UNIVERSITE DE SAAD DEHLEB DE BLIDA

Faculté des sciences
Département d'informatique

MEMOIRE DE MAGISTER

Option: Ingénierie des systèmes et de la connaissance

DEVELOPPEMENT D'UNE APPROCHE DE COSYNTHESE D'INTERFACES DE COMMUNICATIONS POUR LE CODESIGN

Par

BERRHAIL Fouaz

Devant le jury composé de :

Mme H. ABED	Maître de Conférences Université de Blida	Présidente
Mlle K. BENATCHBA	Maître de Conférences INI, Alger	Examinatrice
Mme F. SITAYEB	Docteur INI, Alger	Examinatrice
Mme S. OUKID	Maître de Conférences Université de Blida	Invitée
M.M. KOUDIL	Maître de Conférences INI, Alger	Rapporteur

Blida 2006

A ma très chère mère.
A mon cher père.

ملـخـص

مع ارتفاع نسبة تعقد الأنظمة المختلطة، فان توليد وسائط الاتصال في سير التصميم المتنافس (Codesign) أصبح عملية معقدة.ولقد أصبح من الضروري إيجاد طرق آلية جديدة ذات مستوي عالي، تسمح بالتحكم في التعقيد، وتساعد المصممين في توليد وسائطهم.

في الأنظمة المختلطة (برنامج/عتاد) يمكننا تمييز ثلاثة أنواع من الاتصالات: اتصال بين برنامج وبرنامج، اتصال بين عتاد وعتاد، اتصال بين برنامج وعتاد. النوع الأول و الثاني يطرحان اشكالات قليلة لأنهما يتطلبان آلية بسيطة من اجل الاتصال. على ا لعكس النوع الأخير فانه يطرح عدة إشكالات، بسبب عدم تجانس المركبات المستعملة في القسمين (عتاد/برنامج)، طبيعة المعالجات، اختلاف السرعات...الخ. إشكال آخر يطرح من اجل عملية توليد ا لوسائط و المتمثل في عدم معرفة بعض تفاصيل الوسيطة إلا بعد إجراء التقسيم.

أ ردنا من خلال هذه الرسالة تقديم طريقة آلية ذات مستوى عالي تسمح بتوليد وسائط الاتصال لأجل التصميم المتنافس، و التي تتمثل في إجراء تصفيات متتالية لنص تمييز .Java ولقد قمنا بحل بعض الإشكالات التي صادفناها خلال هته العملية، حلول اقترحت: من اجل تحليل نص التمييز، وصف المركبات البنيوية، و مختلف مراحل توليد الوسائط.

الكلمات الرئيسية: Codesign, Cospecification, Analysis, Interface cosynthesis

RESUME

Au fur et à mesure que la complexité des systèmes mixtes croît, la cosynthèse d'interfaces de communication pour le codesign devient un processus complexe. Il devient nécessaire de définir de nouvelles méthodes automatiques de haut niveau permettant de maîtriser la complexité et d'aider les concepteurs à synthétiser ses interfaces.

Dans les systèmes mixtes (matériel/logiciel), il est possible de distinguer trois types de communication: communication logiciel/logiciel, communication matériel/matériel, et communication logiciel/matériel. Le premier et le deuxième type de communication posent moins de problèmes puisqu'ils nécessitent un mécanisme simple pour la communication. Par contre, le dernier type pose plus de problèmes, en raison de l'hétérogénéité des composants utilisés pour les deux parties, la nature des traitements, les différences de vitesses, etc. Un autre problème posé par l'étape de cosynthèse d'interface est dû au fait que certains détails de l'interface ne sont connus qu'à l'issue de l'étape de partitionnement.

Le présent mémoire introduit une approche automatique de haut niveau pour la cosynthèse d'interfaces de communication dans le codesign, elle consiste à procéder par affinements successifs sur le code de cospécification Java. Nous tentons de résoudre un certain nombre des problèmes rencontrés durant la cosynthèse. Des solutions sont proposées pour l'analyse de code, la description des composants architecturaux, et les différentes étapes de cosynthèse d'interfaces.

Mots clés : Codesign, Cospécification, Analyse, Cosynthèse d'interfaces.

ABSTRACT

As the complexity of the mixed systems grows, communication interface synthesis during the codesign process gets harder and harder. It becomes necessary to define new high-level automated methods that allow mastering complexity and helping designers to synthesize their interfaces.

In mixed systems (made of hardware and software), it is possible to distinguish three types of communication: communication between software and software, communication between hardware and hardware, and communication between software and hardware. The first and the second communication types raise less problems since they require simple communication mechanisms. The last type raises more problems, because of the heterogeneity of the components used for the two parts, the nature of the treatments, the differences of speeds… Another problem in interface cosynthesis is due to the fact that some interface details are only known at the end of the partitioning process.

This document introduces a high-level automatic approach for the communication interface synthesis in Codesign. It proceeds by successive refinements on the Java code of the cospecification. We tried to solve some of the synthesis problems. Solutions are proposed for: code analysis, architectural component description, and the different phases of interface cosynthesis.

Keywords: Codesign, Cospecification, Analysis, Interface cosynthesis.

REMERCIEMENTS

Je tiens à remercier en premier lieu mon dieu « *ELLAH* » tout puissant qui m'a accordé la volonté et le courage pour la réalisation de ce projet.

Je tiens à remercier chaleureusement monsieur KOUDIL Mouloud, Maître de conférences à l'institut national d'informatique d'Alger, mon promoteur et le directeur de ce travail, pour ses larges expériences dans le domaine de codesign, son gentiment et amabilité, ses conseils et disponibilité, et pour avoir accepté de diriger ce travail et de corriger ce document.

J'adresse mes remerciements à Mme H. ABED, Maître de Conférences à l'Université de Blida, de m'avoir fait l'honneur de présider mon jury de mémoire.

Pour m'avoir fait l'honneur de participer au jury de mémoire, je remercie Mlle K. BENATCHBA Maître de Conférences à l'Institut National d'Informatique Alger, Mme F. SITAYEB Docteur à l'Institut National d'Informatique d'Alger, Mme S. OUKID Maître de Conférences à l'Université de Blida, la directrice du laboratoire de recherche et de développement des systèmes informatisés (LRDSI).

Je remercie aussi monsieur Benaouda Abd el hafid, mon promoteur de PFE et le directeur de centre de calcul de l'Université de Ferhat Abbas Sétif, de son encouragement et soutien.

Un immense merci à mes parents (ma mère et mon père), mes frères, mes sœurs et mes proches qui m'ont apporté leur encouragement et soutien tout au long de mes études.

Je remercie aussi l'ensemble des enseignants, qui ont contribué à notre formation.

Enfin, mes remerciements vont à ceux qui ont participés, plus ou moins indirectement à mon mémoire.

Merci à mes intimes amis : Kadour Bachir, Tabet Adel.

Merci à mes collègues de post graduation (Université de Blida) : B.Maamar, H.Hamza, F.Messaouda, B.Djawida, sans oublier Amina et Mehdia.

Merci à mes amies : Kamel, Anis, Abd el salem, Yazid, Ramdan, Hocine, Farid, Hakim, Abd el kader.

Merci à tous ce qui nous ont aidé de près ou loin.

TABLE DES MATIERES

LISTE DES ILLUSTRATIONS

LISTE DES TABLEAUX

INTRODUCTION

La conception des systèmes mixtes (matériels/logiciels) est un processus difficile: la complexité croissante des applications, l'évolution rapide des technologies de réalisation des circuits intégrés nécessitent l'utilisation de nouvelles méthodes de conception qui doivent être capables de tenir compte de nombreuses contraintes : temps, flexibilité, fiabilité, complexité, consommation.

Les techniques classiques de la conception des systèmes (mixtes) contenant du matériel et du logiciel consistaient à séparer la partie matérielle de la partie logicielle très tôt dans le cycle de conception (indépendance de deux parties jusqu'à la phase d'intégration). Les concepteurs du matériel n'avaient donc que peu de connaissances sur la partie logicielle. Il en était de même pour les concepteurs du logiciel. Souvent, de nombreux problèmes étaient rencontrés à la phase d'intégration. Ce type des méthodes pose souvent quelques problèmes, du fait de la complexité croissante des systèmes, l'hétérogénéité des systèmes, la flexibilité des systèmes, la simulation impossible, la difficulté de corriger les erreurs et de maintenir le système en fonctionnement normal, etc.

Tous ces problèmes et d'autres mettent l'accent sur la nécessité de disposer d'une méthodologie qui respecte les exigences de plus en plus grandes en performance, délais, coûts, testabilité, etc. Cette méthodologie est appelée Codesign "*Concurrent design*" ou la conception concurrente. Elle consiste à prendre en considération l'interaction entre les deux parties du système (matérielle et logicielle) très tôt et durant tout le cycle de conception, et retarder la séparation des deux parties le plus tard possible.

Le processus de conception des systèmes mixtes procède par étapes. Il commence par la spécification et la modélisation du comportement et des fonctionnalités du système, ainsi que son architecture. Ensuite une étape de partitionnement sert à déterminer les parties du système qui doivent être implantées en matériel et celles qui doivent l'être en logiciel. Suit une étape de synthèse de la partie matérielle sous forme de circuit ASIC (Application Specific Integrated Circuit), FPGA (Field Programmable Gate Array), de synthèse de la partie logicielle sous forme

un programme exécutable sur un processeur logiciel, et de synthèse de la communication et des interfaces de communication entre les deux parties (Matérielle et logicielle). La dernière étape est la validation pour garantir que le système conçu réalise de façon correcte les fonctionnalités attendues tout en respectant les contraintes fixées.

Les différentes parties du système partitionné sont, en général, indépendantes du point de vue traitement, mais il est fréquent qu'elles aient à échanger des informations à travers des mécanismes dédiés à la communication [46]. Ces mécanismes peuvent être en matériel (bus, contrôleurs...etc.) ou en logiciel (pilotes, protocoles…etc.). Pour assurer la communication entre les différentes parties du système, il faut qu'elles parlent le même langage, et utilisent un protocole de communication qui assure l'échange des informations entre elles. Il est donc nécessaire de générer des interfaces qui adaptent la communication entre ces parties.

L'opération de cosynthèse d'interfaces de communication pour le processus de codesign constitue une étape primordiale. Elle consiste à établir des liens de communication entre les composants (matériels et logiciels) du système mixte. Dans les systèmes mixtes, il est possible de distinguer trois types de communication: communication logiciel/logiciel, communication matériel/matériel, communication logiciel/matériel. Le premier et le deuxième type de communication posent moins de problèmes puisqu'ils nécessitent un mécanisme simple pour la communication. Par contre, le dernier type pose plus de problèmes en raison de l'hétérogénéité des composants utilisés pour les deux parties, la nature des traitements, les différences de vitesses, etc. Un autre problème posé par l'étape de cosynthèse d'interface est dû au fait que certains détails de l'interface ne sont connus qu'à l'issue de l'étape de partitionnement (les goulots d'étranglement ne peuvent être identifiés que lors d'une cosimulation), et donc un feed-back est nécessaire, ce qu'il complique la cosynthèse. Les concepteurs sont confrontés à la difficulté que constitue la tâche de synthèse d'interfaces et mettent l'accent sur la nécessité de disposer d'outils automatiques permettant d'aider le concepteur dans sa tâche.

Le travail introduit dans ce mémoire, présente une nouvelle approche de cosynthèse d'interfaces de communication pour le codesign. Notre objectif est le développement d'une approche automatique de haut niveau pour la cosynthèse d'interface de communication. L'approche de cosynthèse développée est basée sur la notion de ports virtuels et de primitives virtuelles de communication (*Send*, *Receive)*. Elle procède en remplaçant les appels aux

méthodes et leurs paramètres par des appels aux primitives virtuelles (pour les méthodes) et des ports virtuels (pour les paramètres). Elle procède ensuite par affinements successifs sur le code Java de la cospécification des entités de l'application (après une modification dans le code initial). Il est nécessaire de prendre en charge la quantité importante d'informations hétérogènes manipulées tout au long du processus de cosynthèse. Cette approche est basée sur un modèle et un formalisme de cospécification orienté objet. La partie système est décrite en langage Java, et la partie de l'architecture cible est décrite d'une manière graphique à l'aide de composants et de protocoles stockés dans des bibliothèques.

Notre approche permet de rendre l'opération de la synthèse indépendante vis-à-vis de l'architecture cible et des protocoles de communication et ceci grâce à l'utilisation de la notion de bibliothèques de composants architecturaux et de protocoles.

Notre contribution dans ce travail consiste à :

- Développer et réaliser une technique et un environnement permettant d'analyser le code de la cospécification orienté objet (Java), afin d'extraire toutes les informations qui caractérisent la communication entre objets. Ces informations sont représentées à la fin de l'opération de l'analyse par deux graphe : graphe hiérarchique (des classes) qui représente la hiérarchie des classes de la cospécification; et graphe d'instance (fonctionnel) : qui représente les liens de communication entre objets.
- Développer et réaliser une technique et un environnement qui permet la description des composants architecturaux et des protocoles de communication. Nous avons fait le choix d'offrir aux concepteurs la possibilité de composer ses architectures cibles en même temps que l'application qui s'exécute dessus. L'outil développé est basé sur des bibliothèques de composants architecturaux et de protocoles de communication réutilisables.
- Développer une nouvelle approche de cosynthèse des interfaces de communication pour le processus de codesign, qui est basée sur les informations fournies par les deux outils d'analyse et de description des composants architecturaux.

Notre mémoire est structuré en deux parties: la première constitue une étude bibliographique sur la cosynthèse d'interface de communication. Elle comporte quatre chapitres.

Le premier chapitre présente des généralités sur le processus de conception des systèmes mixte (codesign), motivation, avantages et domaine d'application, etc.

Le deuxième chapitre détaille les différentes étapes de processus de codesign afin de situer notre travail.

Le troisième chapitre est consacré aux différents problèmes liés à la cosynthèse d'interfaces de communication.

Le quatrième chapitre présente un état de l'art sur la cosynthèse d'interfaces de communication. Il présente certaines des principales approches citées dans la littérature.

La deuxième partie présente l'approche proposée pour la cosynthèse d'interfaces, elle comporte cinq chapitres.

Le cinquième chapitre présente le schéma général de notre approche, le modèle et le formalisme de cospécification utilisés.

Le sixième chapitre présente l'environnement développé pour l'analyse du code de la cospécification.

Le septième chapitre illustre l'outil développé pour la description de l'architecture cible et des protocoles de communication.

Le huitième chapitre détaille les différentes étapes de l'approche proposée concernant la synthèse d'interfaces.

Le neuvième chapitre présente un jeu de test et la mise en œuvre de l'approche.

CHAPITRE 1
CODESIGN MATERIEL/LOGICIEL

1.1 Introduction

Un système mixte est souvent formé de deux parties: une partie matérielle est une autre logicielle en étroite interaction. Si la conception du système est totalement en matériel, elle peut alors présente l'inconvénient du manque de flexibilité, du temps développement important, du coût élevé et du faible taux de réutilisation des composants. Par contre, la réalisation du système en logiciel qui est basé sur l'utilisation des processeurs à usage général, permet d'augmenter la flexibilité, mais présente l'inconvénient d'être lente.

Afin de concevoir ces systèmes embarqués (intégrés dans l'environnement qu'ils contrôlent), l'utilisation conjointe de processeurs d'usage général, dont les performances atteignent aujourd'hui des niveaux très élevés [9], et de circuits spécialisés chargés de la réalisation des fonctions très spécifiques (opérateurs particuliers de traitement de signaux, des interfaces rapides ou analogiques, des protocoles complexes, …etc.), il est nécessaire de développer des méthodologies de conception nouvelles permettant la mise en oeuvre du système mixte matériel/logiciel.

Avant l'apparition du codesign, les techniques utilisées pour la conception des systèmes (mixtes) contenant du matériel et du logiciel consistaient à séparer la partie matérielle de la partie logicielle très tôt dans le cycle de conception (indépendance de deux partie jusqu'à la phase d'intégration). Les concepteurs du matériel n'avaient donc que peu de connaissances sur la partie logicielle. Il en était de même pour les concepteurs du logiciel. Souvent, de nombreux problèmes étaient rencontrés à la phase d'intégration [46].

La tendance actuelle pour la conception des systèmes mixtes matériels/logiciels consiste à prendre en considération l'interaction entre les deux parties matérielle et logicielle très tôt et durant tout le cycle de développement, et retarder la séparation des deux parties le plus tard possible. Cette approche est appelée conception conjointe Matérielle/logicielle ou Codesign.

1.2 Technique classique de conception des systèmes embarqués

1.2.1 Définition d'un système embarqué

Un système embarqué est un "système digital qui représente un sous-système dans un système global et qui offre un service spécifique à ce système" [67, 22].

Actuellement, les systèmes embarqués sont de plus en plus présents comme constituants de produits industriels ou commerciaux (dans le four micro-onde, le contrôleur d'injection et d'allumage d'une voiture, le robot industriel, le téléphone cellulaire ou le système de pilotage automatique d'un avion …etc).

1.2.2 Problèmes de conception d'un système embarqué

Les méthodes de conception traditionnelles proposent aux différentes équipes de développer chaque partie du système séparément et de faire l'intégration de l'ensemble lors de la construction du prototype [34].

D'après [29], le flot classique de conception part d'une spécification souvent informelle du système. Il distingue immédiatement les parties logicielles des parties matérielles. Ces parties sont développées indépendamment l'une de l'autre par des équipes différentes. A la fin, une équipe d'intégration assemble les parties, ce qu'il pose souvent des problèmes d'incompatibilité. Cette intégration donne directement un prototype à tester. En cas d'erreur, il peut être nécessaire de recommencer complètement le flot.

Ce type d'approche peut avoir pour conséquence une augmentation du temps et du coût de conception du système, à cause des problèmes de validation du système entier, et des problèmes d'interfaces entre les différentes parties au moment de la construction du prototype [34]. La première difficulté rencontrée par les concepteurs est la complexité des systèmes actuels (Les systèmes embarqués sont de plus en plus complexes) ils doivent fournir des fonctionnalités de plus en plus élaborées, et peuvent donc être constitués de composants logiciels ou matériels variés.

Les approches classiques de conception reposent généralement sur une décomposition initiale de la spécification en sous-systèmes qui sont chacun optimisés localement et validés

ensuite globalement par co-simulation. Cette approche rend difficile la vérification fonctionnelle et temporelle du système [2].

D'autres problèmes peuvent être rencontrés par les concepteurs des systèmes embarqués:

- Difficulté de corriger les erreurs et de vérifier le système.
- Difficulté de maintenir le système en fonctionnement normal.
- Préserver des délais de mise sur le marché les plus courts possibles (compétitivité).
- La robustesse du système à concevoir.

La modélisation de systèmes réactifs est donc une activité difficile. Elle n'est pas dépourvue de risques d'erreurs par les méthodes usuelles.

Les composants réutilisables permettent d'éviter aux concepteurs d'avoir développer à chaque fois toutes les fonctionnalités requises. Cependant, pour qu'il ait un gain de temps l'intégration de ces composants au reste du système ne doit pas demander trop d'effort aux concepteurs.

Une autre voie pour surmonter la complexité consiste à élever le plus possible le niveau d'abstraction des descriptions des systèmes à concevoir. Cette méthode, peut donner aux concepteurs une vision plus générale et plus simple des systèmes.

Donc il est nécessaire de suivre des méthodologies et des outils plus efficaces permettant la conception conjointe, pour guider le passage des descriptions de haut niveau aux réalisations finales. Il est nécessaire aussi de tendre vers des approches globales de conception et d'optimisation multi-critères pour réduire le fractionnement actuel du processus de conception [2].

1.3 Comparaison entre conception traditionnelle et le codesign matériel/logiciel

Les Méthodes de conception classiques ou traditionnelles des systèmes imposent du séparer le matériel du logiciel très tôt dans le cycle de conception. Les techniques de conception des deux parties du système (Matériel et logiciel) restent indépendantes [46], n'autorisant l'interaction que lors de la phase d'intégration.

Ce type de conception pose souvent quelques problèmes [47]:

- Mauvaise appréciation des besoins en ressources pour les concepteurs matériels.
- Mauvaise exploitation des ressources pour les concepteurs logiciels.
- Modifications majeures matérielles et/ou logicielles à la phase d'intégration.

La conception conjointe ou concurrente, encore nommée "codesign matériel/logiciel", est donc née de la nécessité de développer des applications respectant des exigences de plus en plus grandes en performances, coût, délais ou testabilité [46]. Il est nécessaire, dès les premières étapes de la conception, de tenir compte des interactions entre le matériel et le logiciel.

Les figures (1.1 et 1.2) illustrent les approches de conception traditionnelle et conjointe pour les systèmes mixtes.

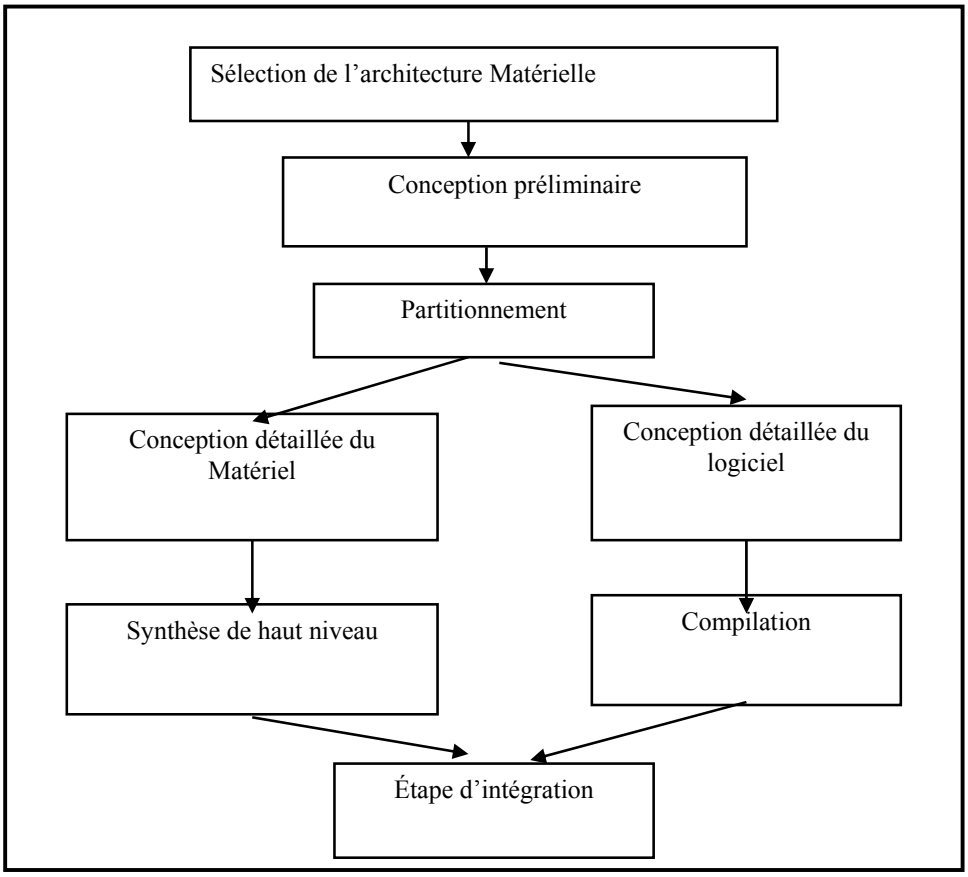

Figure 1.1: Approche de conception traditionnelle [46]

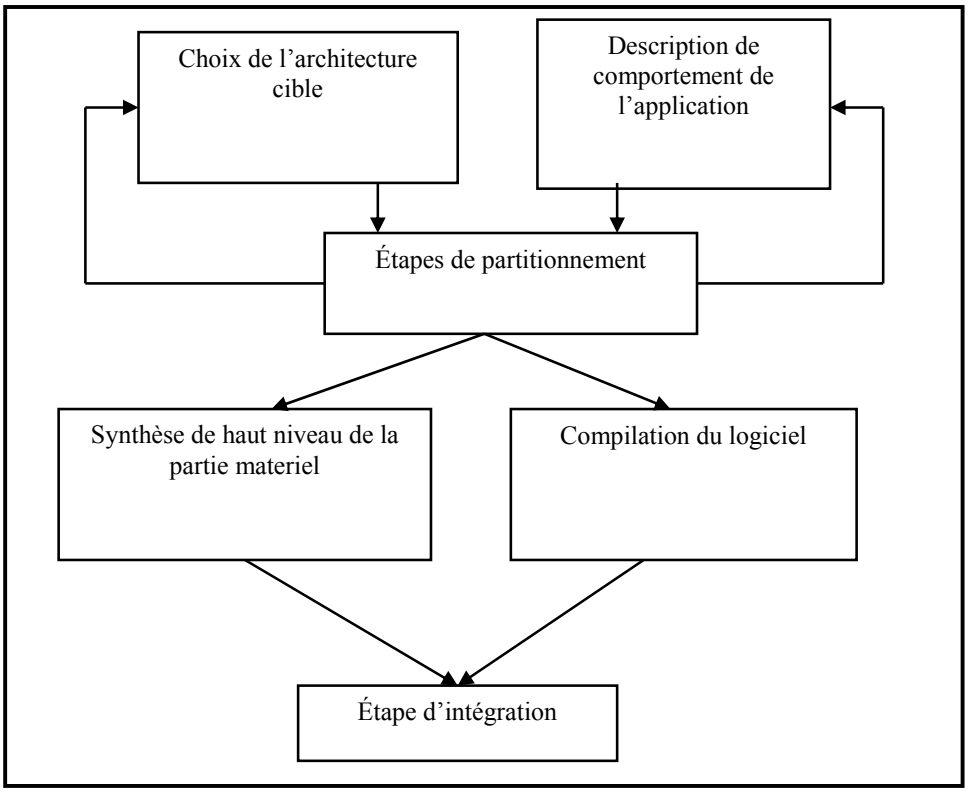

Figure 1.2: Approche de conception conjointe [46]

L'inconvénient des méthodes traditionnelles de codesign est leur trop grande spécialisation: une méthode est faite pour un type de problème [38].

