil CosimX. Actuellement, COLIF-*Checker* s'intéresse seulement à l'intégration des outils externes (c'est-à-dire qu'il permet seulement une vérification du modèle d'entrée du flot ROSES). Enfin, ces travaux de thèse étaient consacrés à des caractéristiques bien déterminées d'un environnement de conception : intégration d'outils et intégration de composants logiciel/matériel. Cet environnement a été construit autour d'un format intermédiaire. Il est important de suivre la construction de cet environnement autour du format intermédiaire afin de supporter d'autres caractéristiques telles que la gestion des versions, le support d'un environnement distribué (par exemple co-simulation distribuée, échange de données basé sur le modèle Client (outil externe et traducteur)/serveur (COLIF)), la qualification et la classification de composants. Ceci permettra à ROSES d'être un environnement central pour la conception et la validation des SoC à base de composants logiciel/matériel.

Des études de cas ont été aussi présentées : pour le cas d'intégration de l'outil VCC dans l'environnement ROSES, il est intéressant de développer de nouveaux modèles de performances pour les bibliothèques de VCC, en particulier pour les schémas de communication synthétisés par ROSES. Ceci fermerait la boucle de conception permettant d'obtenir une simulation des performances VCC préliminaire très réaliste des raffinements de communication alternatifs réalisés par ROSES. Dans le cas de l'intégration du canal TAC, il est intéressant de proposer une génération automatique des adaptateurs matériels en plus de celle des adaptateurs logiciels déjà accomplie. Ceci permettra l'obtention d'une architecture RTL complète et par la suite permettra une simulation plus détaillée du système.

A plus long terme, ce type d'environnement pourra servir pour la construction d'environnement d'optimisation d'architectures pour les systèmes complexes.

Bibliographie

[Ale 01] P.Alexander, C.Kong. “Rosetta: Semantic Support for Model-Centered Systems-Level Design,” IEEE Computer, November 2001.

[Amb 01] ARM AMBA. Available at: **http://www.arm.com/armtech.nsf/html/AMBA**

[Arm 99] AMBA specification (REV 2.0) ARM Limited, 13 Mai 1999. Available at: **http://arm.com**

[Bag 97] A.Baganne, J.Philippe, and E.Martin, “A Formal Technique for Hardware Interface Design,” In Proc. Int’l Symposium on Circuits and Systems, June 1997.

[Bar 99] C.Barna, W.Rosenstiel. “Object-Oriented Reuse Methodology for VHDL”. DATE’1999.

[Ben 98] L.Benini, A.Bogliolo, G.De Micheli. “Dynamic Power Management of Electronic Systems,” ICCAD 1998.

[Ber 00] R.A.Bergamaschi and W.R.Lee. “Designing Systems-on-Chip Using Cores,” In Proc. Design Automation Conf., June 2000.

[Ber 01] R. A. Bergamaschi, et al. “Automating the Design of SoCs using Cores,” IEEE Design and Test of Computers, September-October 2001.

[Bir 99] M.Birnbaum, H.Sachs. “How VSIA Answers the SoC Dilemma,” Computer, June 1999.

[Bok 00] C.Böke. “Combining Two Customization Approaches: Extending the Customization Tool TEReCS for Software Synthesis of Real-Time Execution Platforms,” Workshop on Architectures of Embedded Systems (AES2000), Jan. 2000.

[Bru 00] J-Y.Brunel et al. “COSY communication IP’s,” *DAC 2000,* Los Angeles, California, USA.

[Cad 01] Cadence Inc. Virtual Component Co-design, 2001. Available at: **http://www.cadence.com**

[Ces 01] W.O.Cesario et al. “Colif: a Multilevel Design Representation for Application-Specific Multiprocessor System-on-Chip Design", 12th IEEE International *Workshop on Rapid System Prototyping*, June 25-27, 2001, California, USA.

[Ces 02] W.O.Cesario et al. “Component-Based Design Approach for Multicore SoCs”. Design Automation Conf., June 2002.

[Cho 99] P.Chou et al. “IPChinook: An Integrated IP-Based Design Framework for Distributed

Embedded Systems". DAC'1999.