Le développement conjoint du logiciel et du matériel est un domaine relativement récent. Pour maîtriser la complexité croissante des systèmes et pour répondre aux critères de performances attendus, il est important d'aborder la conception des systèmes mixtes en terme de conception conjointe de matériel/logiciel [31, 7]. Ce type d'approche agit aussi au plus haut niveau d'abstraction, c'est à dire le niveau système. Le besoin d'un niveau d'abstraction plus élevé, notamment le niveau système, vient du fait que les systèmes deviennent de plus en plus complexes [34].

Les buts de codesign sont multiples puisqu'il permet de:

- Trouver le meilleur compromis entre la réalisation logicielle et réalisation matérielle.
- Augmenter la productivité en permettant la conception simultanée du logiciel et du matériel (concurrent design).

1.4 Quelques définitions pour le codesign

D’après différents chercheurs qui travaillent dans le domaine de la conception des systèmes mixtes, on peut citer plusieurs définitions:

- D’après Stoy [67], le but du codesign est d’intégrer la conception du matériel et du logiciel pour former un processus unique de conception. Seul le comportement de tout le système intéresse l’utilisateur. Le codesign permet de trouver des moyens d’améliorer les techniques de conception des systèmes où le matériel et le logiciel travaillent en étroite collaboration pour résoudre des problèmes qui ont le plus souvent des contraintes imposées sur leurs solutions.
- Selon Kalavade [43], la tâche de codesign consiste à produire un système matériel/logiciel optimal, qui répond aux spécifications données, en respectant les contraintes de conception (contraintes temps-réel, de performance, de vitesse, de surface, de taille du code, d'espace mémoire, et de consommation). La doctrine clé dans le codesign est d’éviter la séparation entre le matériel et le logiciel conçus trop tôt.
- D'après Edwards [24], le but principal du codesign est de produire des systèmes contenant un équilibre optimal entre les composants matériels et logiciels qui travaillent conjointement pour réaliser un comportement spécifié et répondre à des contraintes identifiées - y compris satisfaire des objectifs de performance critiques.
- De Micheli [22] affirme que le codesign matériel/logiciel signifie répondre aux objectifs au niveau système en exploitant la synergie du matériel et du logiciel à travers la conception conjointe.
- D’après [46] le codesign matériel/logiciel est une Méthodologie unique de conception intégrant différents composants d'un système mixte en respectant les fonctionnalités et les contraintes associées.
- Selon [69], la conception concurrente matérielle/logicielle est une approche qui intègre dans un même environnement la conception du matériel et celle du logiciel.
- Pour [29] le codesign a pour but de développer conjointement les diverses parties d’un système hétérogène (logiciel, électronique, mécanique, etc.). C’est pourquoi une description globale du système est nécessaire.
- D’après [42], Le codesign est l’une des techniques les plus intéressantes car elle s’efforce de mettre en place une vraie méthode de conception simultanée du matériel et du logiciel. Son

émergence est due à la grande et croissante ressemblance de la conception des systèmes numériques avec la conception du logiciel.

Nous pouvons donc conclure, que le codesign est une approche qui permet d'intégrer dans une méthode de conception unique les différentes composants d'un système mixte (composants matériels, composants logiciels, ou composants des communications) et qu'elle prend en compte principalement l'interaction entre ses composants, afin d'obtenir des produits respectant les fonctionnalités désirées et respectant les différentes types de contraintes.

1.5 Pourquoi le codesign Matériel/logiciel ?

La nécessité d'une nouvelle méthodologie de conception vient essentiellement de la progression que connaît de nos jours le domaine de la conception des circuits intégrés à très grande échelle (plusieurs Millions de transistors sur une même puce). Selon Buchenrieder [10], le codesign progresse dans l'industrie en partie parce que, respecter les conditions du développement de systèmes de complexité croissante sous des contraintes de temps de mise sur le marché rigides nécessite une nouvelle approche.

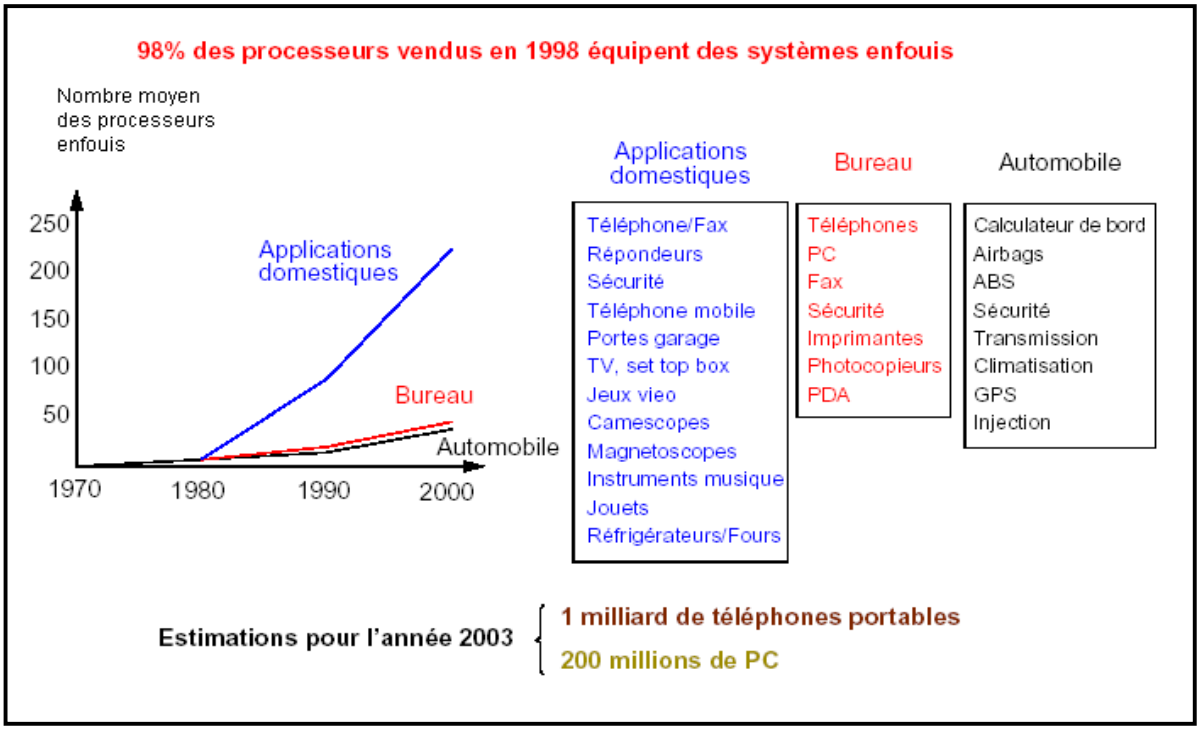

Figure 1.3: Croissance de la production des produits électroniques (systèmes embarqués) [1]

1.6 Intérêts du codesign

Parmi les intérêts du codesign, citons:

- Exploration efficace de l'espace de conception**:** Le concepteur doit explorer toutes les options possibles, outils et architectures (manuellement ou par l'utilisation d'outils automatique). Le codesign facilite l'exploration des compromis à travers les frontières de manière dynamique au fur et à mesure que le processus de conception progresse, il permet d'explorer un plus grand nombre de solutions.
- Flexibilité de la conception: La flexibilité de la conception des systèmes mixtes consiste en des modifications qui peuvent survenir durant le processus de conception, à cause des changements d'objectifs, ou d'une amélioration apportée. Pour ce faire, il est important de pouvoir effectuer des réajustements sans avoir à refaire tout le travail effectué.
- Réduction des délais de mise sur le marché: À cause des problèmes de validation du système entier et des problèmes d'interfaces entre les différentes parties au moment de la conception des systèmes [34], le temps et le coût de conception augmentent. Le fait d'adopter une méthodologie de conception unique et automatisée de toutes les parties du système mixte, permet d'accélérer le processus de conception, et de concevoir d'autres composants plus rapidement.
- Conceptions à moindre coût: La conception concurrente des deux parties (matériel/logiciel) du système à concevoir permet d'explorer efficacement un plus grand nombre de solutions en réduisant les délais de conception. Elle permet donc de fournir rapidement un bon compromis performance/coût et d'aboutir à des solutions plus intéressantes en coût et en performances

En plus de ces intérêts, le codesign facilite la réutilisation des composants matériels et des composants logiciels et il fournit un environnement intégré pour la synthèse et la validation des composants matériels et logiciels.

1.7 Domaines d'application du Codesign

Durant ces dernières années, de plus en plus d'exemples de codesign matériel/logiciel ont été publiés, et parmi les classes de systèmes concernés par le codesign, citons [14]:

- Les systèmes de communication: cette classe englobe les systèmes qui reçoivent, transforment, et émettent des flots de messages. C'est l'un des domaines les plus actifs du codesign, car il nécessite la production à faible coût de systèmes à contraintes strictes dans un environnement de grande concurrence.
- Les systèmes de traitement de signaux: cette classe concerne les systèmes de traitement et toutes formes de signaux digitaux. Le développement de systèmes utilisant des processeurs de traitement de signaux digitaux constitue une application directe des processeurs à jeu d'instruction et des processeurs dédiés.
- Les systèmes de contrôle/commande: regroupent les système qui sont en interaction avec leur environnement, composés essentiellement d'actionneurs qu'il contrôlent en fonction d'évènement perçus par des capteurs.

1.8 Architectures cible utilisées pour le codesign

Une architecture mixte logicielle/matérielle combine à la fois des composants logiciels et des composants matériels. Le but de la conception d'architecture est de décrire pour un système donné le nombre de composants, le type de chaque composant, et le type de connexion entre ces divers composants. Les composants sont les éléments de base pour réaliser une application donnée. Trois types de composants peuvent être distingués: les composants logiciels, les composants matériels et les composants de communication.

Dans la conception conjointe logicielle/matérielle, il y a une grande variété d'architectures cibles. L'utilisation d'une architecture spécifique, bien adaptée à une application déterminée, est plus efficace. Cependant, ce type d'architecture n'est pas assez flexible pour bien s'adapter dans un environnement de codesign à d'autres types d'application [69].

1.8.1 Organisation de l'architecture

L'organisation de l'architecture cible est définie par la composition ou l'assemblage d'éléments de base. Deux types d'organisation peuvent être distinguée [69]:

- L'architecture monoprocesseur: qui est composée d'un processeur et d'un ensemble de composants matériels spécifiques. Dans cette architecture, un seul processeur a la

responsabilité de l'état global du système.

- L'architecture multiprocesseur: dans ce type d'organisation, l'architecture est composée d'un réseau de processeurs autonomes communicants. Les processeurs agissent de manière indépendante. L'interaction entre les différents composants constitue la communication. Cette communication permet l'échange des informations et du contrôle.

1.8.2 Composants de l'architecture

Les systèmes complexes existants comportent plusieurs types de composants. Les composants trouvés dans ce type de système sont les processeurs, les composants matériels (ASICs, FPGAs..), composants logiciels, et composants de communication.

1.8.2.1 Les processeurs

Les travaux de [69] présentent une classification des processeurs: processeurs d'usage universel, processeurs parallèles et processeurs spécifiques.

1.8.2.1.1 Les processeurs d'usage universel

Ce sont des processeurs classiques à jeu d'instructions comme les processeurs RISC (Reduced-Instruction Set Computer) ou CISC (Complex Instruction Set Computer). Ils sont optimisés pour réaliser des cycles d'horloge courts, un nombre restreint de cycles par instruction, et une parallélisation efficace du flot d'instruction (pipelining).

1.8.2.1.2 L'architecture à processeurs parallèles

Regroupe les machines VLIW (Very Large Instruction Word), SIMD (Single Instruction stream Multilple Data stream) ou MIMD (Multilple Instruction stream Multilple Data stream) exploitant le parallélisme en utilisant de multiples unités fonctionnelles dans leur chemin de données.

1.8.2.1.3 Le processeur programmable spécifique

Appelé aussi ASIP (Application-Specific Instruction-set Processor), les applications qui y sont adaptées varient du simple algorithme à un vaste domaine comme celui du traitement du signal. Il est réutilisable avec son logiciel pour la conception de systèmes complexes. Cette réutilisation permet la diminution de temps de conception et le coût des systèmes.

1.8.2.1.4 Les ASICs (Application-Specific Integrated Circuit)

Les ASICs offrent les garanties d'offrir une basse consommation et les meilleures performances. Par contre, la flexibilité, le temps de développement et les possibilités de réutilisation sont ses points faibles. Les concepteurs utilisent plus les ASICs pour les parties d'une application bien maîtrisées qui évoluent peu, laissant le reste pour une implantation en logiciel, et ceci dans un but de flexibilité et de coût. Les circuits ASICs ont fait l'objet de beaucoup d'intérêt pendant la dernière décennie à cause du développement d'outils de synthèse de haut niveau. Ces outils ont été développés pour augmenter la productivité de conception des circuits en réduisant le temps de conception.

1.8.2.1.5 Les FPGAs (Field Programmable Gate Arrays)

Les FPGAs sont souvent utilisés en conjonction avec les composants cités plus haut. Ils fournissent des circuits logiques et numériques avec une complexité moyenne de l'ordre de quelques milliers de portes environ et peuvent être utilisés pour des applications spécifiques dans un seul circuit intégré. La popularité des FPGAs peut s'expliquer par leur facilité d'utilisation et le fait qu'ils soient programmables, tout en offrant des performances intéressantes.

1.9 Conclusion

En raison de la grande variété de réalisation des traitements, leur hétérogénéité, et les différentes contraintes de conception (temps, surface, consommation, temps de mise sur le marché, flexibilité, coût…etc.), le processus de conception des systèmes mixtes est devenu de plus en plus complexe.

Le codesign matériel/logiciel des systèmes mixtes est une méthodologie unique capable d'intégrer différents composants, en respectant un certain nombre de contraintes. Cette technique de conception prend une importance de plus en plus grande, étant donné la complexité des applications à traiter et surtout des possibilités très grandes offertes par les technologies pour la réalisation de circuits mixtes [9]. Un certain nombre de techniques et d'outils basés sur la conception de systèmes sur puce commencent à apparaître dans l'industrie, mais les années à venir devraient voir une explosion du domaine [9].

Les objectifs de ce chapitre étaient de définir le vocabulaire que nous utiliserons dans le processus de codesign, et de présenter la méthodologie de ce processus de conception, ses intérêts, ses domaines d'application et l'organisation des architectures cibles. Le chapitre suivant est consacré aux étapes de ce processus.

CHAPITRE 2
DIFFERENTES ETAPES DE PROCESSUS DU CODESIGN

2.1 Introduction

Le processus de conception des systèmes mixtes codesign suit généralement les étapes illustrées en figure 2.1. Il commence par une étape de spécification conjointe décrivant le système à concevoir [46]. Il est possible de distinguer deux types de spécifications:

- La spécification fonctionnelle qui permet de décrire le comportement attendu de l'application [37].
- La spécification des contraintes externes que le système devra supporter (consommation, contraintes temporelles,…etc).

Une étape de modélisation suite a l'étape de spécification, a pour but de modéliser les entités et les composants architecturaux du système. Elle est suivie d'une étape de partitionnement qui a pour but de répartir les entités de l'application en trois parties: une partie logicielle, une partie matérielle, et des interfaces de communication permettant la communication entre les deux parties précédentes.

Avant de passer à l'étape suivante (cosynthèse), le système partitionné doit être vérifié (fonctionnement, contraintes de temps, espace mémoire occupé, coût…).

L'étape suivante consiste à déterminer les composants à utiliser pour mettre en œuvre la partie matérielle et pour synthétiser le logiciel, en tenant compte de la communication et de la synchronisation entre les différentes composantes du système.

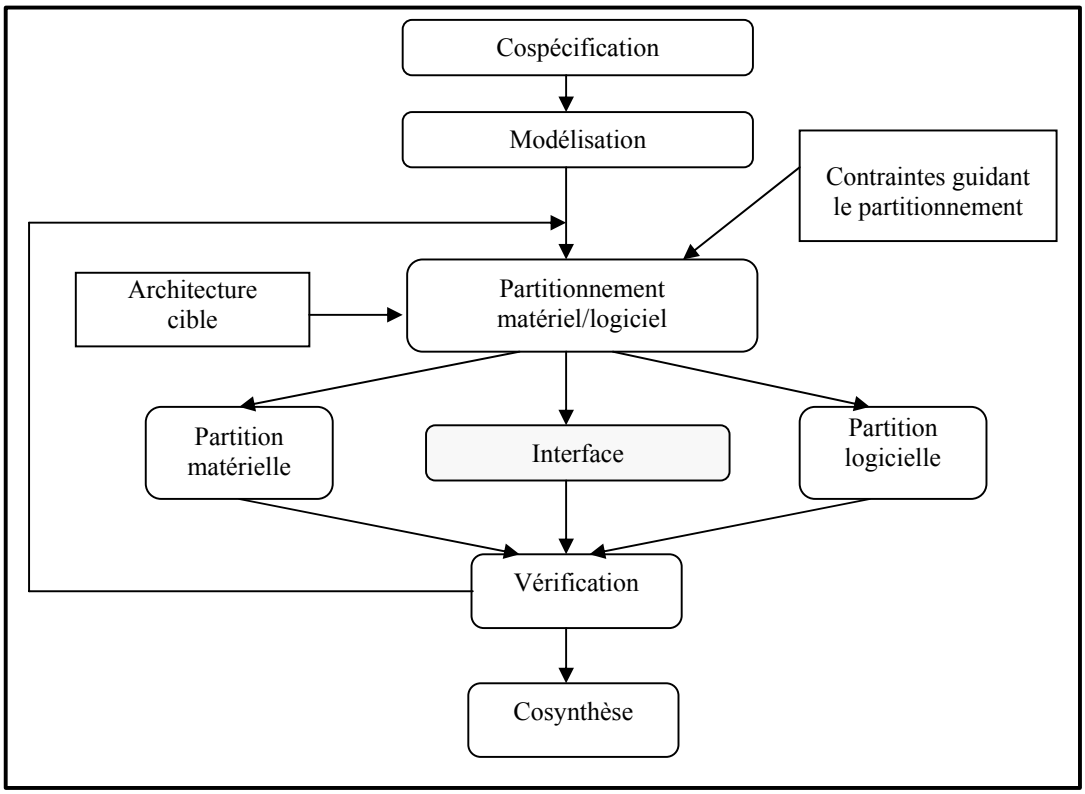

Figure 2.1: Les différentes étapes du processus de codesign [46]

2.2 Etapes de processus de codesign

Dans ce qui suit, nous revenons plus en détail sur ces différentes étapes.

2.2.1La cospécification

Cette étape consiste généralement à décrire les fonctionnalités du système à concevoir et les contraintes qu'il doit respecter [46]. Vu que les systèmes mixtes deviennent de plus en plus complexes, de plus en plus hétérogènes, cela implique la nécessité d'élever le niveau d'abstraction (niveau système). Les spécifications au niveau système sont typiquement conçues comme des ensembles de modules communicants [34]. Chaque module réalise une fonctionnalité spécifique. Les modules peuvent communiquer de manière synchrone ou asynchrone et peuvent être orientés contrôle ou donnée. De plus, ils peuvent avoir des contraintes temps réel très rigoureuses. Cette grande variété de caractéristiques pose le problème de choix du langage à

utiliser pour la spécification de tels systèmes, car aucun langage de spécification ne s'est imposé pour décrire toutes les parties du système de manière efficace.

2.2.1.1 Méthodes pour la spécification des systèmes complexes

Il n'existe pas de technique ou règle précise pour déterminer le type de langage ni le nombre de langages nécessaires à la spécification d'un tel système [34]. Selon le choix du concepteur, la spécification de la fonctionnalité d'un système complexe peut être faite à l'aide de l'une des trois méthodes:

- La spécification homogène.
- La spécification hétérogène.
- La spécification multilangage.

2.2.1.1.1 Spécification homogène

Dans ce type de spécification un seul langage de spécification est utilisé [34]. Le système est spécifié comme un ensemble de fonctions et de contraintes, indépendamment de l'implémentation et du partitionnement matériel/logiciel.

Un environnement générique de codesign basé sur un modèle homogène est schématisé dans la figure 2.2.

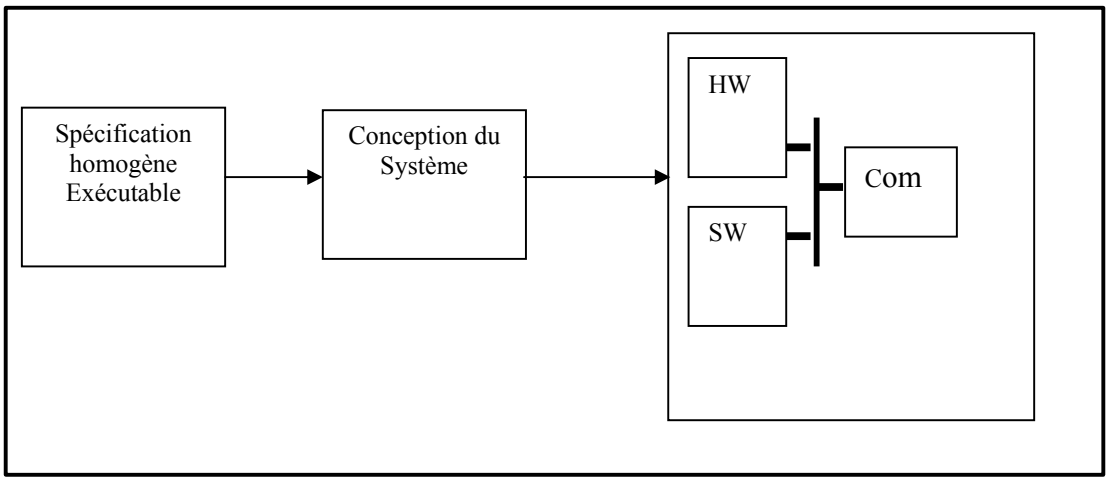

Figure 2. 2: Flot de conception pour une spécification homogène [34].

2.2.1.1.2 Spécification hétérogène

Dans les spécifications hétérogènes, des langages spécifiques sont utilisés pour les parties matérielles et logicielles. Un environnement générique de conception conjointe basé sur un modèle hétérogène est présenté en figure 2.3.

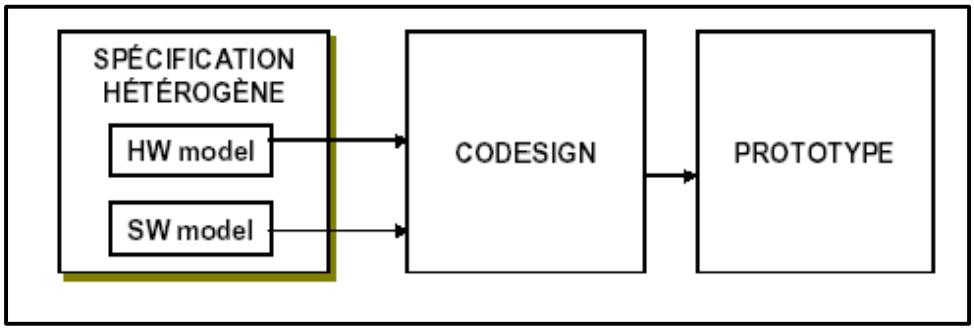

Figure 2.3: Flot de conception pour une spécification hétérogène [69].

2.2.1.1.3 Spécification multilangage

La spécification multilangage peut être vue comme une généralisation de la description hétérogène d'architectures mixtes logicielles/matérielles [34]. Il s'agit de traiter des systèmes composés des modules hétérogènes. La caractéristique de la méthode de spécification multilangage est l'utilisation de plusieurs langages pour la spécification de la fonctionnalité du système. La figure 2.4 illustre un environnement générique de codesign basé sur un modèle multilangage.

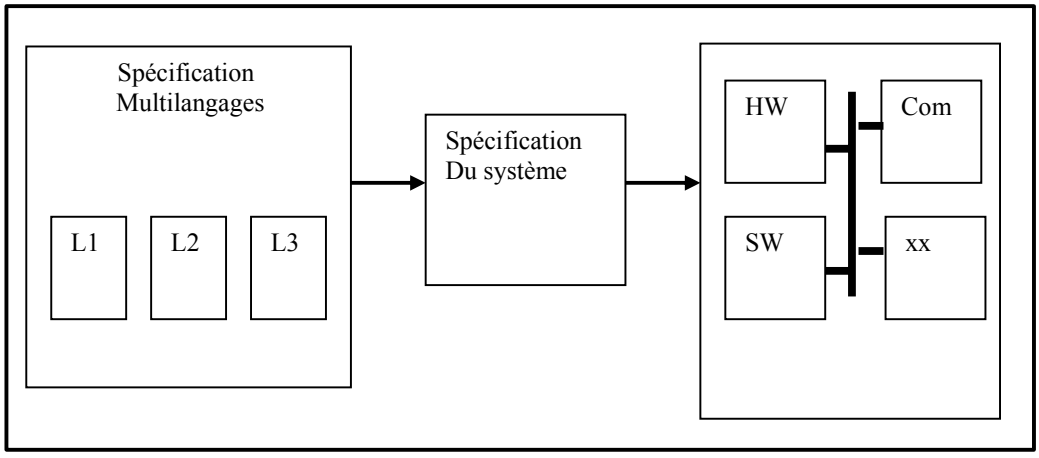

Figure 2.4: Flot de conception pour une spécification multilangage [34].

2.2.1.2 Différentes classes d'approches pour la cospécification

Il existe différentes classes d'approches utilisées pour la cospécification des systèmes mixtes. Certaines approches consistent à maintenir une nette séparation entre la spécification de la partie matérielle et la partie logicielle [46]. Chaque partie du système est spécifiée à l'aide d'un langage correspondant à sa nature (exemple C/C++/java.. pour la partie logiciel, VHDL/VERILOG pour la partie matérielle) [45, 54].

La figure 2.5 présente les différentes classes d'approches prônant l'unification de la spécification.

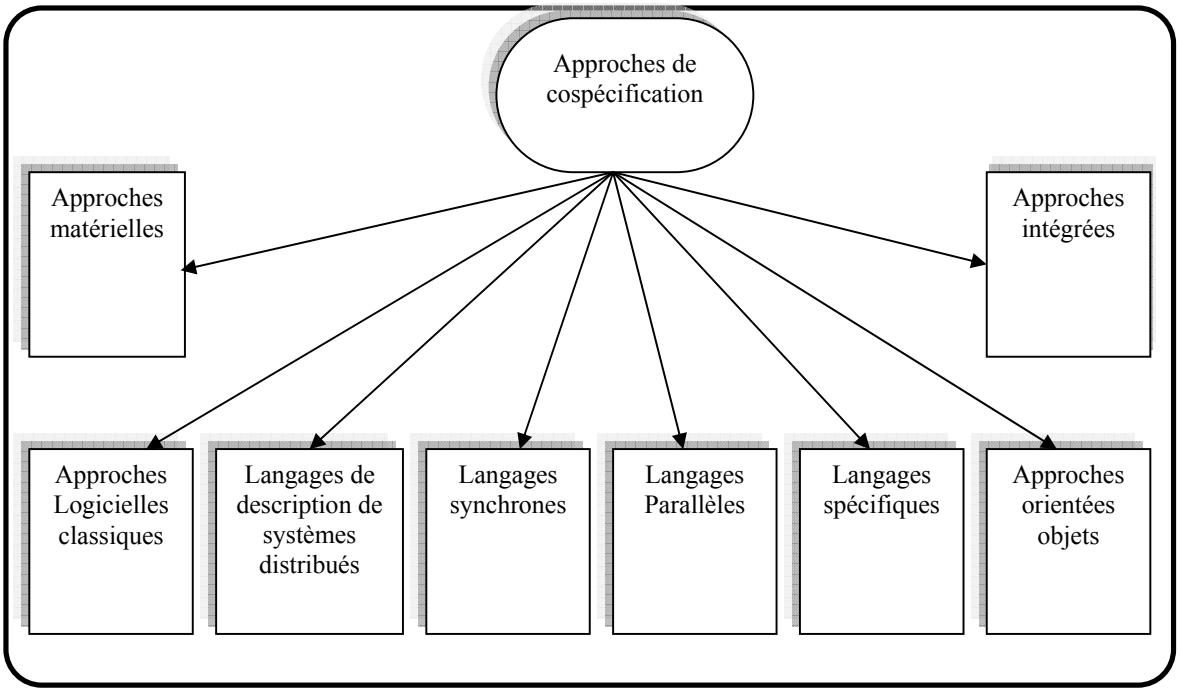

Figure 2.5: Différentes classes d'approches utilisée dans la cospécification [46].

2.2.1.2.1 Les langages de description du matériel

Plusieurs approches utilisent un langage de description de matériel pour décrire les parties matérielles d'un système en conjonction avec un langage logiciel [20]. VHDL et VERILOG figurent parmi les logiciels de description de matériel les plus utilisés soit directement comme langages de spécification des applications mixtes (matériel/logiciel), soit à l'issue de la dernière étape du codesign afin de permettre l'utilisation d'outils de synthèse commerciaux.

2.2.1.2.2 Les approches de description de logiciels classiques

Dans l'approche de description de logiciels classiques, le système est spécifié en logiciel, en utilisant un langage classique servant habituellement à décrire le logiciel. Les premiers langages utilisés pour la spécification des systèmes mixtes sont les langages d'assemblage. Etant donné le manque de souplesse et la relative pauvreté d'expression de ces langages [46], ils sont actuellement désertés au profit de langages de plus haut niveau (C, C++, Java…).

2.2.1.2.3 Autres formalismes de spécification

Il existe d'autres classes d'approches et d'autres langages de spécification des systèmes mixtes. Parmi ces approches, citons:

- Les automates à états finis étendus (EFSM: Extended Finite State Machines).
- Les machines à états finis dédiées au codesign (CFSM: Codesign Finite State Machines).
- Les modèles à diagrammes d'états (StateChart).
- Les CSPs (Communicating Sequential Process) sont largement utilisés pour spécifier les circuits asynchrones [35].
- Tangram et Balsa sont deux langages de spécification utilisant une représentation intermédiaire à base de composants qui implémentent le protocole hand-shake[11].
- Les langages de description des systèmes distribués tels que SDL (Specification and Description Language).
- Le langage synchrone SIGNAL est utilisé pour le développement d'applications embarquées et temps réel.

2.2.1.3 Les aspects que doit posséder un formalisme de cospécification

Un formalisme de cospécification doit garantir les aspects suivants.

- Le formalisme de cospécification permet l'expression des contraintes de temps.
- Il doit assurer la concurrence [21], Cette dernière est caractérisée par le grain du parallélisme. Celle-ci peut être au niveau de l'instruction (opérations parallèles) ou au niveau du processus.

- La hiérarchie qui permet de décomposer un système en sous-systèmes afin de mieux gérer sa complexité.
- La synchronisation qui permet la coordination des différents comportements et de la communication [21].

2.2.2La modélisation

Cette étape consiste à remplacer le système physique mixte qu'on veut concevoir par une représentation symbolique ou modèle [46]. Un modèle peut être défini comme étant une abstraction de ce qui existe dans le monde réel. Il peut être mathématique ou constructif.

En codesign, le choix d'un modèle interne pour un outil de conception assistée par ordinateur dépend fortement du type de système à concevoir.

2.2.2.1 Exemple de modèle

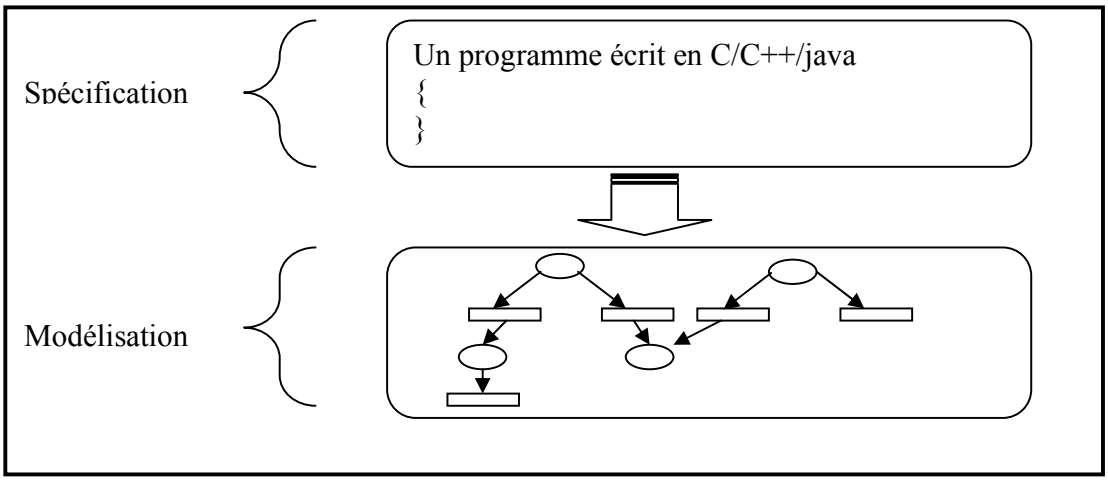

Figure 2.6: Création d'un modèle interne (ex: réseaux de Petri).

2.2.2.2 Objectifs d'un modèle

Parmi les objectifs d'un modèle dans le codesign, citons:

- Permettre au concepteur de capturer les fonctionnalités et les contraintes du système à concevoir.

- Il doit être synthétisable (c'est-à-dire permettre le passage automatique à la phase de synthèse).
- Permettre la vérification et/ou simulation.

2.2.2.3 Quelques modèles utilisés en codesign

Nous présentons ici quelques modèles utilisés pour le processus de codesign. L'objectif n'est pas de détailler chaque modèle, mais juste de citer et de donner des exemples de modèles les plus utilisés dans le codesign.

- Les automates à états finis (FSM): ils constituent un des modèles les plus utilisés pour le flot de contrôle dans les systèmes synchrones.
- Les graphes de flot de données (DFD): ils sont utilisés pour modéliser les applications de traitement de signaux digitaux.
- Les réseaux de petri: ils sont utilisés pour modéliser les systèmes mixtes [65, 66].
- Les approches orientés objets [46].
- L'approche orientée acteurs [50].
- UML (Unified modeling language).

Le choix du modèle dépend fortement du système à concevoir. Il est guidé par l'ensemble des contraintes et des caractéristiques de l'application. Un modèle choisi doit tenir compte de toutes les étapes du processus de codesign: le partitionnement, la synthèse de la communication et des interfaces, la covérification/covalidation. Les travaux de [40, 20, 69, 34] modélisent une unité de communication comme étant un objet fournissant un ensemble de primitives de communication (méthodes) réalisant un protocole spécifique.

La tendance actuelle dans de nombreuses équipes de recherche est à l'unification, pour aboutir à un modèle de calcul concurrent unique [46]. Ceci pourrait être accompli en réalisant un mélange d'un certain nombre de modèles, mais un tel modèle serait extrêmement complexe et difficile à utiliser, et les outils de synthèse et de simulation seraient difficiles à concevoir [48].

2.2.3 Le partitionnement

L'étape de partitionnement constitue le processus qui détermine les parties du système qui doivent être implantées en matériel et celles qui doivent l'être en logiciel. En général, les

partitions qui se trouvent sur le chemin critique de l'application et qui demandent un temps de réponse court sont réalisées en matériel. Les autres partitions sont réalisées en logiciel [20] afin d'offrir plus de souplesse si des changements doivent être apportés. Le partitionnement et l'ordonnancement sont deux étapes en étroite interaction. Ce sont des problèmes NP-difficiles.

2.2.3.1 La granularité du partitionnement

La granularité consiste à déterminer comment découper les entités du système à concevoir dans le but d'affecter ces fragments de code aux différents processeurs matériels et logiciels de l'architecture cible. Le niveau de granularité (degré de granularité) est le niveau de finesse adopté pour les différentes parties de l'application [46]. Dans la littérature, nous pouvons distinguer trois principaux niveaux de granularité:

- Le gros grain (niveau système).
- Le grain moyen (c'est le niveau tâche, processus, fonction...).
- Le grain fin (instruction simple, opération arithmétique).

2.2.3.2 Le choix de l'architecture cible

Le choix de l'architecture cible consiste à déterminer le nombre et le type de processeurs que le système doit contenir [37], le nombre et le type de coprocesseurs implémentant les parties matérielles ainsi que l'organisation et le choix des média de communications. Plusieurs types d'architectures cibles sont utilisés dans les différents travaux sur le codesign, par exemple: un composant unique, une carte dédiée, une carte autonome embarquée, etc.

Nous pouvons distinguer deux approches pour la classification des architectures cibles:

- La première approche consiste à fixer l'architecture avant de procéder au partitionnement de l'application;
- La seconde classe consiste à déduire l'architecture cible qui résulte de l'étape de partitionnement [46].

La figure 2.7 illustre la structure générale (schéma générique) d'une architecture cible.

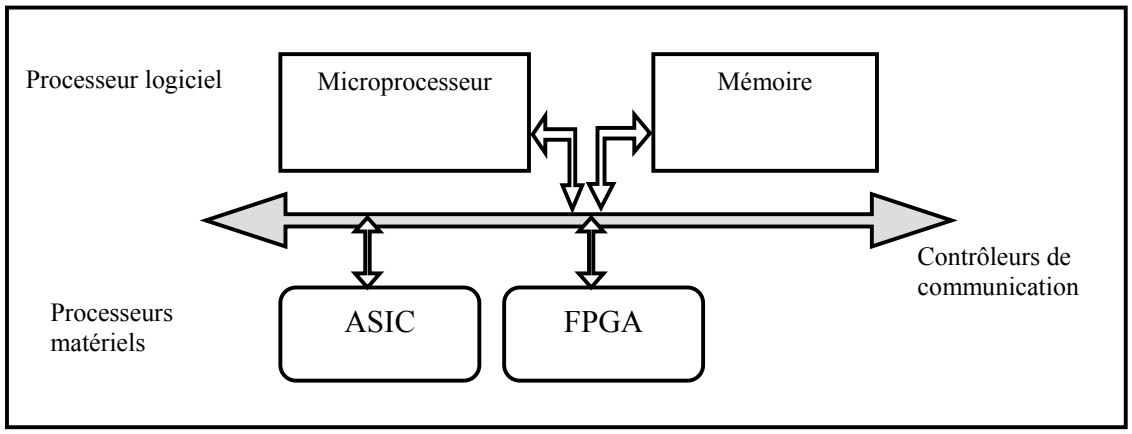

Figure 2.7: Schéma générique d'une architecture cible.

2.2.3.3 Les méthodes de partitionnement

L'étape du partitionnement est effectuée, soit d'une manière automatique, soit d'une manière interactive. Il est possible de distinguer deux méthodes de partitionnement:

2.2.3.3.1 Le Partitionnement interactif

Une technique de partitionnement interactif permet aux concepteurs de balayer manuellement ou d'une manière semi-automatique plusieurs alternatives de partitionnement du système. Cette technique prend en considération l'expérience et le bon sens des concepteurs.

2.2.3.3.2 Le Partitionnement automatique

L'approche de partitionnement interactif est intéressante quand la taille du système mixte est de petite taille. Lorsque la taille et la complexité du système augmentent, le partitionnement devient de plus en plus difficile à gérer manuellement. Dans ce dernier cas, spécialement lorsque le grain de partitionnement est fin, la majorité des chercheurs s'accorde à penser que le partitionnement automatique est la seule solution viable.

2.2.3.4 Contraintes guidant le partitionnement

Plusieurs paramètres influent sur le processus de partitionnement, citons par exemple, parmi les plus utilisés: les contraintes de temps, l'espace mémoire, la communication et la dissipation en puissance.

2.2.3.5 La fonction de coût

L'utilisation d'une fonction de coût permet de comparer les solutions possibles du partitionnement, et d'affecter un coût à chacune. L'algorithme de partitionnement consiste alors à déterminer la solution présentant le coût le minimum (maximum).

Dans les travaux d'Edwards [24], la fonction de coût inclut une combinaison du temps, de l'espace, du coût des composants, de la communication et de la consommation en puissance, où l'importance relative dépend fortement du type d'application. Henkel et Ernst [33], utilisent une fonction de coût qui prend en compte la surface et le temps d'exécution et qui permet de respecter une contrainte temps-réel et de minimiser la quantité de matériel en même temps.

2.2.3.6 Les approches de partitionnement

Dans la littérature, un certain nombre d'approches sont utilisées pour résoudre le problème de partitionnement. Il est possible de les regrouper en fonction de plusieurs facteurs: le degré de granularité, partitionnement manuel ou automatique, partitionnement binaire ou étendu.

Parmi les approches existantes, citons l'approche exacte. Son principe consiste à explorer de manière exhaustive toutes les solutions possibles. Elle a comme avantage d'offrir la certitude de trouver la meilleure solution. Cependant, elle présente un coût élevé en temps, spécialement lorsque le système est complexe.

Une autre approche est basée sur des techniques heuristiques inspirées de la recherche opérationnelle (recuit simulé, algorithmes génétiques…etc).

2.2.4 La cosynthèse d'interfaces de communication

Le partitionnement a pour but de , décomposer le système en trois parties:

- Une partie matérielle implantée sous forme de circuits (FPGA, ASIC,...).
- Une partie logicielle mise en œuvre sous forme d'un programme exécutable sur un processeur à usage général.
- Une interface de communication sert à réaliser l'échange de données entre les deux parties.

Les partitions obtenues doivent échanger des informations à travers des mécanismes de communication. Ces mécanismes peuvent être matériels (bus, arbitre,…etc.), ou logiciels (protocoles, pilotes, …etc.). La synthèse des communications dans les systèmes mixtes consiste à réaliser physiquement les canaux permettant les échanges de données entre les tâches affectées aux différents processeurs [37]. La cosynthèse d'interface de communication peut être définie comme le processus de réalisation des liens de communication entre les processus qui échangent des messages dans une architecture mixte [46].

Autrement dit, La synthèse des interfaces peut être vue comme le processus d'affinement des mécanismes de communication en partant d'un modèle abstrait jusqu'à un modèle plus détaillé de plus bas niveau d'abstraction. L'affinement prend en considération toutes les contraintes, que ce soit celles formulées par l'utilisateur ou celles émanant des règles de conception [4]. La figure 2.8 illustre le principe de cosynthèse.

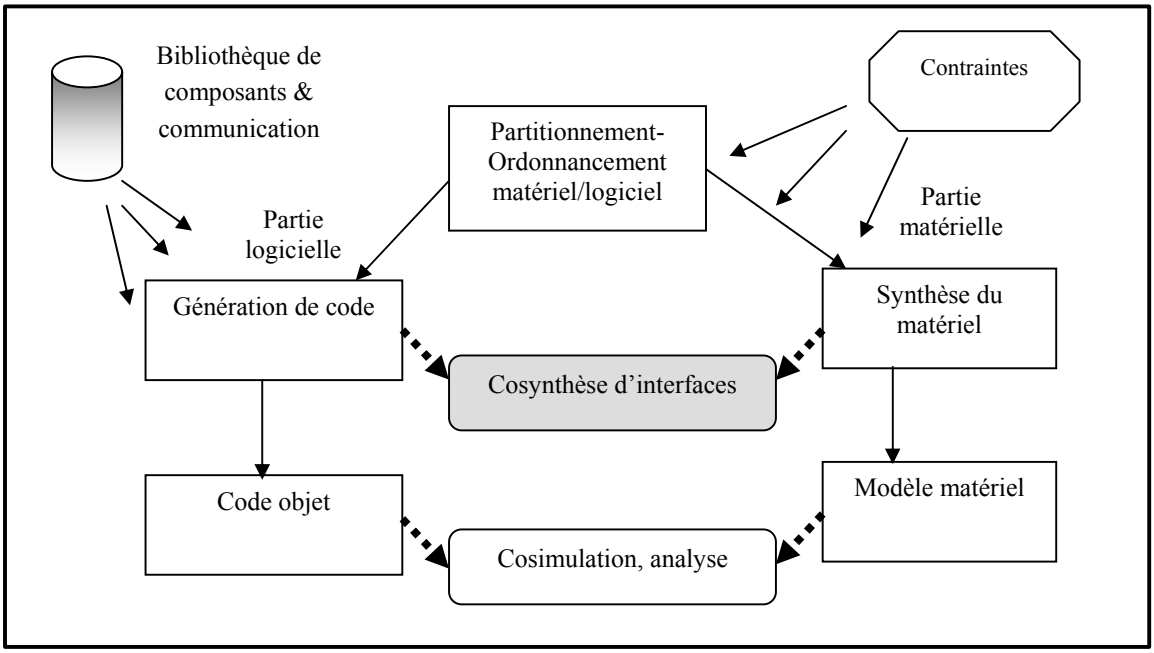

Figure 2. 8: Principe du flot de cosynthèse

2.2.5 La covalidation/covérification

C'est la vérification simultanée et parallèle des parties matérielle et logicielle. La vérification est le processus qui consiste à s'assurer qu'un système est correct et qu'il le restera,

c'est-à-dire, quelle que soit son évolution, il n'entrera pas dans des états qui empêchent son bon fonctionnement [23].

La validation d'un système conçu garantit que ce dernier réalise de façon correcte les fonctionnalités attendues tout en respectant les contraintes fixées. Dans le processus de codesign, l'étape de covérification doit garantir : [46]

- Que les fonctionnalités et les caractéristiques temporelles et de performance du système à concevoir répondent bien aux spécifications initiales de l'utilisateur.
- Que les étapes successives qu'a subi le système durant le processus de codesign n'ont pas introduit de déviations dans son fonctionnement ou dans ses caractéristiques.

Nous pouvons distinguer deux approches de covérification: covérification par cosimulation et covérification par des techniques formelles.

2.2.5.1 La cosimulation

La simulation est aujourd'hui la méthode la plus couramment utilisée [11]. Vu que la conception des systèmes mixtes devient de plus en plus complexe, la cosimulation est le moyen le plus utilisé. Elle consiste à faire fonctionner le système conçu et de comparer ses réponses avec celles du système souhaité.

La cosimulation permet la simulation concurrente de l'ensemble du système en coordonnant et en échangeant les données entre les modules [34]. C'est le fait d'élaborer, à partir de la spécification d'un système intégré, un code compilé pour sa partie logicielle et une description HDL pour sa partie matérielle [4]. Elle est un moyen d'obtenir des informations sur différentes caractéristiques du système à concevoir à partir de la spécification, telles que le temps d'exécution, la surface, le volume et le débit de communication, etc. L'approche proposée dans [51] permet la co-simulation multi-niveaux des IPs (Intellectual Property), à partir des descriptions comportementales ou d'un graphe de communication « GC » fournis par le concepteur, et de modèles « fonctionnels ».

2.2.5.2 La covalidation/covérification formelle

Les approches de vérification classiques par simulation et test ne sont pas très efficaces pour des systèmes complexes (tailles importantes). C'est très coûteux, voire impossible, de tester ou de simuler toutes les entrées possibles d'un système dont le nombre d'entrées est excessivement grand [4].

C'est pourquoi une autre approche plus fiable, consiste à étendre les méthodes formelles au domaine de codesign matériel/logiciel et utiliser ces méthodes pour la validation. La vérification formelle permet de révéler un certain nombre de propriétés d'un système, difficile à mettre en évidence par des techniques usuelles [76]. La validation par des techniques formelles consiste à utiliser de méthodes formelles afin de vérifier le comportement du système pour tous les cas possibles.

L'introduction des méthodes formelles dans le flot de conception est motivée par un besoin de correction et d'assurance plus élevée pour les concepteurs. Jusqu'ici, la simulation était le seul moyen de vérification, et elle intervenait tard dans le processus de conception [11]. Il est donc d'un grand intérêt de concevoir et de développer des méthodes formelles dédiées à la conception des systèmes.

Les techniques de la vérification formelle cherchent à démontrer si un design est correct ou non sans avoir besoin d'une simulation exhaustive. La vérification formelle d'un système se fait à la manière des preuves mathématiques des théorèmes, sans se soucier des valeurs des entrées [4]. Les travaux de Chiodo et al [16] utilisent la validation formelle sur le modèle des CFSM, pour prouver mathématiquement qu'une certaine propriété spécifiée formellement est vérifiée.