[Clo 02] A. Clouard, G. Mastrorocco, F. Carbognani, A. Perrin, F. Ghenassia, "Toward Bridging the Precision Gap between SoC Transactional and Cycle-Accurate Levels". In Proc. Design Automation and Test in Europe (DATE'02), 4-8 March 2002, Paris, France.

[Cor 99] IBM CoreConnect Bus Architecture. Available at: **http://www-3.ibm.com/chips/products/coreconnect**

[Dal 99] M.Dalpasso, A.Bogliolo, L.Benini. "Virtual Simulation of Distributed IP-Based Designs". Design Automation Conf., June 1999.

[Dal 99] M.Dalpasso, A.Bogliolo, L.Benini. "Specification and Validation of distributed IP-Based Design with JavaCAD," In Proc. Design Automation and Test in Europe, Mars 1999, Munich, Germany.

[Des 97] Design & Reuse. Available at: **http://www.design-reuse.com**

[Dia 01] M.Diaz-Nava, G.S.Okvist, "The Zipper prototype: A Complete and Flexible VDSL Multi-carrier Solution", ST Journal special issue xDSL, September 2001.

[DOM] DOM, (Document Object Model) published by W3 consortium. Available At: **www.w3.org**

[Ern 93] R.Ernst, J.Henkel, Th.Benner, "Hardware software cosynthesis for microcontrollers," *IEEE Design and Test of computers,* December 1993.

[Fau 99] N.Faulhaber, R.Seepold. "A Flexible Classification Model for Reuse of Virtual Components". In: R.Seepold and A.Kunzmann (eds), Reuse Techniques for VLSI Design, Kluwer Academic Publishers, 1999.

[Fin 00] A.Fin, F.Fummi, "A Web-CAD Methodology of IP-Core Analysis and Simulation," In Proc. of Design Automation Conference, Los Angeles, California, June 2000.

[Gau 01] L.Gauthier, "génération de système d'exploitation pour le ciblage de logiciel multitâche sur les architectures multiprocesseurs hétérogènes dans le cadre des systèmes embarqués spécifiques." PhD Thesis, December 2001. Available at : **http://tima.imag.fr/publications/publications.asp**

[Gau 01] L.Gauthier, S.Yoo, A.A.Jerraya, "Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software". DATE 2001.

[GEAR] Synchronicity IP Grear Pulblisher Suite. Available at: **http://www.synchronicity.com/products/publisher/publisher.htm**

[Gha 03] F. Gharsalli, "Conception des interfaces logiciels/matériels pour l'intégration des mémoires globales dans les systèmes monopuces." PhD Thesis, July 2003. Available at : **http://tima.imag.fr/publications/publications.asp**

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

[Hes 00] F.Hessel, P.Coste, G.Nicolescu, P.Le Marrec, N.-E. Zergainoh, and A.A.Jerraya, “Multi-level Communication Synthesis of Heterogeneous Multilanguage Specifications,” In Pro. Int’l Conference on Computer Design, September 2000.

[Hoa 85] C.A.R. Hoare, “Communicating Sequential Processes,” Prentice Hall, 1985.

[Ibm 02] IBM, “The CoreConnect Bus Architecture”, 2002. Available at: **http://www.chips.ibm.com/products/coreconnect/**.

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

[Ism 96] T.Ben Ismail, J.Daveau, K.O’Brien, and A.A.Jerraya, “A System-level Communication Synthesis Approach for Hardware/Software Systems,” Microprocessors and Microsystems, 20(3): 149-157, May 1996.

[Kea 02] M.Keating, P.Bricaud. “Reuse Methodology Manual for System-on-a-Chip Designs”. Kluwer Academic Publishers, 2002 (3rd edition).

[Keu 00] K.Keutzer, S.Malik, A.Richard Newton, J.Rabaey, A.Sangiovanni-Vincentelli. “System-Level Design: Orthogonalization of Concerns and Platform-Based Design”. IEEE TCAD, Dec. 2000.

[Kru 92] C.W. Krueger, “Software Reuse,” ACM Computing Surveys, Vol. 24, No. 2, June 1992.

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

[Len 00] C.K.Lennard, P.Schaumont, G. de Jong, A.Haverinen, and P.Hardee, “Standards for System-level Design: Practical Reality or Solution in Search of a Question,” In Proc. Design Automation and Test in Europe, March 2000.

[Lyo 01] D.Lyonnard, S.Yoo, A.Baghdadi, A.Jerraya. “Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip”. Design Automation Conf., June 2001.

[Lyo 03] D.Lyonnard, “Approche d’assemblage systématique d’éléments d’interface pour la génération d’architecture multiprocesseurs. ” PhD Thesis, April 2003. Available at : **http://tima.imag.fr/publications/publications.asp**

[Mel 02] B.A.Mello, F.R.Wagner, "A Standardized Co-Simulation Backplane," In: M.Robert et al. (eds), SOC Design Methodologies. Kluwer Academic Publishers, 2002. pp 181-192.

[Mes 00] D.J.G.Mestdagh, M.R.Isaksson, P.Odling, “zipper VDSL: a solution for robust duplex communication over telephone lines,” IEEE Communication magazine, p. 90-96, May 2000.

[Mic 02] G.de Micheli, L.Benini. "Networks-on-Chip: A New Paradigm for Systems-on-Chip Design". Design Automation and Test in Europe Conf., 2002.

[Moo 98] Phillip R. Moorby, Donald E. Thomas, "The Verilog Hardware Description Language", May 1998, Hardcover.

[Nic 02] G.Nicolescu, "Spécification et validation des systèmes hétérogènes embarqués." PhD Thesis, November 2002. Available at : **http://tima.imag.fr/publications/publications.asp**

[Nil 99] M.O'Nils, "Specification, Synthesis, and Validation of Hardware/Software Interfaces," PhD thesis, Department of Electronics, Electronic System Design, Royal Institute of Technology, Electrum 229, Isafjordsgata 22-26, S-164 40 Kista, Sweden, 1999.

[Nil 99bis] M.O'Nils, and A.Jantsch, "Operating System Sensitive Device Driver Synthesis from Implementation Independent Protocol Specification," In Proc. Design Automation and Test in Europe, March 1999.

[OACC] Si2, OpenAccess. Available at : **http://www.si2.org/Projects/openaccess.htm**

[Obe 99] J.Oberg, "ProGram: A Grammar-Based Method for Specification and Hardware Synthesis of Communication Protocols," PhD thesis, Department of Electronics, Electronic System Design, Royal Institute of Technology, Electrum 229, Isafjordsgata 22-26, S-164 40 Kista, Sweden, 1999.

[Obj 00] ObjecTime. Available at: **http://www.objectime.com/**

[Ocor] OpenCore, Protocol International Partnership. Available at **http://www.ocpip.org/home**.

[Ocp 03] Open Core Protocol. Available at: **http://www.ocpip.org**

[OLA] Si2, OLA (Open Library API). Available at: **http://www.si2.org/ola**

[Omg 97] Object Management Group, CORBA services; Common Object Services Specification, Technical Report, OMG, July 1997.

[Ope 00] Synopsys, Mentor Graphics, "OpenMore," 2000. Available at: **http://www.OpenMORE.com**.

[Pas 98] R.Passerone, J.A.Rowson, A.Sangiovanni-Vincentelli. "Automatic Synthesis of Interfaces between Incompatible Protocols". Design Automation Conf., June 1998.

[Pas 02] S. Pasricha, "Transaction Level Modeling of SoC using SystemC 2.0," Synopsys User Group Conference (SNUG'02), May 2002, Bangalore.

[PARA] The Paradise Design Environment, C-lab, Germany. Available at : **http://www.c-lab.de/home/en/f-e-projects/paradise/**

[PIBus] Open Microprocessor System Initiative, PI Bus VHDL Toolkit, Version 3.1.

[Pim 01] A.D.Pimentel, et al. "Exploring embedded-systems architectures with Artemis," IEEE Computer, Vol. 34, p. 57-63, 2001.

[Pto 02] Ptolemy project, 2002. Available at: **http://ptolemy.eecs.berkeley.edu**

[Ram 00] F.J.Rammig. "Web-based System Design with Components off the Shelf," FDL'2000.

[Raw 97] J.A.Rawson and al. , "Interface Based Design," In Proc. Design Automation Conference, 1997.