Les travaux de [67] utilisent les réseaux de Petri comme formalisme de description en codesign, afin de modéliser les différentes parties des systèmes hétérogène, pour permettre d'effectuer des preuves formelles. Les travaux de [11] intègrent les méthodes formelles dans le flot de synthèse asynchrone TAST[1]. Dans ce flot, les spécifications initiales sont écrites dans le langage CHP, elles sont interprétées en termes de réseaux de Pétri.

---

[1] TAST : Outil de synthèse asynchrone développé à TIMA, est une généralisation de la synthèse de haut niveau.

La validation formelle donne des résultats plus précis par rapport à simulation. Par contre, elle ne permet pas de déduire les caractéristiques temporelles ou les goulots d'étranglement par exemple comme le permet la cosimulation.

2.3 Conclusion

Dans ce chapitre, nous avons présentés les différentes étapes du processus de conception des systèmes mixtes (matériel/logiciel), en commençant par l'étape de cospécification jusqu'à l'étape de validation. L'objectif de ce chapitre est de décrire les différentes étapes afin de situer l'étape de cosynthèse d'interfaces de communication.

Le troisième chapitre est consacré aux problématiques de la cosynthèse d'interface de communication.

CHAPITRE 3
PROBLEMATIQUE DE COSYNTHESE D'INTERFACES DE COMMUNICATIONS

3.1 Introduction

Le résultat de la phase de partitionnement comporte trois parties: une partie matérielle, une partie logicielle et des interfaces de communication qui assurent la communication entre la partie matérielle et la parties logicielle.

L'étapes de synthèse consiste à:

- Synthétiser le logiciels (les entités affectées au logiciels doivent être compilées pour le processeur logiciel et exécuté par lui-même).
- Synthétiser des coprocesseurs matériels, ceci fait par des outils de synthèse de haut niveau [3].
- La synthèse de communication: qui consiste à réaliser physiquement des canaux permettant d'échanger des données entre les entités matérielles et logicielles [37].

La synthèse de la communication est définie par le processus de réalisation des liens dans les applications qui comporte à la fois du matériel et du logiciel. Cette phase est primordiale car les différentes parties communiquent inévitablement. Différents schémas et protocoles de communication peuvent être requis, de même que différentes topologies d'interconnexion [23]. Pour que les différentes parties du système puissent communiquer, elles doivent parler le même langage et donc utiliser un protocole qui assure le transfert et l'échange des informations entre elles. Un protocole représente l'ensemble des règles utilisées pour échanger les informations. La topologie d'interconnexion et le protocole de communication influencent grandement sur les performances d'un système et peuvent mener à des solutions infaisables si le concepteur sous-estime le débit des communications.

Dans les systèmes mixtes, il est possible de distinguer trois types de communication: communication logiciel /logiciel, communication matériel/logiciel, communication logiciel /matériel. Le premier et le deuxième type de communication posent moins de problèmes

puisqu'ils nécessitent un mécanisme simple pour la communication. Par contre, le dernier type pose plus de problèmes en raison de l'hétérogénéité des composants utilisés pour les deux parties, la nature des traitements, les différences de vitesses, etc. [46]. L'hétérogénéité des composants rend l'assemblage des interfaces très difficile, car elle nécessite la prise en compte de nombreux paramètres [30]. De plus, les interfaces doivent être adaptées aux différents besoins de l'application (en bande passante, en latence, et au modèle de communication: mémoire partagée/passage de messages).

Les concepteurs sont confrontés à la difficulté que constitue la tâche de synthèse d'interfaces et mettent l'accent sur la nécessité de disposer d'outils automatiques permettant d'aider le concepteur dans sa tâche. L'automatisation de la conception des interfaces est intéressante dans la mesure où c'est une tâche difficile qui doit être renouvelée pour chaque application [30]. L'automatisation ne doit pas limiter les possibilités de conception pour être efficace.

3.2 La synthèse du logiciel

La partie du système affectée aux processeurs logiciels doit être compilée et executée par le processeur cible. Le principal avantage des réalisations logicielles réside dans leur flexibilité. Le logiciel est synonyme de flexibilité et d'évolutivité, mais aussi de performances limitées. La flexibilité du logiciel permet de faire évoluer d'un produit si différentes versions doivent être développées. L'ajout de nouveaux modules logiciels est rendu beaucoup plus simple que pour l'ajout de modules matériels.

3.3 La synthèse du matériel

Après le partitionnement, les entités du système qui sont affectées aux processeurs matériels, necessitent de procéder une synthèse de ces composants. Cette étape est effectuée par des outils de synthèse de haut niveau admettant en entrée [37]:

- La spécification fonctionnelle de la tâche.
- Les contraintes non fonctionnelles (débit, fréquence, consommation, etc.) provenant des spécifications initiales, ou issues de l'évaluation du système;

- Les directives (indications) du concepteur.

Un ASIC est un circuit dédié aux fonctions qu'il doit réaliser, et il ne comporte donc que les composants nécessaires à la réalisation de ces fonctionnalités. Un FPGA par contre, est un composant matériel standard, permettant de programmer les fonctions qu'on désire réaliser sous forme de cellules programmables implémentant ces fonctions.

3.4 La Cosynthèse de la communication et d'interfaces

La communication entre composants est devenue un véritable goulot d'étranglement. Les performances, la consommation et les coûts de développement de ces systèmes sont fortement dépendants des choix de protocoles de communication et de leur implémentation [60]. Dans les cas des communications entre processeurs ou entre processeurs et IP, la réalisation d'implémentations mixtes logicielles/matérielles permettent de trouver de meilleurs compromis qu'avec des implémentions entièrement logicielles ou entièrement matérielles..

Les deux parties du système sont en général, indépendantes du point de vue traitement, mais elles peuvent échanger des informations à travers des mécanismes de communication [46]. Ces mécanismes peuvent être en matériel (bus, contrôleurs, etc.) ou en logiciel (pilotes, protocoles, etc). Pour communiquer, il est nécessaire de générer des interfaces qui adaptent la communication entre les entités du système affectées aux logiciels, et les entités qui sont implantées en matériel.

Un système contenant à la fois des parties matérielles et des parties logicielles aura donc besoin d'interfaces pour échanger des informations entre ces différentes parties (Figure 3.1). Ces interfaces sont des fonctions matérielles et logicielles permettant de réaliser de façon transparente les primitives de communication utilisées [37].

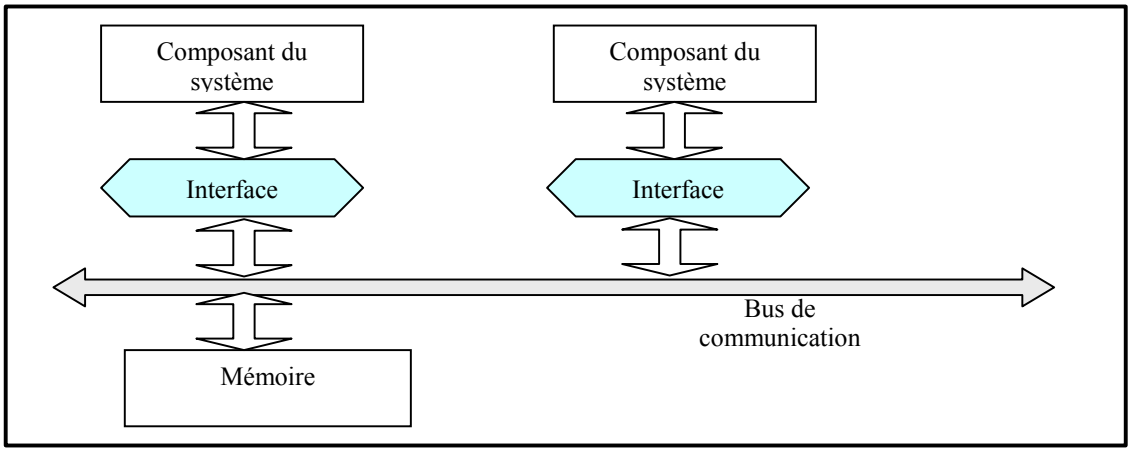

Figure 3.1: Interfaces permettant à deux composants de communiquer par un bus.

La synthèse de la communication permet, par exemple, de transformer un système composé de processus qui communiquent en utilisant des primitives de haut niveau à travers des canaux abstraits, en un ensemble de processeurs interconnectés par des bus et des signaux et partageant le contrôle de la communication [20]. Cette synthèse consiste à choisir les protocoles de communication et les éléments de calcul (processeurs, ASIC, etc.) qui seront utilisés [29]. Elle consiste aussi à réaliser physiquement les canaux permettant les échanges de données entre les tâches affectées aux deux parties du système [37].

Pour qu'un système mixte fonctionne, les divers composants logiciels et matériels doivent être interconnectés. Il est rare que les composants puissent être directement reliés entre eux [29], donc Il est nécessaire de développer des interfaces qui adaptent leur communication. La conception de ces interfaces est la clef de l'étape d'intégration des divers composants du système. C'est un travail long, fastidieux du fait de la très grande diversité des interfaces à créer.

La communication matériel/logiciel pose souvent beaucoup de problèmes et de difficultés, à cause de l'hétérogénéité des composants utilisés pour les deux parties du système, la nature des traitements, les différences de vitesses, la complexité du composants qui mettrent en œuvre la communication, etc. Un autre problème posé par l'étape de cosynthèse d'interfaces est dû au fait que certains détails de l'interface ne sont connus qu'à l'issue de l'étape de partitionnement (les goulots d'étranglement ne peuvent être identifiés que lors d'une cosimulation), et donc un feed-back est nécessaire, ce qu'il complique la cosynthèse.

L'opération de synthèse d'interfaces dépend donc de plusieurs parties qui peuvent apparaître dans un grand nombre de combinaisons, lorsque le concepteur décide d'explorer différentes alternatives de conception [46] (manipule une grande quantité d'informations). La synthèse des communications devient essentielle pour la conception des systèmes, puisque les différentes parties du système communiquent entre elles. Chaque partie du système peut utiliser différents schémas de communication, protocoles de communication et topologies d'interconnexion selon le langage de spécification utilisé et les besoins du concepteur.

3.4.1 Classification des schémas de communications dans les systèmes mixtes

Dans la littérature, différents critères peuvent être cités pour classer les schémas de communication (mode de fonctionnement, type de transfert, …etc.). Le type de transfert peut être bloquant, ou non bloquant (synchrone, asynchrone). Dans ce qui suit, nous citons les principaux schémas de communication utilisés:

3.4.1.1 La communication par mémoires partagées

Dans le modèle de communication par mémoire partagée (figure 3.2), le transfert de données se fait par l'intermédiaire d'un médium partagé (mémoire, ensemble d'adresses mémoire) dans lequel les messages peuvent être écrits et lus par les différents processus. La communication par mémoire partagée s'appuie sur la notion de variable partagée par plusieurs processus. Des mécanismes d'exclusion mutuelle ou de gestion de section critique (sémaphores) sont alors nécessaires pour assurer l'accès atomique et la cohérence des données [20]. Certains systèmes utilisent également ce mécanisme pour effectuer les entrées/sorties sur les périphériques. Le problème évident qui risque de survenir est le conflit lorsque deux ou plusieurs processus tentent d'accéder simultanément au même espace.

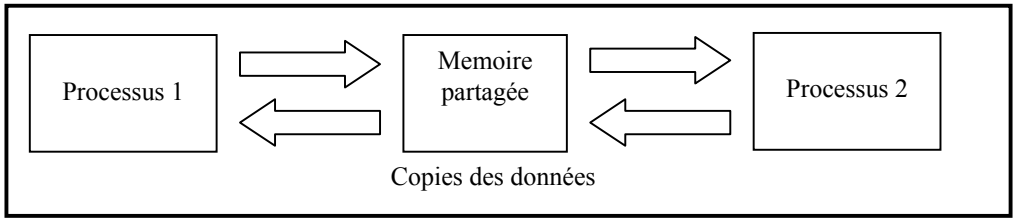

Figure 3.2: Communication par mémoire partagée.

L'avantage de ce modèle dans la programmation parallèle est que le programmeur n'a pas besoin de réfléchir à l'allocation des données dans les mémoires locales des différents processeurs [60]. Le principal inconvénient de ce modèle de programmation est qu'il impose un partage de ressources. L'implémentation d'un arbitrage des accès à une ressource partagée peut vite coûter cher en temps de communication. Cette méthode implique souvent une dégradation des performances et/ou l'augmentation de la mémoire utilisée pour le stockage des messages [60].

3.4.1.2 Communication par passage de messages

Ce modèle de communication est le plus simple. Il nécessite uniquement deux primitives de communication: l'envoi et la réception. Les données sont transmises entres processus sous forme de messages (figure 3.3). Un message est un ensemble de données associées à des informations permettant de les traiter et de les identifier. Le gros avantage du passage de message est qu'il évite les problèmes d'accès à une mémoire commune aux processus [60]. Il permet aussi de faire de la synchronisation entre processus si l'on utilise des primitives d'envoi de messages bloquantes.

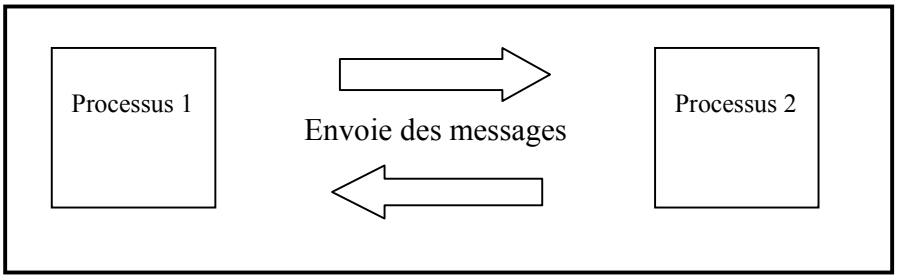

Figure 3.3: Communication par passage de message.

3.4.1.3 Les appels de procédures à distance

Les appels de procédures à distance sont un modèle hybride qui permet la modélisation des modèles de communication par passage de messages et par mémoire partagée. Dans ce modèle, la communication est effectuée par l'intermédiaire de procédures de communication, celles ci sont invoquées par les processus communicants et correspondent à des opérations qui sont effectuées par une unité de communication distante [20].

3.4.2 Niveaux d'abstraction des interfaces

La complexité de la synthèse de la communication dépend du niveau d'abstraction de la communication entre les modules [34]. Différents niveaux d'abstraction permettent de diminuer les difficultés de conception, et l'affinement progressif d'une spécification abstraite vers une réalisation finale du système [60]. Dans les travaux de Nicolescu [57], trois niveaux principaux peuvent être utilisés pour les interfaces implémentant la communication entre processeurs.

3.4.2.1 Le niveau message

Les différents modules du système communiquent via un réseau de canaux de communication dits actifs. Ces canaux permettent la synchronisation et peuvent inclure des comportements complexes. Mais les détails de la communication sont englobés par des primitives de communication de haut niveau du type (send/receive).

3.4.2.2 Le niveau architecture virtuelle

A ce niveau, la communication est assurée par des lignes abstraites englobant les protocoles d'entrées/sorties. Un modèle de ce niveau implique le choix d'un protocole de communication et la topologie des interconnexions [60]. Les langages caractérisant le mieux un système à ce niveau d'abstraction sont: Cossap [68], CSP [36].

3.4.2.3 Le niveau micro-architecture

La communication se fait par des lignes et des bus physiques. La granularité temporelle est le cycle d'horloge et les primitives de communication sont du type set/reset sur des ports. Les langages les plus utilisés pour modéliser un système à ce niveau sont SystemC, Verilog et VHDL [60].

- Dans les travaux de HESSEL [34], le type d'interface de communication de chaque sous-système dépend du niveau d'abstraction de la description du sous-système. Trois niveaux d'abstraction est utilisés comme il est présenté dans le tableau 3.1.

	Canal	**Interface**	**Primitives de communication**
Niveau application	Canaux abstraits	Portes abstraites de communication (accès)	Primitives de communication de haut niveau (put, get, etc)
Niveau registre	Bus logique	Portes logiques	Opération de lecture et d'écriture
Niveau physique	Bus physique	Ports Physiques	Opération de set et reset

Tableau 3.1: Niveaux d'abstraction des communications inter-modules [34].

• Le niveau d'abstraction application***:*** l'interface du sous-système est composée d'un ensemble des portes abstraites de communications [40], accessibles au travers de primitives de communication de haut niveau, comme par exemple *send/ receive, put/ get*. Les détails de la communication sont encapsulés par les primitives de communication et aucune hypothèse sur la réalisation de la communication n'est faite.

• Le niveau abstraction intermédiaire, (niveau registre): l'interface est composée d'un ensemble de portes logiques, appelées *"connecteurs"*. La communication est exécutée au travers des opérations de lecture et d'écriture sur des registres d'entrée et de sortie.

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

3.4.3 Types de synthèse d'interfaces (Types de conception des interfaces)

Il existe deux types de méthodes pour synthétiser les interfaces de communications.

3.4.3.1 Synthèse interactive (manuelle, semi automatique)

La conception manuelle des interfaces est complexe et longue car elle demande une double compétence: celle de la conception numérique et celle du logiciel [60]. Cette double compétence est généralement répartie sur deux équipes de développement. La difficulté de dialogue peut poser des problèmes. Pour la cohésion de l'implémentation, il faut être sûr que chaque équipe se comprend. Une erreur de compréhension peut se voir tardivement dans le flot de conception et impliquer des efforts de recherche et des modifications coûteuses dans les deux parties.

3.4.3.2 Synthèse automatique

Sans la synthèse automatique de ces interfaces, les concepteurs ne sont pas capables de simuler complètement et évaluer leurs systèmes. Fréquemment, ils sont découragés d'explorer l'espace de conception des différentes parties matériel/logiciel [17], ce qui peut donc mener à des mises en oeuvre plus coûteuses chères que nécessaire.

Vu les pressions économiques de coût et de temps de mise sur le marché (time to market), l'automatisation de la conception et de la synthèse des interfaces de communication dans les systèmes mixtes est donc devenue cruciale. La synthèse des interfaces de communication est une activité très complexe qui nécessite de disposer d'outils automatiques permettant d'aider le concepteur à sa tâche. L'automatisation peut réaliser les tâches les plus laborieuses que sont l'assemblage et l'implémentation des composants [30]. Elle permet au concepteur de considérer un plus grand nombre de solutions, et libère ce dernier de la gestion des détails de bas niveaux de l'implémentation qui sont souvent sources d'erreurs.

3.4.4 Composants et protocoles de communication

Dans ce qui suit, les composants et les protocoles de communication sont abordés.

3.4.4.1 Composants de communication

Les composants de communication sont des composants spécialisés (contrôleurs, arbitres…) matérialisant le canal de communication. Pour mettre en œuvre la communication, il est nécessaire de disposer de ces composants puisque les unités d'interfaces de chaque composant processeur ne suffisent pas pour exécuter la communication. Un composant de communication

comporte généralement trois unités: une unité interface, une unité de traitement et une unité de stockage.

3.4.4.2 Les protocoles de communication

Le but d'un protocole de communication est de décrire la sémantique (les règles) de communication à travers un port du canal de communication donné. Le protocole de communication offre une réalisation possible pour un ensemble de primitives de communication. La définition des protocoles de communication est souvent réalisée dans une bibliothèque de communication et peut contenir plusieurs réalisations différentes du même protocole [34]. Un protocole de communication contient aussi une interface sur laquelle les primitives de communication lisent et écrivent. Cette interface représente les ports et les interconnexions nécessaires aux différents sous-systèmes utilisant ce protocole [34]. Un protocole de communication est donc nécessaire pour gouverner la communication et maintenir un fonctionnement correct [11].

3.4.4.3 Les canaux de communication

Le concept de canal de communication est similaire au concept d'objet utilisé dans les langages de programmation. Un canal est un objet capable d'exécuter une communication avec un protocole défini [20]. Il est un moyen d'échange de données point à point entre modules asynchrones [11]. Un processus qui désire communiquer peut, par exemple, invoquer une des primitives de communication offertes par l'intermédiaire d'un appel de procédure à distance. La primitive est exécutée par le canal indépendamment du processus appelant. La communication est transparente pour le processus appelant [20].

3.5 Position du problème de cosynthèse d'interfaces

La figure 3.4 illustre le problème de synthèse de la communication et d'interfaces dans le flot de conception du codesign. L'étape du partitionnement décompose le système en différentes parties qui doivent être exécutées par les différents processeurs de l'architecture cible (processeur à usage général, ASIC, FPGA,…etc).

Le type de communication qui pose des problèmes est la communication matériel/logiciel. Le problème qui se pose est: comment procéder d'une manière automatique pour adapter la communication entre les deux parties du système (matériel/logiciel) ?

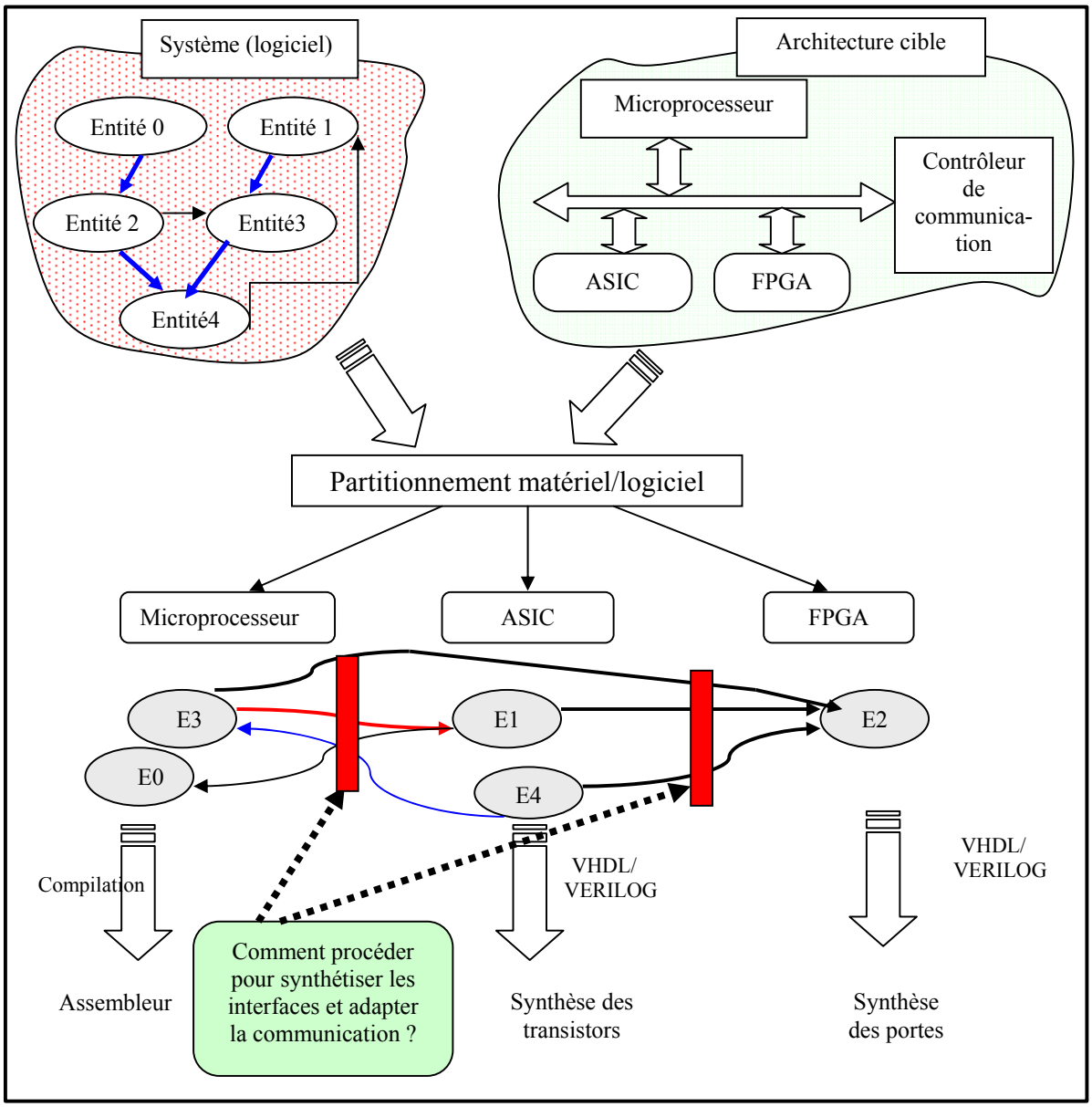

Figure 3.4: Problème de cosynthèse de la communication.

Dans l'approche objet, les entités de la spécification de l'application communiquent entre elles grâce à des méthodes publiques. Ces méthodes constituent les points de communication des

entités. Pour la partie matérielle, les composants de l'architecture cible communiquent entre eux grâce à des ports et suivant un protocole de communication qui définit l'ensemble des règles pour échanger les informations. Comme il est présenté dans la figure 3.5, les parties du système ont besoin de générer et d'adapter des interfaces de communication. Ces interfaces ont pour but d'établir une correspondance entre les appels des méthodes publiques des entités de l'application, et les ports des composants de l'architecture cible, en respectant la sémantique d'un protocole. Le problème posé est: comment procéder d'une manière automatique pour adapter les trois entités (appels des méthodes publiques, ports des composants matériels, protocole de communication) ?