[RMS] Reuse Management System (RMS), FZI. Available at : **http://fzi.de**

[Roa 01] International Technology Roadmap for Semiconductors. Available at: **http://public.itrs.net/Files/2001ITRS/FEP.pdf**

[Row 97] J.Rowson, A.Sangiovanni-Vincentelli. "Interface-based Design," Design Automation Conf., June 1997.

[Sch 99] P.Schaumont et al. "Hardware Reuse at the Behavioral Level," Design Automation Conf., June 1999.

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

[See 02] R.Seepold et al. "A Qualification Platform for Design Reuse," ISQED'2002.

[See 99] R.Seepold, L.Rosenberg, M.Genoe, and G.Matthew, "Special Session – Virtual Socket Interface Alliance," In Proc. Design Automation and Test in Europe, 1999.

[Sha 02] J.Shandle, G.Martin. "Making Embedded Software Reusable for SoCs," EEDesign, March 1th 2002.

[Slog] Superlog. Available at: **http://www.superlog.org**

[Smi 98] J.Smith, G.deMicheli. "Automated Composition of Hardware Components," Design Automation Conf., June 1998.

[SOC] SOCworks, Sonics. Available at: **http://www.socworks.com**

[Son 00] Sonics, Inc. "Sonics μNetworks, Technical Overview," June 2000. Networking Technology Datasheets, available at: **http://www.sonicsinc.com/Documents/Overview.pdf**

[Son 03] Sonics SiliconBackplane MicroNetwork. Available at: **http://www.sonicsinc.com**

[SRMIF] OVI Accelera, Semantic Reference Model Intermediate Format. Available at: **http://www.eda.org/arch**

[Syn 02] Synopsys, Inc., « Eaglei ». Available at: http://www.synopsys.com/products/hwsw/eagle_ds.html**.**

[Syn 97] Synopsis. January 1997, COSSAP reference manuals, Synopsys.

[Sys 00] Synopsys, Inc., "SystemC". Available at: **http://www.systemc.org/**

[Sys 01] Functional Specification for SystemC 2.0, Version 2.0-M, January 17th 2001. Available at: **http://www.systemc.org/**

[Tam 96] K.Tammemae, M.O'Nils, A.Jantsch, and A.Hemani, "Akka: a toolkit for cosynthesis and prototyping," In IEE Digest No.96/036 of Colloquium on Hardware-Software Co-synthesis for Reconfigurable Systems, p. 81-88, 1996.

[Uml 02] Unified Modelling Language UML. Available at: **http://www.rational.com/uml**

[Vah 95] F.Vahid and D.Gajski, "SLIF: A Specification-Level Intermediate Format for system design," In Proc. European Design and Test Conf., p. 116-123, March 1995.

[Vah 98] F.Vahid and T.Givargis, "Integrating Cores into System-level Specification," In Proc. Int'l Symposium on System Synthesis, December 1998.

[Val 01] C.A.Valderrama, A.Changuel, P.V.Vijaya-Raghavan, M. Abid, T.Ben Ismail, and A.A.Jerraya, "A Unified Model for Co-simulation and Co-synthesis of Mixed Hardware/Software Systems," p 579-583. Morgan Kaufmann Publishers, 2001.

[VCT] VSI Alliance, "VCT 1: Virtual Component Transfer Specification Version 2.1," January 2001. Available at: **http://www.vsi.org/**

[Ver 96] S.Vercauteren, B.Lin, and H.de Man, "Constructing Application-specific Heterogeneous Embedded Architectures from Custom HW/SW Application," In Proc. Design Automation Conf., June 1996.

[Ver 98] S.Vercauteren, "Hardware/Software Co-design of Application Specific Heterogeneous Architectures," PhD thesis, Katholieke Universiteit Leuven, Fakulteit Toegepaste Wetenschappe, Departement Elektrotechnick (ESAT), Divisie INSYS, Kard. Mercielaan 94, B-3001 Leuven, België, December 1998.

[Ver 99] S.Vercauteren, J.van der steen, and D.Verkest, "Combining Software Synthesis and Hardware/Software Interface Generation to meet Hard Real Time Constraints," 1999.

[Vor 01] A.Vörg, N.M.Madrid, W.Rosenstiel, R.Seepold, "IP Qualification of reusable designs," FZI Forschungszentrum Informatik, Microelectronic System Design, Karlsruhe, Germany. Available at: **http://www.fzi.de/**.

[VSIA] Virtual Socket Interface Alliance. Available at: **http://www.vsi.org**

[Vxw] VxWorks. Available at: **http://www.windriver.com**

[Wil 99] P.A. Wilsey, "Web-based Analysis and Distributed IP," In Proc. of the 1999 Winter Simulation Conference.

[XML 00] XML 1.0 Specification, 2d edition, W3C Recommendation, October 2000, available at: **http://www.w3c.org/XML**

[Yoo 01] S.Yoo, G.Nicolescu, D.Lyonnard, A.Baghdadi and A.A.Jerraya, "Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design," Int. Symposium on HW/SW Codesign (CODES) 2001.

[Yoo 03] S.Yoo and A.A.Jerraya, "Introduction to Hardware Abstraction Layer for SoC," Proc. DATE 2003.

[Zia 02] T.Ziadi, B.Traverson, J.M.Jézéquel, "From a UML Platform Independent Component Model to Platform Specific Component Models," Workshop in Software Model Engineering, Dresden, Germany, October 2002.

Glossaire

A

ASIC **Application Specific Integrated Circuit.**
Circuit intégré développé spécifiquement pour une application.

API **Application Programming Interface.**
Interface logicielle permettant à une application d'accéder aux services proposés par le système d'exploitation. Une API, matérialisée par un ensemble de prototype de fonctions systèmes, fournit une abstraction de l'architecture matérielle aux différentes taches logicielles, en facilitant ainsi la portabilité.

ASAG **Application-Specific Architecture Generation.**
Outil de génération d'architecture spécifique à une application. Cet outil a été développé dans le cadre des travaux de thèse de Damien Lyonnard.

ASOG **Application-Specific Operating Systems Generation.**
Outil de génération de système d'exploitation spécifique à une application. Cet outil a été développé dans le cadre des travaux de thèse de Lovic Gauthier.

C

COLIF **COdesign Language Independent Format.**
Langage de représentation de la composition de systèmes hétérogènes en niveaux d'abstraction, en protocoles de communication et en langages de description.

CosimX **Cosimulation tool.**
Outil de génération de modèle de co-simulation spécifique à une application. Cet outil a été développé dans le cadre des travaux de thèse de Gabriela Nicolescu.

CPU **Central Processing Unit.**
Ce terme est utilise pour designer le processeur, c'est-à-dire l'unité centrale de traitement, d'un ordinateur.

CORBA **Common Object Request Broker Architecture.**
Standard définit par Object Management Group (OMG). Il offre des mécanismes permettant la communication, par envois de messages, entre des objets géographiquement distribués et provenant d'environnements différents.

CSP **Communication Sequential Processes.**
Langage formel utilise pour la description des systèmes parallèles.

D

DFU **Deframing/Framing Unit.**
Un sous-ensemble du modem VDSL.

DSL **Digital Subscriber Line.**

DSP **Digital Signal Processor.**
Processeur spécialisé pour le calcul d'algorithme de traitement du signal.

DTD **Document Type Definition.**
Fichier de définition de structure de donnée permettant l'extension de la syntaxe du langage XML.

F

FSM **Finite State Machine.**
Modèle de calcul définit par un ensemble d'états, un ensemble d'événements d'entrée,

un ensemble d'événements de sortie et une fonction de transition entre les états.

FIFO **First-In First-Out.**
Protocole de communication qui assure que les premières données envoyées sont les premières reçues.

FPGA **Field Programmable-Gates Array.**
Réseau pré-diffusé programmable par l'utilisateur : réseau pré-diffusé de portes logiques qui est destine à être programmé sur place, avant d'être utilisé pour une fonction particulière.

H

HAL **Hardware Abstraction Layer.**
Couche logicielle de bas niveau interfaçant les services de haut niveau, fournis par un système d'exploitation, aux ressources matérielles. Ces services sont ainsi indépendants de l'architecture. Cette couche fournis les pilotes et les contrôleurs pour la communication avec le matériel.