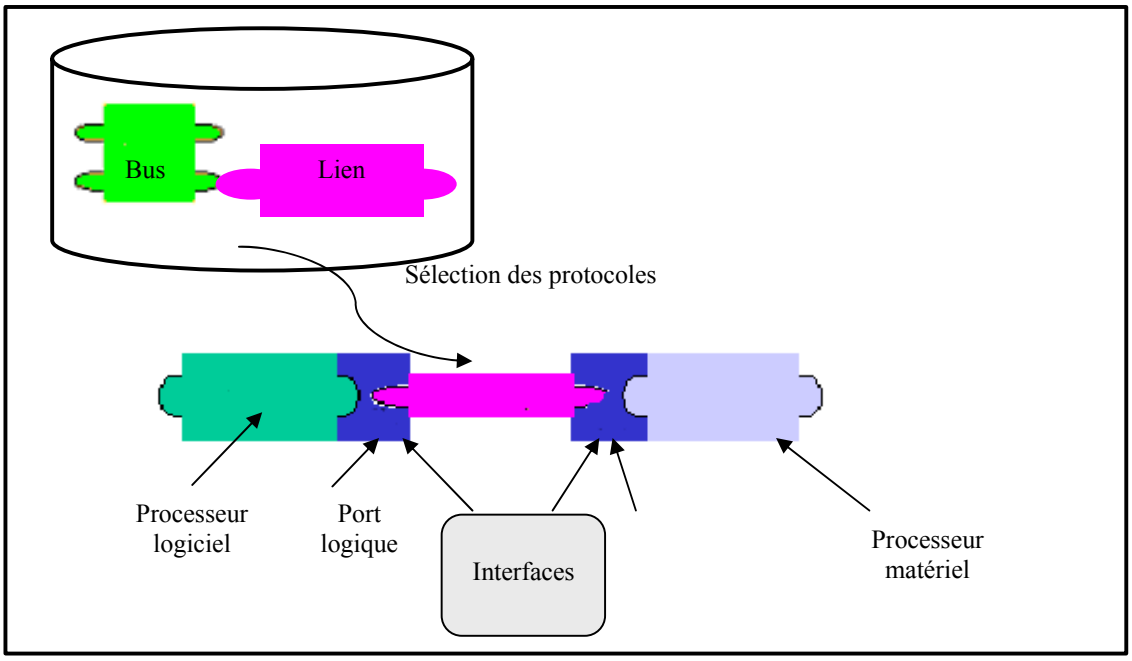

Figure 3.5: Synthèse d'interfaces de communication

Le problème de la synthèse de communication dans les travaux de [34] se résume, à produire une interface adaptée à chaque partie du système, selon leurs caractéristiques, de façon à obtenir un ensemble de modules communiquant par des bus, des signaux et éventuellement un composant de contrôle. La synthèse de la communication intermodule s'effectue en parallèle avec l'affinement des modules comme présenté en figure 3.6.

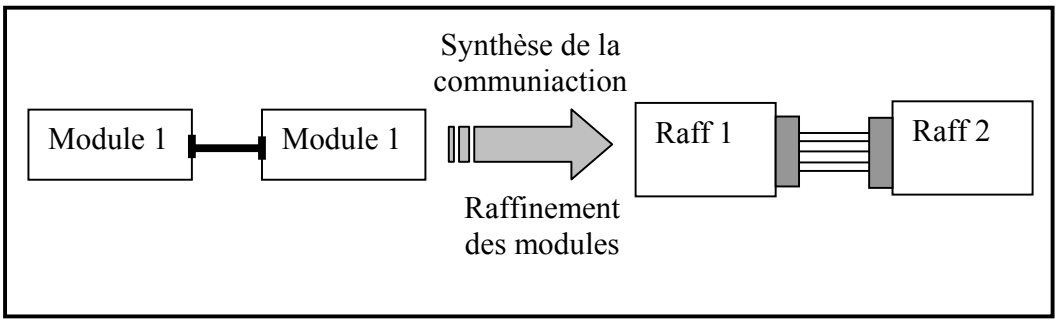

Figure 3.6: Affinement de la communication [34]

Dans [20] la synthèse de la communication est formulée comme un problème d'allocation qui choisit dans une bibliothèque un ensemble d'unités de communication qui implémentent les communications requises par les différents processus. Ce problème peut être résolu de façon automatique ou interactive.

3.6 Conclusion

Un système mixte (matériel/logiciel) a besoin d'interfaces pour adapter les communications entre ses différentes parties. La synthèse d'interfaces de communication pour les systèmes mixtes constitue un domaine de recherche très intéressant pour les chercheurs qui travaillent en codesign, elle nécessite le développement d'outils automatiques pour aider le concepteur à sa tâche de conception.

Dans ce chapitre, nous avons introduit l'étape de cosynthèse de la communication afin de citer les différents problèmes et les difficultés que rencontrent les concepteurs durant le processus de conception et la cosynthèse des interfaces de communication. Nous avons également abordé les niveaux d'abstraction des interfaces qui aident à diminuer la complexité de la conception. Enfin, nous avons positionnés le problème de cosynthèse d'interfaces.

Différentes approches existent pour la synthèse d'interfaces de communication, chacune d'elles traite le problème à sa manière et utilise ses propres techniques de synthèse.

Le prochain chapitre présente les différentes approches de cosyntèse de la communication.

CHAPITRE 4
DIFFERENTES APPROCHES POUR LA COSYNTHESE D'INTERFACES DE COMMUNICATION

4.1Introduction

La synthèse de la communication permet d'affiner les interfaces des sous-systèmes communicants. Elle permet aussi, par exemple, la définition des protocoles de communication ou de l'interface d'entrée/sortie de chaque partition du système. Les approches de synthèse qui sont présentées dans ce chapitre, se basent sur l'un des modes de communication suivants: communication fixe (communication point-à-point..), communication à travers une mémoire partagée, ou communication avec un protocole qui peut avoir différents degrés de complexité [69].

La plupart de ces approches réalisent un découpage (partitionnement) sur un graphe du flux de contrôle et de données; la communication étant alors limitée par la simplicité des protocoles considérés [69]. La synthèse d'interfaces matériel/logiciel est loin d'être une tâche triviale et constitue un intéressant domaine de recherche très pratiqué [46], pour cela plusieurs équipes travaillent sur les protocoles de communication pour des applications codesign, et de nombreux groupes de recherche se sont récemment focalisés sur le problème de la synthèse de la communication pour les systèmes temps réel embarqués.

Le but de ce chapitre est de présenter les différentes approches de cosynthèse d'interfaces de communication dans les systèmes mixtes, et de tirer quelques critères sur l'efficacité des approches proposées. Ces critères sont: le type de protocole de communication utilisé (standard, non standard ou le protocole est décrit par l'utilisateur), la synthèse est de bas niveau ou de haut niveau, le type de synthèse (automatique ou manuel), l'utilisation de système d'exploitation, l'utilisation de mécanismes d'interruptions, l'utilisation de bibliothèques de communication ou l'utilisation de l'accès direct à la mémoire (DMAC).

4.2 Différents approches de synthèse d'interfaces de communications

Différentes approches et outils de synthèse de communication dans les systèmes mixtes sont actuellement explorées. Dans ce qui suit, les principales approches et méthodologies pour ce domaine sont parcourues. L'accent est essentiellement mis sur les points suivants:

- La spécification du système et de l'architecture cible: une description brève du contexte système et du style de la description en entrée, et la description de l'architecture cible (le nombre et le type des composants architecturaux).
- L'approche de conception et le modèle de communication utilisés: une présentation des étapes suivies pour la conception mixte, en se basant sur la cosynthèse de la communication et des interfaces.

4.2.1 L'approche de Kalavade [44]

- La spécification du système et de l'architecture cible:

Dans cette approche, l'architecture cible comporte un processeur à usage général qui exécute la partie logicielle, et plusieurs coprocesseurs (processeurs matériels) qui implémentent la partie matérielle. Ces modules son connectés à un bus unique.

Le module hardware (ou coprocesseur matériel) comprend un noyau (kernel) contenant le contrôleur et le chemin de données du module. Le module communique avec le reste du système grâce à deux interfaces: une en entré et l'autre en sortie.

- L'approche de conception et le modèle de communication:

L'architecture comporte un contrôleur central câblé qui gère les communications entre le processeur et les modules matériels. Pour cela, il connaît à priori l'ordre dans lequel les modules matériels doivent être activés. Cet ordre est le résultat de l'étape de partitionnement du système (matériel/logiciel).

Les communications matériel/matériel sont réalisées en connectant simplement les circuits du module producteur aux circuits du module consommateur. Le signal « *completion* » du module producteur autorise l'écriture des circuits du module consommateur.

Les communications entre la partie matérielle et la partie logicielle sont de simples lectures ou écritures en mémoire pour le processeur. Lorsque le processeur fait une écriture en direction d'un module matériel (coprocesseur), le contrôleur valide l'écriture. Si toutes les commandes d'écritures dans le module matériel ont été effectuées, le signal *ready* commande au noyau du module de lire les données pour commencer le traitement. Lorsque le processeur fait une lecture en direction d'un module matériel, le contrôleur vérifie que le signal *completion* est actif. S'il ne l'est pas, il bloque le processeur jusqu'à ce que *completion* soit valide. Dès qu'il le devient, le contrôleur sélectionne le buffer pour que le processeur lise la donnée.

Le transfert d'une donnée entre la partie logicielle et la partie matérielle est un simple accès mémoire. Pour réaliser la communication, un contrôleur central est utilisé, et l'ordre des communications est décrit statiquement dans ce contrôleur.

Cette approche remplace le décodeur d'adresses classique par un contrôleur qui permet de s'affranchir complètement de la notion d'adresse dans le matériel. Cependant le système résultant est figé. Il est n'est pas possible de changer le logiciel, puisque l'ordre d'accès des données du matériel est codé dans le contrôleur. Un des avantages principaux de l'utilisation du logiciel est la souplesse qu'il procure.

4.2.2 L'approche Vulcan [64]

- La spécification du système et de l'architecture cible:

La description initiale est une description fonctionnelle donnée dans le langage HardwareC. L'approche de synthèse mixte utilise d'autres langages de description de matériel tels que VHDL et Verilog. Les descriptions HardwareC consistent en un ensemble de processus interagissant qui sont instanciés dans des blocs. Dans le langage de description de matériel HardwareC, un système est décrit par plusieurs processus qui communiquent entre eux. Un mécanisme de communication de mémoire partagée est utilisé pour les communications intra-processus, et le mécanisme de passage de message est utilisé pour la communication inter-processus. Chaque processus est compilé en un graphe dit « graphe de flot de données » (*dataflow graph*). Le résultat est un graphe, dont les sommets sont des opérations et les arcs représentent les dépendances de données entre les opérations.

L'architecture cible utilise un seul processeur qui est intégré avec des composants matériels (ASIC), une mémoire, et un bus. Le processeur utilise un seul niveau de mémoire et d'espace d'adressage pour les instructions et les données.

- L'approche de conception et le modèle de communication:

Le résultat du partitionnement est un ensemble de modèles matériels et logiciels. Ces modèles sont représentés par des graphes hiérarchiques acycliques. Les opérations d'entrées–sorties sont réalisées par passage de messages. Les boucles dépendant des données sont considérées comme des opérations dont le délai n'est pas borné. Deux types de contraintes temporelles sont utilisées: les contraintes de délai Minimum/Maximum et les contraintes de temps d'exécution.

La partie logicielle est découpée en *threads*, chaque *thread* est défini comme un ensemble d'opérations linéarisées (*basic bloc*). Un graphe flot de données spécifie un ordre partiel pour l'exécution des opérations. Les *threads* sont construits de façon à ce que cet ordre partiel soit respecté. Le modèle de logiciel utilisé repose sur l'exécution séquentielle de chaque *thread*. L'exécution concurrente est obtenue en entrelaçant l'exécution des *threads* sur le processeur.

L'interface matériel/logiciel consiste en des files de données et un contrôle qui maintient les identificateurs pour les processus activés dans l'ordre de l'arrivée de leurs données.

Dans cette approche, la description du système permet d'exprimer le parallélisme de l'application d'une manière explicite. La génération de l'échéancier permet de bénéficier de ce parallélisme aussi bien pour le matériel que pour le logiciel. Mais, cette génération présente des limites, c'est du fait qu'elle ne permet pas l'utilisation de systèmes d'exploitation existants.

4.2.3 Approche de sélection et d'allocation de communication à partir de protocoles abstraits [21]

- La spécification du système et de l'architecture cible:

Dans cette approche, un système est décrit par un ensemble de processus communicants à travers de canaux abstraits. Pour cela, les processus utilisent des primitives prédéfinies et des services. Lorsque les processus font appel à ces primitives, le canal réalise la communication de façon transparente du point de vue du processus. Ces procédures sont la seule partie visible du

canal. Ceci permet aux processus de communiquer par des schémas de communication de haut niveau en masquant les détails de leur implémentation.

Une unité de communication est un objet capable d'exécuter une ou plusieurs primitives de communication implémentant un protocole spécifique. Il est composé d'un ensemble de primitives de communication, un éventuel contrôleur de communication, une interface et un ensemble d'interconnexions. Les services des unités de communications peuvent partager des ressources communes (comme un bus, un arbitre de bus ou une mémoire tampon).

Tous les accès à l'interface d'une unité de communication sont faits par ces services. Ceux-ci déterminent aussi le protocole utilisé pour échanger des paramètres entre le processus et une unité de communication. L'usage de services permet de masquer les détails du protocole dans la bibliothèque, un service pouvant avoir différentes implémentations suivant l'architecture cible.

- L'approche de conception et le modèle de communication:

Le but de la synthèse de la communication dans cette approche est de transformer un système composé de processus communicant à l'aide de primitives de haut niveau, en un ensemble de processeurs interconnectés communicant à travers des signaux.

La synthèse de la communication comporte deux étapes:

a. Sélection du protocole et allocation des unités de communication: l'allocation des unités de communication admet en entrée la description du système sous forme de processus communicant par les canaux abstraits et une bibliothèque d'unités de communication. Cette étape choisit dans la bibliothèque un ensemble d'unités de communication capables d'exécuter toutes les primitives de communication requises par les processus. Elle fige ainsi le protocole utilisé par chaque primitive de communication.

b. Synthèse d'interface: en fonction des débits de données, de la mémoire requise, de la taille des bus et de l'interface, l'implémentation de chaque unité de communication est choisie dans la bibliothèque d'implémentation, cette dernière peut contenir plusieurs implémentation de la même unité de communication. La synthèse de la communication consiste à générer les interfaces nécessaires avec chaque processus utilisant les unités de communication.

L'approche proposée permet de réaliser la synthèse des communications comme un problème d'allocation de composants d'une bibliothèque. La sélection de protocole de

communication se fait automatiquement et la synthèse d'interface peut être réalisée automatiquement. L'approche proposée repose sur l'existence d'une bibliothèque contenant tout ce que l'on veut pouvoir faire dans la communication, et la réalisation pratique reste difficile à concevoir. La description du système en processus communicant par des canaux abstraits explicite le parallélisme et normalise les communications.

4.2.4 Méthodologie SpecSyn [26, 27, 28]

- La spécification du système et de l'architecture cible:

La spécification de niveau système qui est en entrée est exprimée en SpecCharts, et le logiciel généré est décrit en langage C. Les composants matériels générés sont exprimés en VHDL. L'architecture cible est une machine mono-processeur (les processeurs à mémoire cache et/ou avec instructions pipelinées ne sont pas pris en considération)

- L'approche de conception et le modèle de communication :

La conception au niveau système est divisée en trois phases distinctes:

- Prise en compte de la spécification.
- Affinement des spécifications.
- Réalisation de la conception.

La phase d'affinement joue un rôle important dans le processus de conception. Elle inclut les étapes suivantes:

- Groupement d'objets: par exemple, les variables sont groupées pour former une unité de stockage qui pourrait être implantée par la suite comme une mémoire.
- Association des groupes d'objets aux composants disponibles dans une bibliothèque.
- Affinement: il inclut à son tour les sous-tâches: synthèse d'interfaces, insertion d'arbitrage pour les canaux de communication, fusion de protocoles de communication, transposition des variables en adresses mémoire.

Dans cette approche le mécanisme de communication est basé sur l'utilisation des variables partagées. Elle génère des bus pour la communication entre deux processus pour une technique d'affinement d'interface. Les canaux de communications abstraits sont assignés aux bus

physiques. L'affinage de l'interface détermine la largeur de bus et le protocole pour le bus, cela permet d'implémenter les canaux.

4.2.5 Méthodologie de l'université Brauschweig Cosyma [32, 25].

- La spécification du système et de l'architecture cible :

Dans l'environnement de codesign Cosyma, la description en entrée est une spécification textuelle en langage C^x. Le langage C^x est une extension au langage C permettant d'inclure des contraintes de temps et le parallélisme entre les processus. L'architecture cible utilise un seul processeur avec un seul circuit intégré (ASIC), une mémoire et un bus.

- L'approche de conception et le modèle de communication:

Elle commence par la traduction de la spécification donnée en C^x vers une représentation interne appelée graphe syntaxique étendu. Ensuite, une simulation préliminaire et un profilage (*profiling*) sont réalisés. Cet environnement possède un outil de découpage logiciel/matériel qui est automatique et basé sur un algorithme de recuit simulé.

La stratégie adaptée pour le découpage matériel/logiciel consiste à partir d'une solution purement logicielle et d'extraire les parties critiques pour les réaliser en matériel en exploitant les informations obtenues par le profilage. Les parties à réaliser en logiciel sont traduites en langage C et les parties à réaliser en matériel sont traduites en langage HardwareC.

Le mécanisme de communication présenté dans COSYMA [32, 25] utilise une mémoire partagée et propose des solutions pour la communication et la synchronisation des parties logicielle et matérielle qui sont supposées s'exécuter en exclusion mutuelle, n'autorisant donc aucun parallélisme.

4.2.6 Approche proposée par [8]

- La spécification du système et de l'architecture cible:

Dans les travaux de [8], un système est décrit par un ensemble de sous système. La spécification en entrée du système est exprimée en langage SDL. La synthèse de la communication est vue comme un problème d'allocation d'unités de communication.

- L'approche de conception et le modèle de communication:

Dans cette approche, les spécifications sont décrites au niveau système.

L'approche de découpage qui a été développée se base sur une boîte à outils qui offre au concepteur le moyen de transformer, affiner, découper un système puis d'affecter chaque sous-système à une technologie particulière en logiciel (langage C) ou en matériel (VHDL). La méthode de découpage est interactive et utilise une forme intermédiaire basée sur un modèle de machines à états finis étendues communicantes via des canaux abstraits.

Une autre tâche tout aussi importante dans cette méthodologie d'affinement est la synthèse de la communication entre les différentes partitions résultantes du découpage. Cela se traduit par une étape d'allocation de protocoles de communication et une étape de synthèse d'interfaces entre les sous-systèmes communicants. La première étape consiste à sélectionner dans une bibliothèque les modèles de communication nécessaires entre les sous-systèmes. La deuxième étape consiste à adapter ou générer les interfaces des différents sous-systèmes.

L'activité de synthèse de communication comporte deux étapes:

- La synthèse de protocoles: il s'agit de partir d'un système préalablement découpé et qui est composé d'un ensemble de processeurs qui communiquent eux à travers un canal logique. La synthèse de protocoles commence par l'allocation d'une unité de communication qui réalise ce canal logique. Cette allocation s'effectue à partir d'une bibliothèque composée de canaux de communication qui suivent le principe RPC (Remote Control Protocol).
- La synthèse d'interfaces: l'objet de cette synthèse d'interfaces est d'affiner et d'adapter les interfaces des processeurs pour qu'ils puissent communiquer à travers un bus physique et éventuellement au moyen d'un contrôleur de communication.

Cette synthèse procède donc par étapes interactives qui évitent de prendre des décisions très tôt, ce qui risquerait d'écarter plusieurs alternatives de réalisation. L'idée est donc de retarder autant que possible les choix concernant la réalisation physique de la communication puisqu'une décision prise prématurément peut restreindre l'espace des solutions possibles pour la communication.

4.2.7 L'approche IPCHINOOK [18]

- La spécification du système et de l'architecture cible:

IPCHINOOK est un outil de synthèse pour les systèmes embarqués distribués qui est orienté vers la réutilisation de composants. Il reçoit en entée une description comportementale, une description de l'architecture cible et une fonction d'allocation qui définit les relations entre les deux descriptions.

La description comportementale contient plusieurs modules concurrents et communicants appelés « *modal processes* ». Elle comprend des *ports*, des *handlers* et des *modes*. Les *ports* permettent la communication entre différents processus. Le comportement d'un modal process est spécifié par ses *handlers* (manipulateurs). Un message arrivant sur un port d'entrée est un événement qui est géré par l'appel d'un *handler*. Pour le contrôle, les processus IPCHINOOK utilisent des primitives appelées ACT (*Abstract Control Types*). Ces primitives sont des règles établissant des relations entre les *modes*.

La description de l'architecture cible décrit les entrées/sorties, les processeurs, les bus de communication et la topologie du système cible. La fonction d'allocation décrit l'affectation des *modals processes* aux processeurs du système. Elle associe les éléments de la description comportementale à ceux de l'architecture. Un élément de l'architecture peut être un processeur exécutant du logiciel ou un bloc de logique programmable comme un FPGA.

- L'approche de conception et le modèle de communication:

L'étape de synthèse consiste à transformer la représentation de haut niveau vers une description plus proche de l'implémentation. Elle comprend deux étapes: la synthèse du gestionnaire de mode (*mode manager*) et la synthèse des communications et des interfaces.

- Synthèse du *mode-manager**:*** le *mode-manager* est synthétisé pour coordonner les processus. Si aucune architecture n'est spécifiée, IPCHINOOK synthétise un *mode-manager* centralisé utilisable dans une architecture monoprocesseur pour la simulation.

- Synthèse des communications et des interfaces***:*** la synthèse des communications et des interfaces implémente une infrastructure de communication spécifique à l'application permettant de gérer les messages de données de l'application et les messages de contrôle du *mode-manager*.

La synthèse des communications dans IPCHINOOK est constituée de quatre étapes: la distribution des échéances multi-saut, le calcul des attributs du protocole de bus, la génération du routeur de messages et l'instanciation des pilotes bas-niveaux. Elle est suivie de la synthèse des interfaces qui instancie les pilotes à partir de l'architecture du système.

L'approche de IPCHINOOK permet au concepteur d'appliquer les spécifications de haut-niveau sur une architecture du système comportant n'importe quelle topologie de bus. Elle permet de synthétiser les communications d'un système en utilisant une bibliothèque décrivant les protocoles à utiliser. L'utilisation de techniques de routage permet aux processeurs du système de communiquer avec tous les autres. Cette approche permet de décrire de nouveau protocoles de communication et de les insérer ou de les intégrer dans la bibliothèque. Cependant, la description du système est complexe. Le comportement des tâches y est décrit comme des gestionnaires d'évènements. Cette description semble inadaptée par exemple à la description de systèmes de type flot de données.

4.2.8 Approche de Conception des systèmes hétérogènes multilangage [34]

- La spécification du système et de l'architecture cible:

Les travaux de [34] présentent une méthodologie pour la synthèse de la communication dans le cadre des systèmes complexes hétérogènes multilangages. Cette méthodologie est une extension de la méthodologie présentée par [21], qui traite uniquement de la synthèse de la communication de systèmes homogènes.

La spécification d'un système est réalisée au niveau système à travers l'utilisation de plusieurs langages comme par exemple SDL, Matlab, C, VHDL, etc. Les interactions entre les différentes parties du système sont représentées par le biais d'un fichier de coordination. Ce fichier décrit les ports d'interface de chaque partie ainsi que les interconnexions entre les différentes parties du système.

La spécification en entrée de l'étape de synthèse de la communication est composée d'un ensemble de sous-systèmes représentant les différentes parties du système global qui communiquent via des canaux de communication abstraits. Le résultat de la synthèse est un

ensemble de processus interconnectés par des signaux qui peuvent être commandés par une unité de contrôle.

- L'approche de conception et le modèle de communication:

Cette approche permet d'affiner les interconnexions entre les sous-systèmes spécifiés dans différents langages et à différents niveaux d'abstraction. L'approche proposée s'appuie sur une bibliothèque de modèles de communication et de canaux de communication hétérogènes.

A partir de la spécification du système et du fichier de coordination, le système peut être validé une première fois afin de vérifier la fonctionnalité requise. Après cette première validation, le système peut être affiné. Cela consiste, d'abord, à découper le système en parties matérielles, parties logicielles et parties *IP* (*Intellectual Property*). L'étape suivante d'affinement est la synthèse de la communication qui détermine le protocole et les interfaces utilisées par les différentes parties pour communiquer. L'approche de synthèse de la communication est basée sur le canal de communication SOLAR. Le canal SOLAR a été étendu pour modéliser la communication hétérogène. Après les étapes d'affinement, le système peut être validé par cosimulation.