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

HW **Hardware.**
Terme anglo-saxon couramment utilisé pour designer les composants matériels et peu flexible de l'architecture d'un système.

I

IP **Intellectual Property.**
Elément (logiciel ou matériel), dont la réutilisation pour fabriquer de nouveaux systèmes est protégée par les règles de la propriété intellectuelle. Le fonctionnement de cet élément connu et documenté, mais sa structure interne est inconnu. Les termes cœur IP et bloc IP ont été construits en utilisant l'abréviation anglaise du terme intellectual property. Les termes bloc fonctionnel et composant virtuel (en anglais virtual component) sont parfois employés pour définir le cœur de propriété intellectuelle.

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

ISS **Instruction Set Simulator.**
Simulateur d'un processeur (ou jeu d'instruction) permettant d'émuler la fonctionnalité d'un processeur.

M

MA **Module Adaptor.**
Composant élémentaire dans la structure de l'interface matériel et généralisé pour l'interface de co-simulation. Ce composant est spécifique au module à intégrer dans le système pour la synthèse/co-simulation.

MCU **Micro-Controler Unit.**
Processeur ayant des performances, une consommation électrique et une occupation surfacique toutes modérées. Ils sont par conséquent fortement utilisés dans des applications embarquées où ils ont pour charges contrôler des accélérateurs matériels.

Middle **Mark-up internal data-description language extension.**
Adjonction au langage XML sous forme d'un DTD afin de définir de nouvelles structures de données telles que COLIF.

modem **Modulateur/démodulateur.**
Partie fonctionnelle d'une terminaison de communication, en charge d'implémenter la couche physique du protocole par modulation/démodulation de fréquence du signal porteur.

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

N

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

NoC **Network-on-Chip.**
Système monopuce organise comme un réseau de processeur.

O

OCP **Open Core Protocol.**

OS **Operating System.**
Un operating system ou système d'exploitation, est une couche logicielle permettant d'abstraire une architecture matérielle à la vue de l'application logicielle afin de gérer implicitement l'utilisation partagée des ressources, mais aussi de garantir la portabilité de l'application sur des architectures diverses.

P

PC **Personal Computer.**

PCA **Processor Centric Architecture.**

R

RISC **Reduce Instruction Set Computer.**
Famille d'architecture de processeur ayant un jeu simplifié et réduit d'instructions. Bien que leurs instructions soient moins puissantes que celles de la famille CISC, l'utilisation de processeur RISC est répandu dans les applications embarquées en raison de leur plus faible consommation. De plus une meilleure prédiction des temps d'exécution de leurs instructions permet de vérifier leur tenue aux contraintes « temps-réel » de l'application.

RTL **Register Transfert Level.**

S

SAP **Service Access Port.**

SE **Système d'Eploitation.**
Cf. OS.

SHM **Shared Memory.**
Ensemble de ressources de mémorisation à usage (ou accès) partagé.

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

SoC **System-on-Chip.**
Système monopuce, circuit intégrant sur une même puce divers composants tels que des processeurs, des mémoires, des accélérateurs matériels, etc.

SW **Software.**
Terme anglo-saxon couramment utilisé pour designer les composants logiciels.

T

TIMA **Techniques Informatiques et Microélectroniques pour l'Architecture des ordinateurs.**
Laboratoire de recherche de l'INP Grenoble.

TLM **Transaction Level Modeling.**
Un ensemble de bibliothèques construit au dessus de SystemC2.0, permettant la

modélisation des systèmes au niveau transaction. C'est un modèle proposé par ST Microélectronique.

U

UCOM **Communication Unit.**
Adaptateur de communication généré par l'outil GAUT.

UML **Unified Modelling Language.**
Langage de modélisation par objets de troisième génération, permettant de déterminer et de présenter les composants d'un système lors de son développement, ainsi que d'en générer la documentation. En présentant une description unifiée des concepts objets, le langage UML fait franchir le pas menant de la technologie des objets à la technologie des composants.

V

VCC **Virtual Component Codesign.**
Outil d'exploration d'architectures commercialisé par Cadence, Inc.

VCI **Virtual Component Interface.**
Modèle standard d'interface proposé par VSIA.

VDSL **Very high data rate Digital Subscriber Line.**
Standard de communication par paire torsadée.

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

VSIA **Virtual Socket Interface Alliance.**
Consortium d'industriels oeuvrant pour la définition d'un standard de modélisation des interfaces de composants matériels.

X

XML **eXtensible Mark-up Language.**
Langage à balises, extensible par l'adjonction d'un modèle de définition de document (DTD) définissant les nouvelles structures de données. Ce métalangage est standardisé par W3C

RESUME

La technologie de fabrication des circuits intégrés a permis de passer des composants spécifiques ASIC aux systèmes embarqués sur une seule puce (*SoC*). Ces systèmes sont construits par un assemblage de composants hétérogènes existants (préconçus). De plus, le flot de conception de SoC nécessite l'intégration de plusieurs outils provenant de différentes sources et ayant des domaines d'application variés, dans le but d'obtenir un flot de conception complet, ce qui n'existe pas aujourd'hui.

L'intégration de composants hétérogènes est très difficile. Elle requiert une adaptation de leurs interfaces au réseau de communication embarqué. Cette adaptation nécessite la construction d'adaptateurs divers et très sophistiqués. Ces derniers sont obtenus par un assemblage de composants d'interface élémentaires. L'intégration d'outils provenant de différentes sources dans un environnement de conception existant est aussi difficile. Elle nécessite une interopérabilité entre les différents outils dans le cadre d'un seul flot de conception complet.

La manipulation d'outils et de composants hétérogènes dans un flot complet de conception SoC est un travail fastidieux, source d'erreurs, et coûteux en terme de temps de conception. Vu la pression du temps de mise sur le marché, un environnement ouvert à l'intégration automatique d'outils et de composants logiciels/matériels est devenue cruciale.

La contribution de cette thèse concerne la construction d'un environnement de conception ouvert autour d'un format intermédiaire. Cet environnement permet l'intégration d'outils selon un modèle bien défini. Il permet aussi l'intégration automatique de composants logiciels/matériels selon un flot générique et des techniques de composition. Les concepts proposés ont été validés sur des études de cas différentes : l'intégration de l'outil VCC de Cadence et l'intégration d'un IP de communication décrit à un haut niveau d'abstraction dans le flot de conception ROSES.

MOTS CLES

Flot de conception des systèmes monopuce, réutilisation de composants, hétérogénéité, interopérabilité entre les outils de conception (format intermédiaire), conception à base de composants (abstraction et techniques de composition).

TITRE EN ANGLAIS

Design tools and hardware/software components integration models for heterogeneous embedded systems design.

ABSTRACT

The continuous evolution of integrated circuits technology is challenging designers to shift from application-specific components (ASIC) to full systems on a single chip (SoC). In order to manage the complexity of these SoC systems, they are built by assembling pre-designed components from different providers. Moreover, a complete SoC design flow requires the integration of several design tools from different providers and for different application domains.

The integration of these heterogeneous components into a single system is very difficult, requiring an adaptation of their interfaces to the embedded communication network. This adaptation often needs sophisticated interface sub-systems that can also be constructed by assembling pre-designed interface components. Integrating design tools from different providers into a complete SoC design flow is also a difficult task, requiring seamless interoperability among the different tools.

Handling tool and component integration on a complete SoC design flow is a fastidious, error-prone, and time-consuming manual work. Due to the always increasing time-to-market pressure, an open environment for the automation of tool and component integration is becoming crucial.

The main contribution of this thesis is the definition of an open environment for component/tool integration built around an intermediate format. This environment eases design tool integration according to a well-defined model. It also defines a generic design flow and composition techniques for hardware/software component integration. The proposed concepts were validated using two case-studies: the integration of Cadence VCC design tool and of a communication IP, described in a high abstraction level, into the ROSES design flow.

KEY WORDS

SoC design flow, IP components re-use, heterogeneity, design tool interoperability (intermediate format), component-based design (component abstraction and composition techniques).

INTITULE ET ADRESSE DU LABORATOIRE

Laboratoire TIMA, 46 avenue Félix VIALLET, 38031 GRENOBLE CEDEX, FRANCE.

ISBN : 2-84813-032-6 (Broché) ISBNE : 2-84813-033-4 (Format Electronique)