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

La méthodologie de synthèse proposée peut être exécutée automatiquement et permet au concepteur de transformer les primitives de communication de haut niveau, décrites dans différents formalismes, en une réalisation physique.

4.2.9 L'environnement MUSIC [19]

- La spécification du système et de l'architecture cible:

MUSIC est un outil pour la synthèse de systèmes mixtes logiciels/matériels à partir de spécifications système. Cette méthodologie est plus particulièrement orientée vers la conception

de systèmes distribués. La spécification du système peut être composée de plusieurs modules spécifiés dans différents langages (SDL, C, VHDL, Matlab) et à différents niveaux d'abstraction.

L'architecture cible peut être un circuit intégrant plusieurs processeurs programmables ou un réseau de processeurs. Le domaine d'application visé est celui des systèmes embarqués et plus précisément celui des télécommunications.

- L'approche de conception et le modèle de communication:

L'outil MUSIC s'appuie sur une forme intermédiaire, appelée SOLAR [40], qui est utilisée pour réaliser les différentes étapes de la synthèse. Le flot de conception de MUSIC est composé de trois étapes principales:

-L'affinement d'une spécification au niveau système dans une réalisation: les sous-systèmes sont affinés dans une réalisation logiciel/matériel.

-L'affinement de la communication: cette étape permet l'affinement de la coordination entre les différents sous-systèmes.

-La validation du système à travers la cosimulation: cette technique permet la validation du système global à différentes étapes de conception du système.

4.2.10 Approche de Génération des Interfaces de Communication pour les Systèmes multiprocesseurs Monopuces [30]

- La spécification du système et de l'architecture cible:

Dans cette approche, le système est spécifié par une architecture virtuelle. Elle représente une architecture abstraite composée de modules virtuels connectés à un réseau de communication. Chaque module virtuel est composé d'un module et de son enveloppe. Cette dernière est composée de ports virtuels qui offrent une abstraction des communications. Dans ce cas, le module représente un composant logiciel (un programme) connecté à un réseau de communication décrit au niveau RTL (niveau transfert de registres).

Les ports virtuels sont spécifiés avec le modèle de communications, qui est composé de fonctions décrivant les différentes fonctionnalités que l'interface doit fournir pour répondre aux besoins de la communication. Chaque fonction fournit des services aux autres fonctions et requiert des services de celles-ci.

- L'approche de conception et le modèle de communication:

Le flot de génération des interfaces de communication est composé de trois étapes:

▪ La première étape, qui est l'analyse fonctionnelle, transforme la spécification de l'interface en un modèle abstrait de l'interface. Ce modèle est plus approprié que la spécification pour la sélection et l'assemblage des composants. Il définit les opérations élémentaires (translation d'adresses, détection d'erreur, arbitrage, gestion de FIFO …) qui permettent de réaliser en matériel les fonctions de la spécification de l'interface.

▪ La seconde étape génère l'architecture de l'interface de communication. Cette deuxième étape assigne à chaque composant abstrait des composants génériques et détermine la topologie des interconnexions entre les composants, elle doit permettre d'explorer l'espace des solutions architecturales possibles.

▪ La troisième étape est l'implémentation, elle sélectionne et configure les composants RTL stockés dans une bibliothèque. Un outil sélectionne automatiquement les composants pour obtenir le meilleur compromis entre le coût (surface) et la performance (chemin critique). Le concepteur n'a alors pas besoin de connaître les implémentations disponibles.

Cette approche présente une méthode pour la génération automatique des interfaces de communications. Même si certaines décisions devront être laissées à l'utilisateur, l'automatisation peut réaliser les tâches les plus laborieuses que sont l'assemblage et l'implémentation des composants. Cette méthode cible la réalisation d'interfaces de communication complexes.

4.2.11 Méthodologie Ptolemy (Université de Californie/Berkeley) [12, 13, 49, 78, 63]

- La spécification du système et de l'architecture cible:

La version *Ptolemy* classique (1990-1997) traite les applications de traitement de signaux et les systèmes communicants. Ptolemy permet la simulation de systèmes en mode mixte, la spécification et la conception ainsi que la génération de code d'assemblage DSP à partir d'une description sous la forme de diagrammes blocs de l'algorithme.

Le modèle initial en entrée est formé d'un ou plusieurs programmes, s'exécutant sur des composants programmables. Au niveau système, Les contraintes de conception pour un système peuvent inclure les contraintes temps-réel, les performances requises, la vitesse, la surface, la taille du code ou la consommation en puissance.

L'architecture cible est composée d'un ensemble de processeurs qui peuvent avoir différentes configurations (monoprocesseur, multiprocesseurs, architectures parallèles,… etc.).

- L'approche de conception et le modèle de communication:

L'approche de conception est formée par une étape de découpage matériel/logiciel, une étape de synthèse de logiciel, une étape de synthèse du matériel et une étape de synthèse d'interfaces matériel/logiciel. Une dernière étape concerne la simulation du système hétérogène.

La synthèse du logiciel consiste en une génération de code pour des processeurs programmables. Le code obtenu peut être du langage C ou de l'assembleur selon le processeur cible. Du côté synthèse du matériel, la description en entrée est un code SILAGE. La synthèse d'interfaces inclut:

- L'ajout de registres, files d'attentes FIFOs, et décodeurs d'adresses dans le matériel;
-L'insertion du code pour les opérations d'entrée/sortie et les sémaphores de synchronisation dans le logiciel.

La simulation du système hétérogène a lieu une fois que les synthèses du logiciel et du matériel sont effectuées.

A partir de 1996, le groupe *Ptolemy* a commencé à travailler sur une nouvelle version du logiciel pour étendre ses fonctionnalités: c'est le projet Ptolemy II. Les principales raisons étaient de pouvoir bénéficier de l'intégration de fonctionnalités de réseau, de migration de code, de permettre de supporter nativement les threads et des possibilités offertes par Java pour les interfaces utilisateurs.

Ptolemy II est basé sur une représentation unifiée pour l'implémentation de modèles de calcul. L'architecture globale est composée d'un ensemble de bibliothèques Java qui fournissent les fonctions nécessaires pour la réalisation de tous les modèles de calcul. Cet ensemble permet la modélisation concurrente du système à un niveau d'abstraction élevé, ainsi qu'une spécification modulaire. De plus, il permet une conception géographiquement distribuée par l'utilisation de

protocoles de communication comme CORBA et Java RMI. Cet environnement réalise aussi la synthèse matérielle et logicielle.

Edward et Stephen dans [78] présentent le modèle de calcul concurrent orienté acteur (*Oriented Actor*). Les langages orientés acteurs conviennent bien aux logiciels embarqués et aux systèmes temps réels distribués. La configuration d'un modèle OA contient aussi des canaux de communication explicites véhiculant des données d'un port à l'autre. Cela impose donc qu'un canal de communication existe entre deux *acteurs* pour qu'ils puissent communiquer.

4.2.12 Autres approches de cosynthèse d'interfaces

De nombreux travaux ont été menés sur la synthèse de la communication pour le codesign ces dernières années. Plusieurs équipes travaillent sur les protocoles de communication pour des applications codesign et la synthèse de la communication. Vu la complexité croissante des systèmes mixte et la diversité des types de systèmes existant dans la littérature, il est difficile de citer toutes les approches existantes. Dans ce qui suit, nous abordons d'une manière abrégée quelques unes des approches de synthèse de la communication non encore citées plus haut.

Narayan [55] aborde le problème de la génération d'interfaces entre deux modules matériels d'une spécification. L'objectif est d'optimiser l'utilisation d'un bus en y entrelaçant les différentes communications point à point. Dans [52, 56], Narayan et Lin considèrent le problème de la synthèse d'interfaces avec conversion automatique du protocole pour une ou deux interfaces fixées.

Yen et Wolf [73] traitent la synthèse de la communication dans les architectures hétérogènes distribuées sur une topologie arbitraire. Cette approche crée un nouvel élément de calcul et un bus lorsqu'il n'est plus possible d'affecter un processus à un élément de calcul déjà existant ou une communication à un bus tout en respectant les contraintes fixées.

Dans l'environnement TOSCA [5], la spécification est effectuée en langage Occam II. Pour la synthèse de la communication, cette approche utilise des primitives de communication *Send/Rceive* pour accéder aux canaux statiques.

L'environnement CoWare [70] permet au concepteur de spécifier et simuler les interfaces de communication à plusieurs niveaux d'abstraction. Ces approches effectuent la synthèse de la communication à partir d'une spécification unifiée du système.

L'environnement de conception Polis [6, 61] permet la spécification, la synthèse et la validation de systèmes temps réels orientés contrôle. Le format unifié utilisé par Polis est basé sur la représentation d'une machine à états finis (CFSM). Le modèle de communication utilisé dans cette approche est basé sur les machines à états finis étendues pour le codesign. Dans cette approche, les communications entre des modules logiciels ou entre des modules logiciels et matériels sont réalisées au travers d'une mémoire partagée ou au travers de portes d'entrée et de sortie. Un seul bus peut être utilisé, l'utilisation de plusieurs bus n'est pas autorisée. Les interfaces entre les différents modules du système (matériel/logiciel) sont automatiquement synthétisées.

Dans les travaux d'Ortega et al [59], un modèle de communication pour les systèmes réactifs temps-réels embarqués est développé. Le modèle est basé sur un ensemble de processus qui communiquent en échangeant des messages non bloquants. Ces travaux proposent l'utilisation de mécanismes de synchronisation intégrés.

L'approche proposée par [58] traite l'assignation de la communication avec DMAC. Cette approche ne peut cependant pas manipuler des protocoles complexes.

Dans l'approche [60], la conception part d'une spécification de services de communication (protocole) dans le but d'obtenir une implémentation sous forme d'interfaces logicielles/matérielles. Ces interfaces sont des adaptateurs de communication entre les tâches logicielles exécutées par un processeur et le réseau de communication. Elles implémentent des services de communication permettant à des tâches exécutées sur deux processeurs différents de communiquer via un réseau de communication.

4.3 Comparaison

L'étude effectuée précédemment des différentes approches de cosynthèse des interfaces de communication, nous permet de déterminer quelques critères et paramètres utiles pour juger de l'efficacité de la synthèse de la communication.

Chaque approche étudiée traite le problème de cosynthèse d'interfaces à sa manière. Elle propose des solutions spécialisées et dépend fortement du type de système à concevoir, de sa spécification (haut niveau, bas niveau), et du modèle utilisé pour la communication.

Certains travaux qui ont été menés utilisent des protocoles de communication standard. [32, 25] utilisent par exemple une mémoire partagée pour la communication. Les travaux de [59] utilisent un mécanisme de passage des messages. La méthodologie SpecSyn [26, 27] utilise aussi les variables partagées comme mécanisme de communication.

D'autres approches utilisent par contre des protocoles non standard. Les travaux de [73] supposent un protocole abstrait basé sur les priorités des processeurs.

D'autres travaux utilisent des systèmes d'exploitation et des systèmes d'interruption. Un exemple de ces travaux peut être trouvé dans [59, 58].

Une autre classe d'approche utilise des primitives abstraites *Send/Receive*. Ces primitives permettent de modéliser tous les échanges des informations [36, 27, 5, 8, 64, 34]. L'utilisation des bibliothèques de communication permet de rendre l'application indépendante du système d'exploitation comme les travaux de [5, 21, 34, 60]. Peu de travaux permettent la synthèse de la communication en incluant la possibilité d'effectuer des transferts par accès direct mémoire DMAC.

Parmi les critères utiles pour juger de l'efficacité de la synthèse de la communication, citons:

- L'automatisation de la cosynthèse: elle permet de diminuer le temps de conception et de minimiser le temps de mise sur le marché. Elle permet aussi de considérer un plus grand nombre de solutions potentielles. Le caractère automatique de la cosynthèse libère le concepteur de la gestion des détails de bas niveaux qui sont souvent sources d'erreurs, lui permettant ainsi de mieux explorer les compromis entre différentes allocations, partitions, topologies ou protocoles de bus.
- Le fait que l'architecture du système sur laquelle l'application s'exécute, soit paramétrable ou non.
- L'utilisation de primitives virtuelles standard: ces primitives permettent de modéliser les transferts des informations et de normaliser la communication.
- Le fait de permettre l'ajout de nouveaux protocoles de communication.

- La prise en compte de primitives de système d'exploitation.
- L'utilisation de bibliothèques de communication permet de rendre l'application indépendante du système d'exploitation.

4.4 Conclusion

L'automatisation de la conception des sous-systèmes de communication permet au concepteur de considérer un plus grand nombre de solutions potentielles [59]. Elle lui permet ainsi de mieux explorer les compromis entre différentes partitions, topologies ou protocoles de communication.

Bien que l'opération de synthèse des interfaces de communication dans le processus de conception des systèmes mixtes matériel/logiciel intéresse les chercheurs, aucune approche ne s'est imposée et aucune n'a totalement traité tous les problèmes posés par cette opération. Par ailleurs, peu de détails de réalisation sont donnés.

Les critiques qu'il est possible d'apporter aux approches citées précédemment sont les suivantes:

- certaines d'elles possèdent des protocoles standard mais trop figés;
- d'autres utilisent des protocoles non standard et complexes;
- beaucoup ne permettent pas la description de nouveaux protocoles;
- d'autres approches utilisent des modèles de communication assez complexes.
- de nombreuses approches synthétisent les interfaces au bas niveau, ou offrent des affinements manuels des mécanismes de communication.

La complexité de la cosynthèse des interfaces de communication dépend fortement de la complexité du système à concevoir, du formalisme de la cospécification et du modèle de communication utilisé**.**

Les travaux qui continuent à paraître sur le domaine prouvent que la synthèse d'interfaces de communication constitue un domaine de recherche très pratiqué et encore très ouvert.

CHAPITRE 5
SPECIFICATION DU SYSTEME MIXTE

5.1 Introduction

L'étude bibliographique effectuée dans la première partie, souligne le fait que des travaux récents concernant la cosynthèse des interfaces de communication dans le processus de codesign sont menés. Cependant, aucune approche ne s'impose comme meilleure que les autres. Ceci est dû à la diversité des types des systèmes existants, la complexité croissante des systèmes à concevoir, la diversité des types de formalismes de cospécification ainsi que les composants architecturaux, ou encore à l'hétérogénéité des composants (entités) sur lesquels les interfaces sont générées.

Tous ces problèmes soulevés et bien d'autres fonds clairement apparaître que les concepteurs des systèmes mixtes sont confrontés à la difficulté que constitue la tâche de synthétiser ses Interfaces [46], et mettent l'accent sur la nécessité de disposer d'outils automatiques permettant d'aider le concepteur pour cette tâche.

Dans cette seconde partie, nous présentons l'approche proposée pour la synthèse des interfaces de communication dans le processus de codesign. Nous tentons de résoudre le problème lié aux manques constatés sur les approches automatiques intégrées de haut niveau pour la cosynthèse des interfaces. L'approche de cosynthèse proposée et testée est basée sur la notion de ports virtuels et de primitives virtuelles de communication (*Send*, *Receive)*. Elle consiste à procéder à des affinements successifs sur le code de cospécification (après une modification dans le code initial et le modèle des composants de communication). La quantité importante d'informations hétérogènes manipulées tout au long du processus de cosynthèse doit être prise en charge.

Notre contribution dans ce travail consiste à:

- Développer et réaliser une approche permettant d'analyser le code de la cospécification orientée objet (en Java), afin d'extraire toutes les informations qui caractérisent la communication entre objet. Ces informations sont représentées à la fin de l'opération de l'analyse par deux graphes: un graphe hiérarchique (des classes) qui représente la hiérarchie des classes de la cospécification, un graphe d'instance (fonctionnel): qui représente les liens de communication entre objets (présenté en chapitre 6).
- Développer et réaliser une technique permettant de décrire les composants architecturaux et les protocoles de communication (une présentation détaillée est illustrée dans chapitre 7).
- Développer une nouvelle approche de cosynthèse des interfaces de communication pour le processus de codesign, qui est basée sur les informations fournies par les deux étapes précédentes.

Cette approche proposée est mise. Cette étape est présentée en chapitres 8 et 9.

5.2 Présentation générale de l'approche proposée

Comme il est illustré dans la figure 5.1, notre approche commence dès la première étape du processus de codesign, par la cospécification de l'application et la description des composants de l'architecture cible.

L'étape de cospécification de l'application constitue l'étape clé du processus de conception, car elle a une influence sur l'efficacité des étapes suivantes. Nous avons fait le choix d'un formalisme de cospécification orienté objet (proposé dans [46]). Notre cospécification est effectuée à l'aide du langage orienté objet Java qui permet de décrire toutes les entités de l'application à concevoir.

Dans un premier temps, une étape d'analyse du code de la cospécification permet d'extraire toutes les informations pertinentes concernant les classes, les méthodes, les appels aux méthodes et la communication entre objets. Ces informations sont représentées par le graphe fonctionnel (d'instance), et le graphe hiérarchique (graphe des classes).

L'environnement de description de l'architecture cible permet la création des composants architecturaux (composants processeurs, composants de communication…). Il offre au concepteur de créer de nouveaux composants, ou de choisir dans les bibliothèques, des

processeurs et des composants de communication pour composer son architecture d'une manière graphique. Il peut également sélectionner le(s) protocole(s) de communication décrivant le déroulement des opérations de communication ou d'en créer de nouveaux.

L'étape de cosynthèse des interfaces de communication consiste à réaliser des liens de communication entre les objets (objets matériels, objets logiciels), qui échangent des informations dans un système mixte. La figure 5.1 illustre le schéma global de notre nouvelle approche pour la cosynthèse d'interface de communication, cette approche se déroule en quatre étapes principales:

- La première étape est effectuée durant la description de l'architecture cible et des composants de communication, elle consiste à superposer les ports et des protocoles virtuels sur des composants architecturaux.

- La deuxième étape est réalisée pendant l'analyse du code de la cospécification Java. cette tâche consiste à créer des ports virtuels pour les paramètres d'appels des méthodes des objets, puis à insérer des appels aux primitives virtuelles (*Send*/*Receive*) dans le code de la cospécification.

- La troisième étape se déroule à l'issue de l'étape de partitionnement. Elle procède à des affinements successifs sur les appels des primitives virtuelles insérées dans le code de cospécification de l'application.

- Enfin, la dernière tâche consiste à éliminer les parties inutiles des mécanismes d'interfaces.

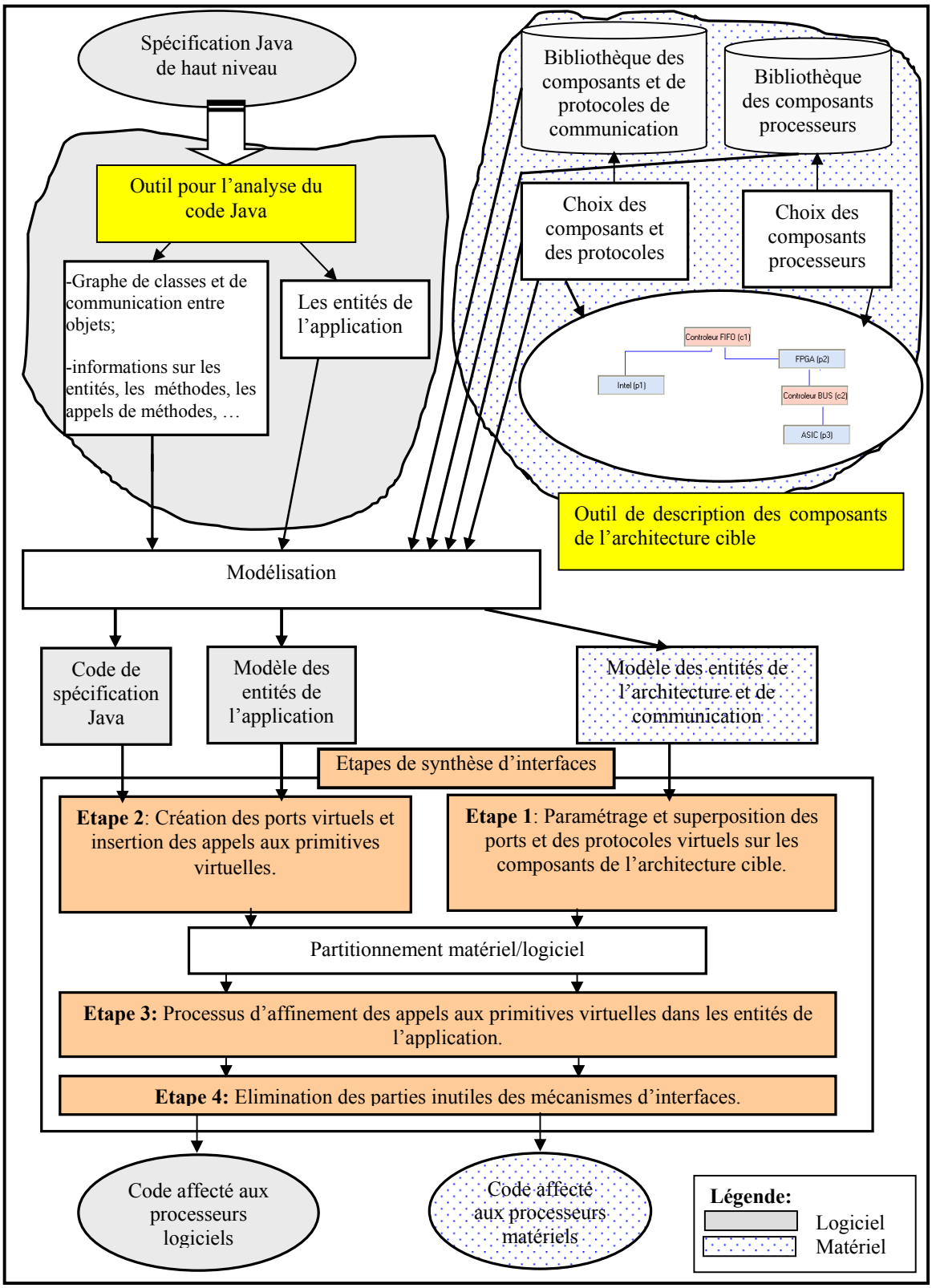

Figure 5.1: Présentation générale de notre approche et ses étapes de cosynthèse d'interface

5.3 Formalisme de cospécification orienté objets

Dans cette approche, l'étape de cospécification est basée sur l'approche orientée objet. La partie application est décrite en langage Java, et la partie de l'architecture cible est décrite d'une manière graphique en utilisant les composants et les protocoles stockés dans des bibliothèques.

L'approche orientée objets est certainement la plus utilisée pour la spécification de systèmes complexes du fait qu'elle simplifie le développement de logiciels plus gros et plus complexes et du fait qu'elle soit sujette à moins de défaillances [62]. Cette approche consiste à créer une représentation abstraite, sous forme d'objets, d'entités ayant une existence matérielle (processeur, mémoire,…) ou bien virtuelle (temps,...etc). Un objet est caractérisé par son nom (identité) qui permet de le distinguer des autres objets indépendamment de son état. Il est caractérisé aussi par ses attributs qui sont les variables stockant des informations d'état de l'objet, et les méthodes membres qui définissent le comportement de l'objet.

Le mécanisme de base de l'approche orientée objet c'est l'encapsulation des données. Cette dernière permet une séparation nette entre les détails d'implémentation de l'objet et l'interface publique d'utilisation, facilitant l'ajout, le retrait et la réutilisation d'un objet dans l'application [46]. Elle limite également les effets que les changements sur un objet particulier peuvent avoir sur le reste du système, et la modification locale d'un objet est facilitée si cette modification ne met en cause ni l'interface publique de l'objet, ni ses relations avec les autres objets.

Les mécanismes de composition et d'héritage permettent de faciliter le développement du logiciel; la notion de l'héritage permettant de partager hiérarchiquement des propriétés (attributs et méthode) des objets. Elle permet aussi la création et la réutilisation de nouvelles entités. Un autre mécanisme aussi important que les deux précédents est le polymorphisme. Ce dernier signifie que les différentes méthodes d'une classe peuvent avoir le même nom. Lorsqu'une méthode est invoquée sur un objet, celui-ci connaît sa classe et par conséquent, il est capable d'invoquer automatiquement la méthode correspondante. Une surdéfinition (ou surcharge) permet d'utiliser plusieurs méthodes qui portent le même nom au sein d'une même classe, avec une signature différente. Ces mécanismes offrent ainsi une plus grande souplesse de développement et de conception d'un système mixte.

5.4 Motivation du choix d'un formalisme orienté objets

Le choix du formalisme de cospécification orienté objet est motivé par les avantages qu'il offre:

- Un model unique pour le matériel et le logiciel qui facilite la spécification du système global.
- La réutilisabilité des composants dans le modèle objet reste valable pour les systèmes mixtes en codesign.
- Extensibilité incrémentale du système permise aussi grâce à la notion d'héritage.
- L'utilisation des mécanismes d'abstraction (hiérarchisation, encapsulation, héritage et agrégation, …) qui sont adapté à la description des systèmes complexes.
- La granularité de l'environnement est au niveau objet, ce qui permet de réduire la complexité de l'analyse.
- Le concept d'objets est adapté à la description du matériel, car les composants matériels sont naturellement décrits par des états et des opérations, et les détails de mise en œuvre de bas niveau sont rarement connus par l'utilisateur. Prenons l'exemple d'un état d'une zone mémoire ou d'un registre: l'accès à l'information stockée dans le registre se fait à travers les opérations qui le manipulent et qui constituent une interface d'accès à ce composant.

5.5 Langage Java pour la cospécification

Nous avons présenté dans la première partie différents formalismes de cospécification pour le codesign. Certaines équipes préfèrent le langage C++, Java, alors que d'autres équipes utilisent un formalisme de cospécification plus ou moins dédiée (ex: VHDL pour la partie matérielle).

Pour notre approche, et en particulier l'étape de cospécification de l'application, nous avons porté notre choix sur un langage unique pour la partie matérielle et logicielle: le langage Java. La totalité du système peut ainsi être décrite d'une manière unifiée: aucun biais n'est introduit vers une tendance de mise en œuvre (matériel ou logiciel). En effet, l'utilisation de langages différents pour la cospécification de la partie matérielle et la partie logicielle influe négativement sur l'étape du partitionnement (méthode de partitionnement), conduisant souvent à éliminer certaines solutions et donc à réduire l'espace des solutions.

Java est un langage objet proche du langage C++. Il a été mis au point en 1991 par la firme Sun Microsystems. Etant donné que le langage C++ comportait trop de difficultés, James Gosling, un des acteurs du projet (considéré désormais comme le père de Java) décida de créer un langage orienté objet reprenant les caractéristiques principales du C++, en éliminant ses points difficiles, et en le rendant moins encombrant.

Dans la littérature, plusieurs travaux utilisent le langage C/C++ pour la cospécification de niveau système [39, 71, 15, 72]. Les travaux présentés dans [53] utilisent les langages C et Java. Les travaux de [75] utilisent un sous-ensemble de Java pour la description de systèmes embarqués. Les travaux présentés dans [41] illustrent une approche pour le codesign matériel/logiciel. Ils utilisent le langage Java comme langage de spécification du système. Un compilateur est décrit qui permet d'extraire toutes les informations de la représentation orienté objet (graphe de représentation intermédiaire).

5.6 Motivation du choix du langage Java

Le choix du langage Java comme langage de cospécification est motivé par les nombreux arguments en sa faveur, dont les principaux sont:

- Il est puissamment supporté par une grande entreprise.
- Il garantit une cospécification de haut niveau, ce qui représente, outre les avantages classiques de l'approche orientée objet adoptée, la possibilité de structurer le système de manière hiérarchique, modulaire, afin de décrire de manière simple et naturelle ses différentes composantes [46].
- Il a été conçu pour être de petite taille, simple, portable entre les plates formes et les systèmes d'exploitation, adapté à l'architecture des réseaux et les systèmes embarqués.
- Il permet de décrire le parallélisme à travers l'exécution de processus multiples (processus légers ou threads), c'est exactement le modèle Multi-threading.
- La synchronisation entre threads est assurée à travers les sections critiques (sémaphores…).
- Il a été étendu par des bibliothèques afin de prendre en compte les contraintes de temps: JavaTime [74] et Real-Time Java [77].

- Langage orienté objet, Modulaire, extensible.
- Existence d'outils de déboguage et de mise en œuvre fiable.

5.7 Modèle orienté objet pour les entités de l'application et les composants de l'architecture cible

Dans ce travail, nous avons utilisé le modèle orienté objet pour les entités de l'application et les composants architecturaux proposé dans les travaux de [46]. La figure 5.2 présente un modèle générique. Ce modèle permet de modéliser à la fois les objets de la cospécification de l'application et les composants architecturaux (les processeurs matériels et logiciels et les composants de communication).

Un objet de la cospécification de l'application comporte trois parties: une partie données représentée par les données membres de l'objet, une partie traitement qui est représentée par les corps des méthodes membres de l'objet et une partie interface qui est représentée par les entêtes des méthodes qui permettent l'accès à l'objet.

De même, la modélisation d'un composant matériel de l'architecture cible est inspirée de la modélisation d'objets. Un composant matériel est composé généralement de trois parties: une partie matérialisant les traitements (unité de traitement) et les calculs effectués par le composant à modéliser, la seconde partie est réservée au stockage (mémoire de stockage de composant), et la dernière partie constitue l'interface de communication. Cette interface permet la connexion du composant modélisé avec les autres composants à travers des ports. Un dictionnaire peut être rajouté au trois parties pour contenir les informations de chaque composant.

Prenons l'exemple d'un composant mémoire ou un composant de communication. Le concepteur ne peut agir sur l'état du composant mémoire (les données mémorisées dans la mémoire) qu'à travers les opérations qui le manipulent et qui constituent une interface d'accès à ce composant. Il est possible de faire une lecture, ou une écriture sans avoir à connaître le détail de bas niveau de construction de ce composant. Un composant mémoire dispose d'une unité de traitement qui met en œuvre ces opérations (lire/ écrire).

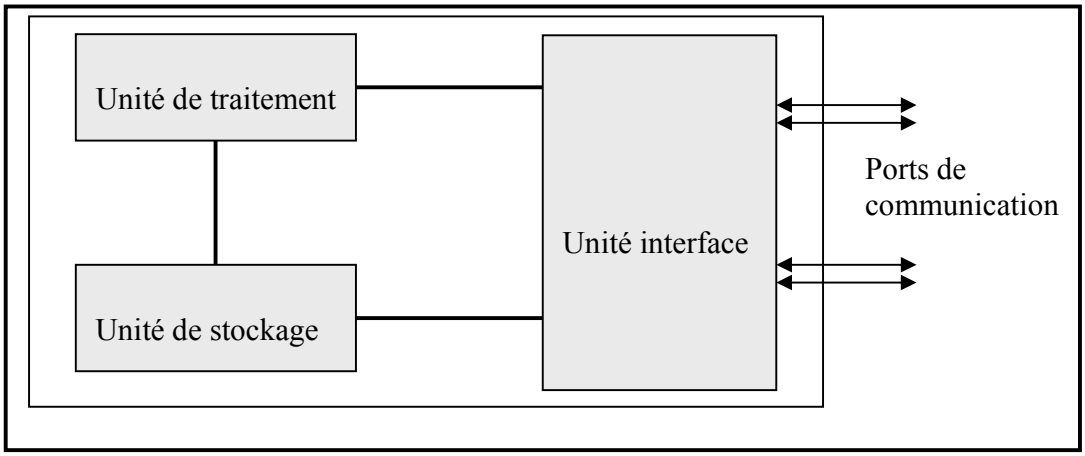

Figure 5.2: Modèle générique représente les différentes unités d'un composant [46].

5.8 Conclusion

La cospécification des systèmes mixtes constitue une étape importante dans le processus de conception codesign, car elle permet de décrire tous les paramètres logiciels et matériels du système à concevoir. Nous avons fait le choix d'un formalisme de cospécification orienté objet. Ce formalisme permet une décomposition de haut niveau du système (en objets), facilitant la modélisation et la représentation des différentes informations nécessaires aux étapes ultérieures. De plus, le formalisme de cospécification orienté objet offre plusieurs mécanismes, tel que l'héritage, la composition et le polymorphisme. Ces mécanismes permettent d'offrir une plus grande souplesse de développement qui favorise l'amélioration et la maintenance du système, réduisant ainsi le temps de conception et les risques d'erreurs.

Le modèle orienté objet permet l'encapsulation des données et des traitements qui s'effectuent dessus. Cette encapsulation permet de cacher tous les détails d'implémentation aux autres objets. L'accès aux objets se fait à travers des interfaces bien définies.

Nous avons présenté dans ce chapitre une vue générale de notre approche pour la cosynthèse des interfaces de communication, en choisissant un formalisme de cospécification orienté objet et le langage Java comme langage de spécification orienté objet. Nous utilisons un modèle pour la modélisation des composants architecturaux et des protocoles de communication, ce modèle est proposé dans les travaux de [46].

Le prochain chapitre donne plus de détails sur notre première contribution qui est l'analyse du code de la cospécification Java.

CHAPITRE 6
ANALYSE DU CODE DE LA COSPECIFICATION

6.1 Introduction

L'étape de cospécification est l'étape clé du processus de conception. C'est une représentation abstraite du système qu'on veut concevoir. Dans l'approche proposée, le comportement d'un système mixte est décrit en Java.

L'étape de modélisation, quant à elle, consiste à établir une correspondance entre le modèle générique utilisé et les classes de l'application (les classes de la cospécification Java). Il est donc nécessaire d'effectuer une analyse de la cospécification Java pour extraire toutes les informations concernant les objets, les classes, et les liens de communication entre objets. Le résultat de l'analyse est l'obtention de deux graphes: le premier permet de représenter la structure hiérarchique de l'application (graphe des classes); le second représente la structure fonctionnelle de l'application (graphe d'objets ou d'instances).

6.2 Principe de l'analyse de la cospécification

Le code de la cospécification Java à analyser est supposé correct d'un point de vue syntaxique sémantique, et lexical. C'est l'un des avantages de l'utilisation du langage Java comme formalisme de cospécification dans les systèmes mixtes. Le concepteur peut valider son application en utilisant un compilateur classique avant de la passer au processus de codesign.

6.3 Différentes étapes de l'analyse

L'analyse de la cospécification Java se déroule en deux étapes, comme illustre la figure 6.1. Le module d'analyse reçoit en entré la spécification du système sous forme d'un programme Java. La lecture du code se fait à travers un éditeur texte qui permet aux concepteurs du système d'entrer le code de la cospécification sous forme textuelle. La figure 6.2 illustre la partie de l'outil développé qui permet de lire le code source (Java).

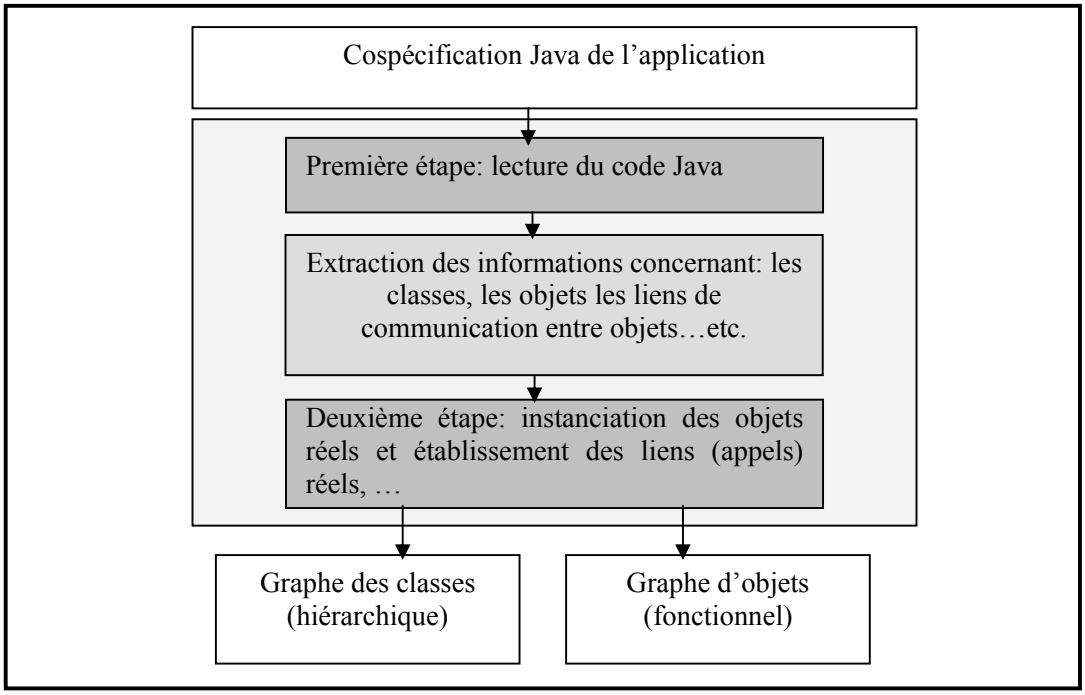

Figure 6.1: Etapes de l'analyse de la cospécification Java.

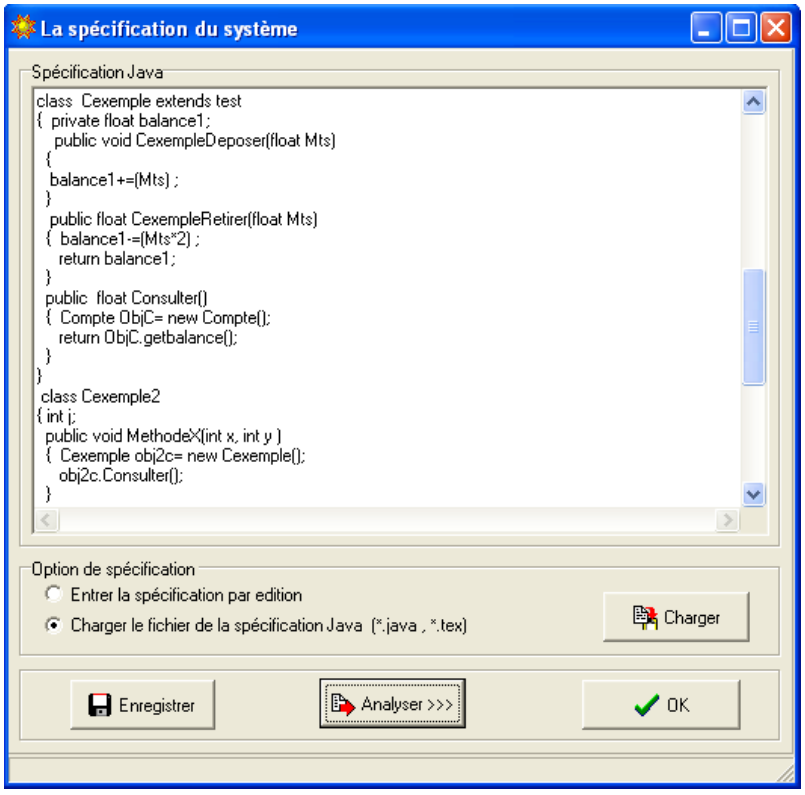

Figure 6.2: Outil permettant la lecture du code de la cospécification Java

6.3.1 Première étape de l'analyse

Cette étape permet de lire toutes les unités du code source afin d'extraire un certain nombre d'informations sur les classes, les objets, les appels aux méthodes des objets (les liens de communication entre objets). Elle a pour buts:

- D'identifier toutes les classes de l'application afin d'extraire leur nom, leurs classes ascendantes, leurs classes internes, leurs méthodes membres, leurs objets internes, etc.
- D'identifier tous les objets déclarés dans le programme: objets membres d'une classe ou d'une méthode, objets globaux, etc.
- D'identifier toutes les méthodes membres de chaque classe et d'en extraire le nom, la liste des objets membres, la liste des paramètres objets en entrée et en sortie, etc.
- Pour chaque méthode, détecter tous les appels aux méthodes, les paramètres d'appels (en entrée et en sortie).
- Pour chaque méthode de chaque classe, établir tous les liens de communication qui existent entre les objets (méthode appelante, méthode appelée, paramètres d'appels).

6.3.2 Deuxième étape de l'analyse

Après extraction des différentes informations qui caractérisent le code de cospécification Java, l'étape suivante consiste à instancier réellement les objets ainsi que tous leurs descendants et de les insérer dans la structure de données globale qui représente le graphe d'objets du système. Ensuite, établir les liens réels de communication entre tous les objets de l'application. La correspondance est établie entre les paramètres réels (de l'appel) et les références d'objets. Cette étape sert donc à compléter les informations contenues dans le graphe de communication.

6.4 Le résultat de l'analyse de la cospécification

L'analyse de la cospécification du code Java permet de construire les deux graphes: graphe des classes (hiérarchique) et graphe d'objets (fonctionnelle).

6.4.1 Graphe des classes (hiérarchique)

Le graphe des classes représente la hiérarchie des classes d'objets. Il est utilisé pour décomposer hiérarchiquement les différentes classes d'objets de l'application. Il est important pour l'étape de partitionnement (partitionnement multi-niveaux).

Ce type de graphe permet d'identifier les classes, les objets, les méthodes membres et de déterminer les relations entre classes. La figure 6.3 illustre un exemple de graphe de classes.

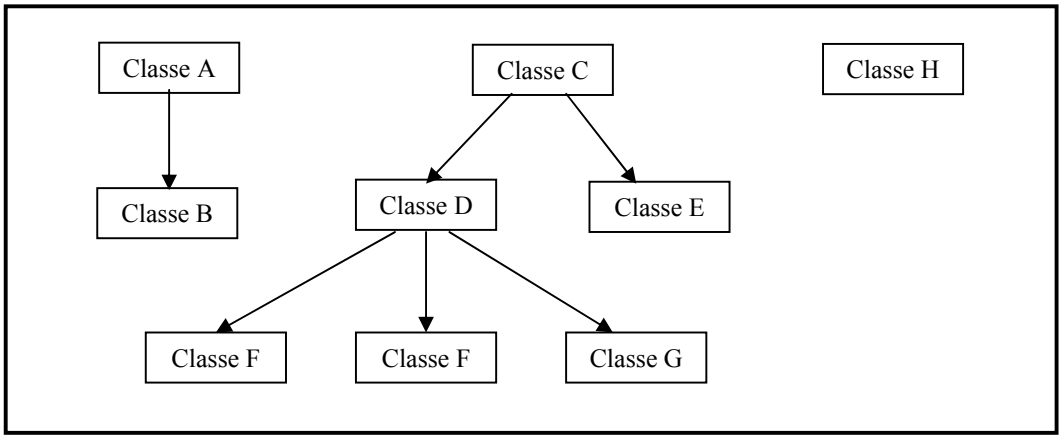

Figure 6.3: Exemple d'un graphe de classes (hiérarchique)

6.4.2 Graphe d'instances d'objets (fonctionnel)

Ce graphe représente toutes les instances d'objets qui sont les instances des classes du graphe hiérarchique. Les nœuds de ce graphe représentent les instances d'objets de l'application et les relations entre objets sont représentées par les arcs du graphe. Ce graphe permet de déterminer les quantités d'informations transférées entre les objets (l'information est exploitée durant le partitionnement).

Il est possible de distinguer deux types de relations:

- la relation « utilise » ou «communique »: qui signifie qu'un objet utilise un autre objet;
- la relation « a-un » ou « est-composé-de »: qui signifie une relation de composition, où un objet est un composant d'un autre objet. La figure 6.4 présente le graphe d'instances d'un système quelconque.

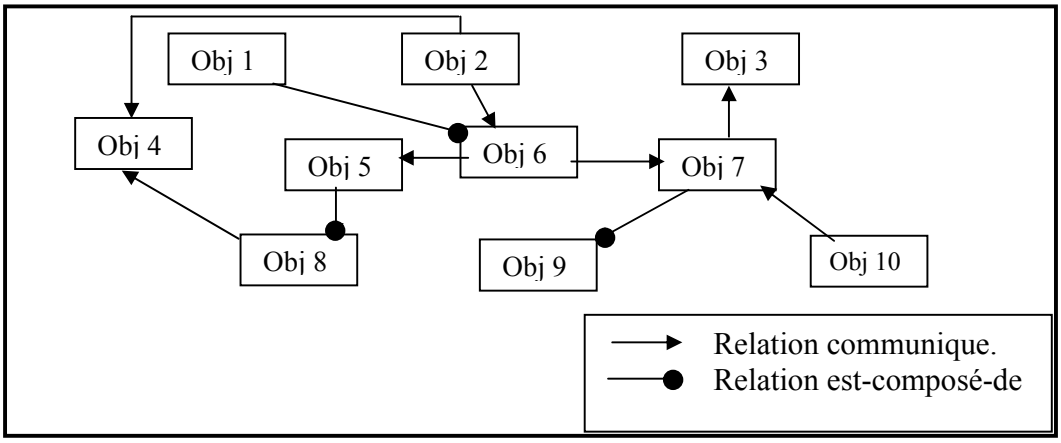

Figure 6.4: Exemple d'un Graphe d'instances d'objets.

6.5 Conception de l'analyse

Dans la littérature, certains travaux ont abordé des problèmes rencontrés durant l'étape d'analyse. Ils font émerger des solutions générales permettant de traiter ces problèmes, et affirment que l'analyse de la cospécification pour le processus de codesign constitue une étape très importante, car elle permet d'extraire un certain nombre de paramètres et de caractéristiques du système, qui ont aidés les concepteurs lors des étapes ultérieurs du processus de conception. Par exemple: extraire les paramètres de temps, d'espace et de coût de communication pour guider l'étape du partitionnement ainsi qu'extraire les caractéristique de communication, les arcs de communication, les liens de communication etc. afin de synthétiser les interfaces de communication entre les différentes parties du système.

Dans cette approche, nous essayons de développer une technique qui nous permettra d'analyser le code de la cospécification Java de l'application. Nous nous sommes basés sur l'aspect communication entre objets, car c'est notre objectif dans ce travail.

La première étape consiste à détecter toutes les méthodes membres des objets, les appels des méthodes et leurs paramètres d'appels. Ces méthodes constituent les seuls points d'accès aux données internes. Nous avons proposés quelques structures de données afin de garder les informations à extraire et de faciliter l'opération d'analyse.

Durant l'opération d'analyse, nous avons cité quelques problèmes et quelques cas particuliers auxquels nous avons proposé quelques solutions.

6.5.1 Les informations à extraire de la spécification Java

Comme mentionné précédemment en section (6.3.1.), nous nous intéressons à l'aspect communication entre objets de l'application, afin d'extraire toutes les informations et les paramètres de la communication. A la fin de l'analyse, les informations sont représentées graphiquement sous forme de deux graphes (graphe d'objets et graphe des classes). Pour cela, les informations que nous avons extraites sont:

- Informations sur les classes: l'analyse permet d'identifier chaque classe de l'application pour extraire son nom, les classes ascendantes (héritage), les interfaces implémentées, les données et les objets membres et les méthodes membres, etc.
- Informations sur les méthodes: pour chaque méthode membre d'une classe, on doit identifier son nom, la liste des paramètres d'appels en entrée et en sortie, la liste des appels des méthodes d'autres objets, etc.
- Informations sur les appels des méthodes: l'analyse de la cospécification doit recenser tous les appels des méthodes et de déterminer pour chaque méthode membre d'une classe: l'objet appelant, l'objet appelé, la méthode appelante, la méthode appelée, la liste des paramètres d'appels en entrée et en sortie, etc.
- Informations sur les instances d'objets: pour chaque instance d'objets, on doit déterminer sa classe d'appartenance, ses méthodes membres et les appels des méthodes effectués par cet objet.

6.5.2 Structures de données proposées et hiérarchie des structures de données

Dans ce qui suit, les principales structures de données servant de noyau à la technique d'analyse sont introduites.

6.5.2.1 Structure générale d'un programme de cospéfication Java

Un programme Java est plus simple qu'un programme C ou C++. Il comporte trois parties optionnelles: une partie pour la déclaration des packages, une partie pour importer des classes située dans un package, et une partie pour la déclaration des classes et des interfaces.

Une classe en Java peut comporter des classes membres internes, des membres données ou des objets membres, et des méthodes membres. La figure 6.5 illustre un exemple de structure générale d'un programme de spécification Java.

```
package NomDuPackage;        // Une éventuelle déclaration de package
import NomDeClasse;          // Une partie pour importer les classes


// Déclarations des classes et des interfaces du fichier

public class NouvelleClasse      // Une seule classe ou interface déclarée public,
{                                // et par convention qui porte le même nom que le fichier
  // Corps de NouvelleClasse
  class NouvelleClasse1          // Une classe interne
  {
   // Corps de NouvelleClasse1
  }
   // Le reste du code
}

class NouvelleClasse2
{
// Corps de NouvelleClasse2
}

interface NouvelleInterface
{
// Corps de NouvelleInterface
}
```

Figure 6.5: Exemple de structure générale d'un programme de spécification en Java

6.5.2.2 Hiérarchie des structures de données proposées pour le graphe des classes

La figure 6.6 présente une représentation hiérarchique de nos structures de données proposées pour faciliter l'opération d'analyse. Une application est composée d'une ou plusieurs classes (C_Classe). Elle peut implémenter des interfaces (C_Interface). Chaque classe comporte des méthodes membres, des données ou des objets membres. Elle peut comporter d'autres classes dites classes internes. Chaque méthode membre d'une classe peut comporter des appels aux méthodes d'autres objets.

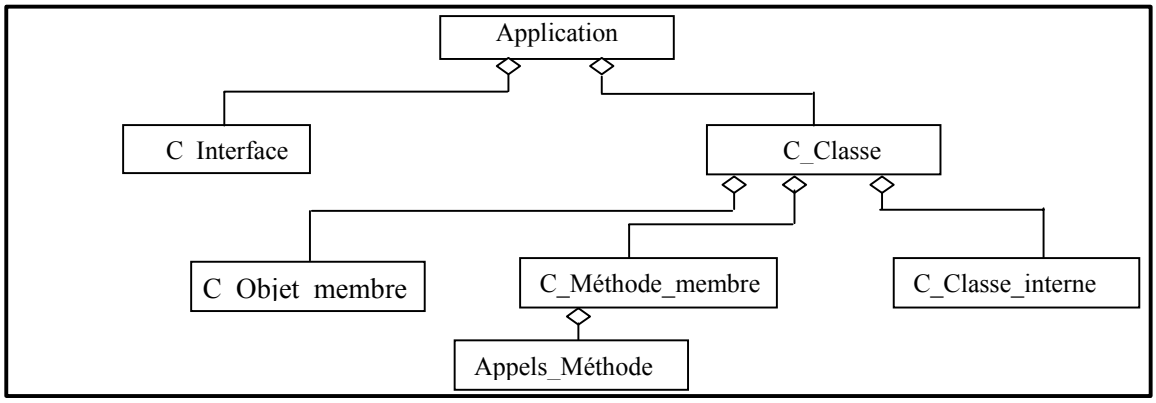

Figure 6.6: Hiérarchie des principales structures de données proposées pour le graphe des classes.

6.5.2.3 Hiérarchie des structures de données proposées pour le graphe d'objet

Le graphe d'objet constitue le graphe d'instance du graphe des classes. Il est inclus dans le graphe des classes. Chaque objet a une classe à laquelle il appartient. Il a des propriétés et des méthodes membres. Une méthode membre comporte à son tour des appels aux méthodes appartenant à d'autres objets. Nous nous sommes basés sur les appels de méthodes qui constituent les opérations de communication entre objets. La figure 6.7 illustre les structures de données proposées pour le graphe d'objets.

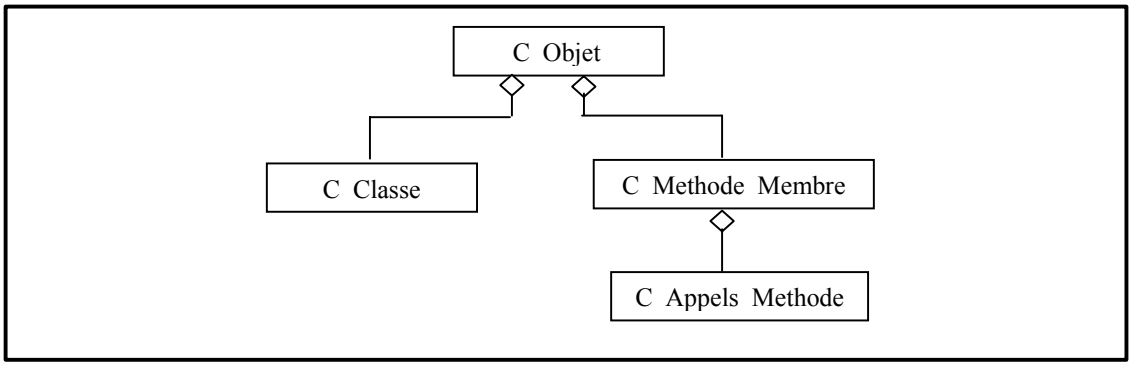

Figure 6.7: Hiérarchie des principales structures de données proposées pour le graphe d'objet.

6.5.3 Déroulement de l'analyse de la cospécification

L'analyse de la spécification Java commence par l'élimination des directives de compilation telles que les inclusions des classes situées dans d'autres fichiers (*Import*). Toutes les classes du programme sont alors groupées dans un seul fichier pour simplifier l'opération d'analyse.

6.5.3.1 Détection des classes et de leurs caractéristiques

Comme nous l'avons mentionné précédemment, la première phase de l'analyse consiste à extraire toutes les informations sur les classes. La structure de données C_Classe proposée encapsule toutes les informations sur une classe de l'application: le nom, les classes ascendantes, les interface implémentées, les objets et les méthodes membres, etc. La figure 6.8 présente un exemple.

```
class Entite1 extends EntitéX
 {   private String Nom;
     public int R1, R2;
     void Methode1 (int E1, int E2)
     {
     R1=E1;
     R2=E2;
    }
    String public Methode3 ()
    {    return Nom;
     }
}
```

NomClass	Entité1
Textends	EntiteX
Timpléments	null
List<TDataMembre>	R1, R2, Nom
List<TMethoMembre>	Methode1, Méthode3.
List<TClassInterne>	null

Figure 6.8: Exemple de détection d'une classe et la structure de données correspondante.

6.5.3.2 Détection des méthodes membres des classes

Chaque méthode doit être identifiée par son nom, ses paramètres d'appels en entrée et en sortie. Elle peut être contenir des appels à des méthodes d'autres objets. La figure 6.9 illustre un exemple d'identification des méthodes membres d'une classe.

```
public void Methode_X (int i, int j)
 {
  Entite2 En2=new Entite2 () ;
  En2.MethodeY (i, j) ;

 }
```

NomMethode	Methode_X
Type_Visiblite	public
Type_Retour	void
List<TParamApp>	i, j
Nbr_ParamApp	2
List<TApplMethode>	MethodeY

Figure 6.9: Exemple de détection des méthodes membres et la structure de données proposée

6.5.3.3 Détection des appels aux méthodes

L'analyse des méthodes appelantes dans la spécification de l'application permet de déterminer les appels aux méthodes. Ces appels constituent les opérations de transfert de données entre objets. Pour chaque appel de méthode, on doit extraire les informations suivantes: le nom de l'objet appelant, le nom de la classe appelante, le nom de la méthode appelante, la liste des paramètres d'appel en entrée et en sortie, le nom de la méthode appelé, le nom de la classe appelé, et le nom de l'objet appelé. Ces informations sont insérées dans une table afin qu'elles soient utilisées par la suite du processus de conception. La figure 6.10 présente un exemple d'extraction des appels aux méthodes.

```
class Ma_classe1
 { private String Nom;
  public void MethodeX (int E1, int E2)
  {
  Ma_classe2 obj;
  obj=new Ma_classe2 () ;
  obj. AfficherLigne (i, j) ;
  }
  String public GetNom ()
  {
   return Nom;
  }
}
```

```
class Ma_classe2
{
Graphics g;
public void AfficherLigne (int E1, int E2)
{
 g.DrawLine (0, 0, E1, E2) ;
 }
}
```

NomObjAppelant	Un_Objet (exemple)
NomClassAppelante	Ma_classe1
NomMethodeAppelante	MethodeX
NomObjAppelé	obj
NomClassAppele	Ma_classe2
NomClassAppele	AfficherLigne
List<ParamètresAppel>	I,j

Figure 6.10: Exemple de détection des appels aux méthodes et la structure de données proposée

6.5.3.4 Instanciation réelles des objets du système

Comme nous l'avons introduit précédemment, la deuxième phase de l'analyse consiste à instancier réellement les objets, et les insérer dans une structure de données qui représente le

graphe d'instances. Pour chaque instance d'objet de l'application, on doit identifier: la classe à laquelle appartient l'objet, ses méthodes membres (identifier pour chaque méthode membre la liste des appels aux méthodes). Enfin, les liens de communication entre objets sont établis.

6.6 Problèmes rencontrés et cas particuliers

Durant l'analyse de la spécification du système, certains cas et problèmes particuliers peuvent se présenter. Dans ce qui suit, nous en présentons quelques exemples.

6.6.1 Cas des classes internes

Contrairement au langage C++, une classe dans le langage Java peut comporter une ou plusieurs classes nommées "classes internes". C'est l'un des avantages de l'utilisation du langage Java comme formalisme de cospécification.

L'utilisation des classes internes permet de diminuer les niveaux de complexité des classes mères. Elles servent au partitionnement multi-niveau [46]. La figure 6.11, illustre un exemple de classes internes.

```
class MonGraphe extends Graphics
{
 private int x1, y1, x2, y2;
 public void init (int p1, int p2, int p3, int p4)
 {
 x1=p1;
 y1=p2;
 x2=p3;
 y2=p4;
 }
 class MaClasseInterne1
 {
 // Le reste du code
}

 class MaClasseInterne1
 {
 // Le reste du code
}
}
```

Figure 6.11: Exemple de classes internes dans le code de cospécification

6.6.2 Problème de détection des instances d'objets

Notre approche consiste à détecter toutes les instances d'objets de l'application à l'aide du modificateur d'instanciation *new*. Dans Java, il n'existe pas seulement l'opérateur *new* pour instancier des objets du système, il y a d'autres opérateurs et méthodes utilisées qui permettent de fournir des instances d'objets de l'application. Nous en présentons dans ce qui suit quelques unes qui sont très utilisées.

- La méthode *newInstances ():* est une méthode de la classe *Class,* qui permet de retourner une référence d'objet.
-La méthode *clonne* (): est une méthode de la classe *Object,* elle permet de créer une copie identique sur l'objet appelant.
-L'utilisation des méthodes qui permettent de renvoyer des références d'objets.

Le problème qui se pose lors de l'utilisation de ces méthodes, vient du fait que les références renvoyées ne portent pas des noms afin de les représenter dans le graphe d'instances. La solution que nous avons adoptée consiste à donner des noms standard pour chaque référence renvoyée par ces méthodes.

6.6.3 Traitement des objets internes ou objets cachés

Les objets déclarés dans les structures de données ou les classes qui sont des objets locaux (des objets internes), ne sont pas traités par l'étape d'analyse car ils peuvent être considérés comme des concepts purement software et ne participent pas au codesign. Par exemple l'utilisation d'un tableau d'objets dans une structure de données, une liste des variables, des objets locaux…etc.

6.6.4 Problème des variables globales et de la cohérence de la mémoire

Supposons que la classe main d'un programme principal de l'application comporte deux classes (classe A, classe B). Après l'étape du partitionnement, supposons que la classe A est affectée au processeur *Pr1,* et la classe B est affecté au processeur *Pr2* (voir l'exemple de la figure 6.12).

Les objets globaux qui sont déclarés dans la classe main sont recopiés dans les deux classes; c'est-à-dire que les deux classes partagent les mêmes copies des variables globales. Le problème qui se pose est lié à la cohérence de la mémoire: quand il y a une modification au niveau d'une seule classe (classe A par exemple) des valeurs des variables globales recopiées.

```
class Main
{   private int x, y;                    // déclaration des variables globales
    public Thread th1, th2;
  class A
  {
  // Le reste du code
  }
  class B
  {
     // Le reste du code
  }
}
```

Figure 6.12: Exemple de problème des variables globales

La solution que nous proposons consiste à insérer un mécanisme de cohérence (protocole de cohérence) de la mémoire. Ce mécanisme consiste à regrouper toutes les variables globales et les insérer dans une table dont la structure est illustrée en tableau 6.1. Le tableau de cohérence décrit, pour chaque variable, toutes ses copies affectées aux différents processeurs de l'architecture cible à l'issue de l'étape de partitionnement.

Noms des variables globales	**Copie 1 Processeur 1 (valeur)**	**Copie 2 Processeur 2 (valeur)**			**Copie N Processeur N (valeur)**
Variable 1	Val 1 …	Val 2 …			Val n …

Tableau 6.1: Table de mécanisme de cohérence de la mémoire

6.6.5 Cas des appels internes de méthodes et des méthodes privées

Dans cette approche, les appels des méthodes privées et les appels internes des méthodes dans la même classe ne participent pas à l'étape de partitionnement. Il est donc inutile de les traiter.

Dans l'exemple de la figure 6.13, l'appel de la méthode *SetNom ("XXXXXX")* ne participe pas au partitionnement, car la classe appelante est la même que la classe appelée.

```
class AppelInterne extends EntitéX
{
 private String Nom;
 public int R1, R2;

 void MethodeX (int i, int j)
 {
  R1=i;
  R2=j;

 SetNom ("XXXXXX") ;
 }

void public SetNom (String UnNom)
 {
  Nom= UnNom;
 }
}
```

Figure 6.13: Exemple d' appels internes des méthodes

Il est inutile de traiter les cas des méthodes déclarées privées, parce qu'on ne peut pas appeler une méthode privée à l'extérieur de sa classe.

6.6.6 Cas de deux instances d'objets portes un nom identique dans des méthodes différentes

Supposons qu'on a deux objets ayant le même nom (Obj1), et qui appartiennent aux deux méthodes différentes Mi, Mj, voir l'exemple présenté en figure 6.14.

La deuxième étape de l'analyse de la cospécification consiste à détecter et recenser toutes les instances d'objets et les insérer dans une table. Chaque objet est identifié d'une manière unique par son nom. Le problème qui se pose concerne l'ambiguïté des noms d'objets dans la table (deux objets portent le même nom) et automatiquement dans le graphe. On ne peut pas savoir de quel objet il s'agit dans le graphe.

Pour résoudre ce problème, nous avons proposé de renommer les deux objets par la concaténation des deux noms: nom de l'objet appelant + nom de l'objet. Comme il est présenté dans l'exemple de la figure 6.14, les méthodes Mi, Mj portent le même nom d'objet (Obj1) dans les deux instances différentes. La concaténation des objets appelants (O1, O2) avec l'objet Obj1 est effectuée et donne: O1Obj1, O2Obj1.

```
class MaClassExemple
{ public Mi ()
  {
   UnInstance obj1= new UnInstance () ;
  }
  public Mj ()
  {
  UnInstance obj1= new UnInstance () ;
 }
}
public class Main
{  public static void main (String Args [])
  {
   MaClassExemple O1= new MaClassExemple () ;
   O1. Mi () ;
   MaClassExemple O2= new MaClassExemple () ;
   O2. Mj () ;
 }
}

class UnInstance
{    // le reste du code
}
```

Figure 6.14: Exemple de concaténation des noms d'objets

6.7 Présentation de l'outil développé pour l'analyse

L'environnement que nous avons développé pour l'analyse de la spécification permet d'introduire la spécification du système sous forme d'un fichier texte (qui est supposé correct du point de vue syntaxique, sémantique et lexical). L'opération d'analyse est ensuite effectuée afin d'extraire et d'afficher des informations concernant les classes, les méthodes et les appels aux méthodes. Prenons un exemple de spécification Java d'un système (figure 6.15).

```
class Compte
{ private float balance;
 Cexemple obj1;
 int quantite;
 public void sauverCompte (int R)
 { quantite=R;
 }
 public void deposer (float montant)
 { obj1 = new Cexemple () ;
  obj1. CexempleDeposer (montant*2) ;
 }
 public float retirer (float montant)
 { float i;
  Cexemple obj2= new Cexemple () ;
  i= obj2. CexempleRetirer (montant) ;
  return i;
 }
 public float getbalance ()
 { return balance;
 }
 }

class Cexemple extends test
{ private float balance1;
 public void CexempleDeposer (float Mts)
 { balance1+= (Mts) ;
 }
 public float CexempleRetirer (float Mts)
 { balance1-= (Mts*2) ;
  return balance1;
 }
 public float Consulter ()
 { Compte ObjC= new Compte () ;
  return ObjC. getbalance () ;
 }
}

public class test
{ public static void main (String [] args)
 { int X1=20; int X2=30;
 Compte Ocpt= new Compte () ;
 Ocpt. deposer (15) ;
 Ocpt. retirer (10) ;
 }
}
```

Figure 6.15: Exemple de spécification Java

La figure 6.16 illustre la partie de l'environnement permettant l'extraction des informations de l'exemple précédent. Les deux graphes (graphe d'instance et de classe) qui représentent les résultats d'analyse, sont présentés dans les figures 6.17, 6.18.

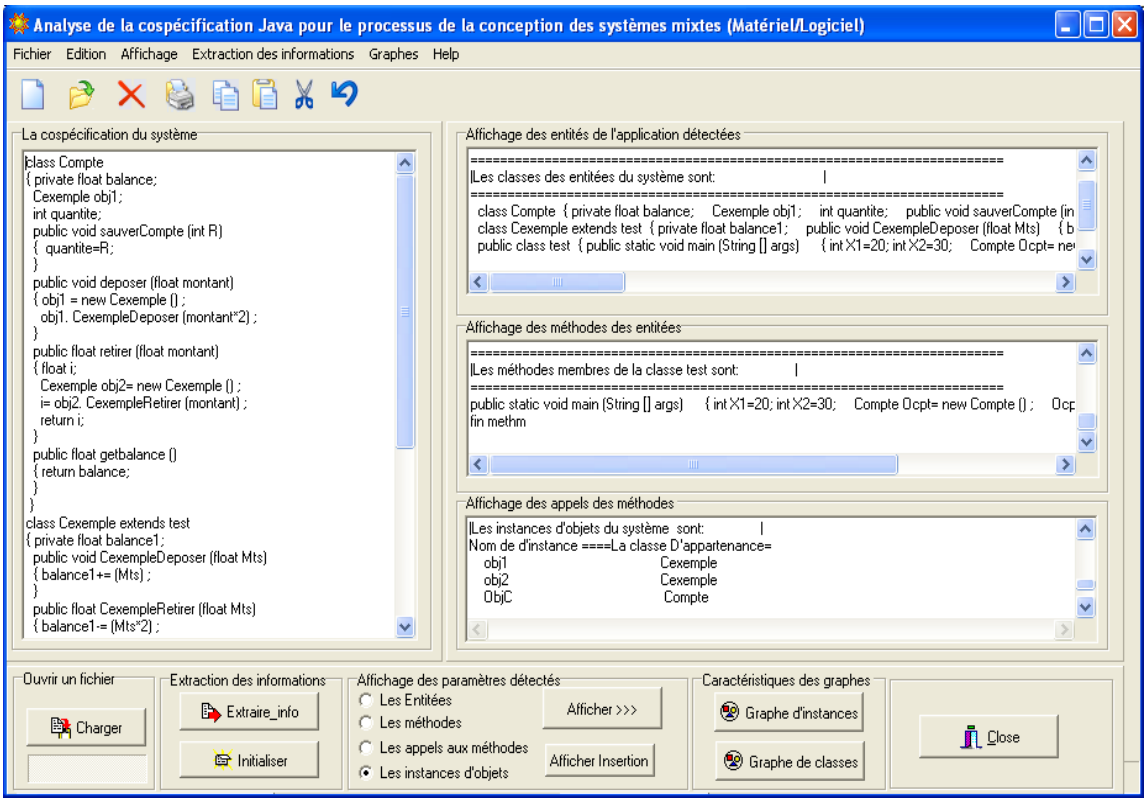

Figure 6.16: Environnement d'analyse pour l'extraction des informations de l'application

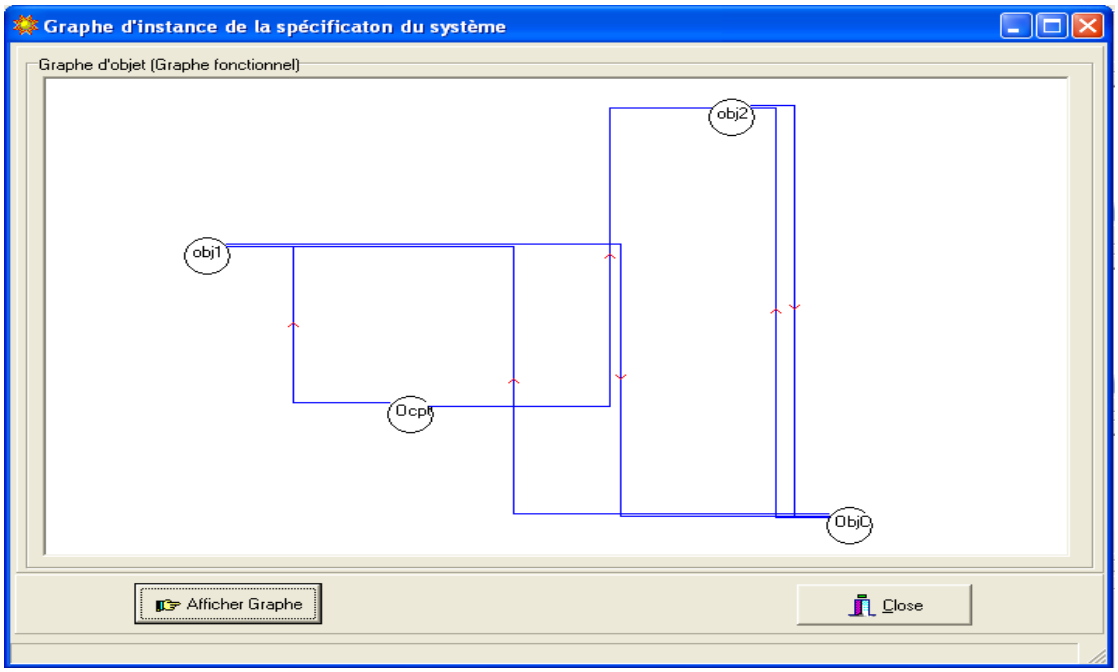

Figure 6.17: Graphe d'instance correspondant à la spécification Java de l'exemple (figure 6.15)

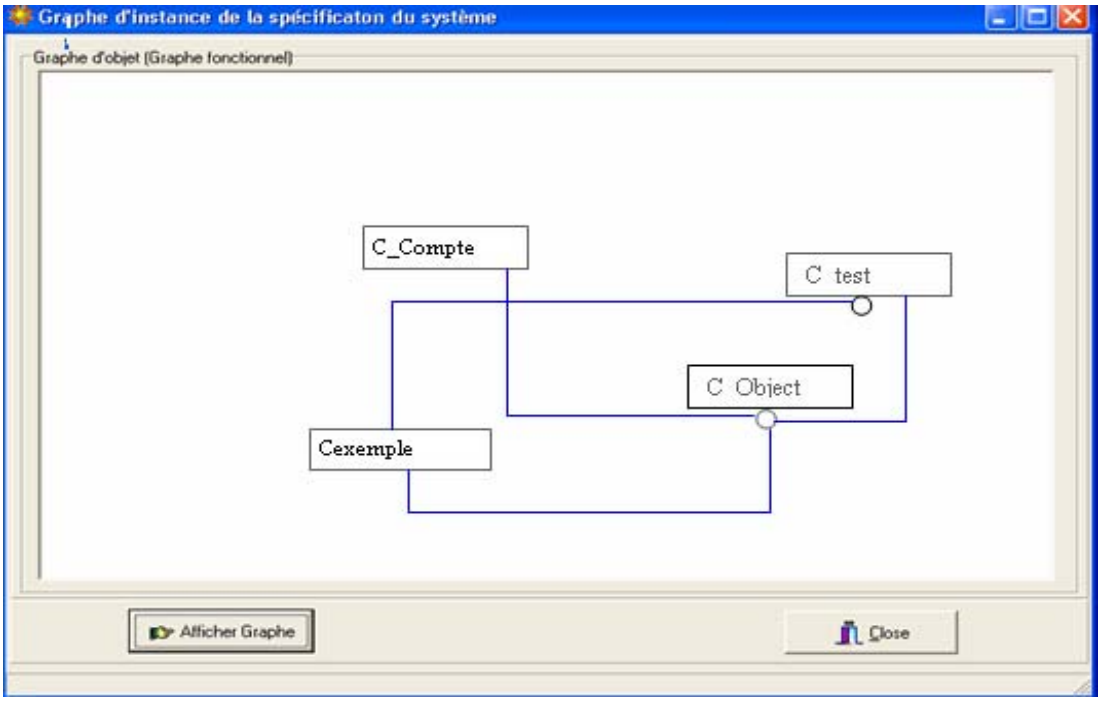

Figure 6.18: Graphe des classes correspondant à l'exemple (figure 6.15)

6.8 Conclusion

Nous avons présenté dans ce chapitre l'approche proposée et l'environnement développé pour l'analyse du code de spécification Java du système. Le module d'analyse n'est pas un environnement d'aide à la programmation, mais plutôt un outil qui permet à l'utilisateur d'introduire le code de la cospécification en Java (le code est supposé correct d'un point de vue syntaxique, sémantique et lexical). C'est d'ailleurs l'un des avantages de l'utilisation du langage Java comme formalisme de cospécification pour les systèmes mixtes vu la disponibilité de compilateurs fiables pour la mise au point préalable de l'application. Ensuite la cospécification est automatiquement analysée afin d'extraire les différentes informations relatives à la communication entre les objets de l'application. Ces informations permettent, dans un premier temps, de construire les deux graphes (graphe fonctionnel et graphe hiérarchique), et dans un deuxième temps d'alimenter les étapes ultérieures du processus de conception (étape de modélisation, étape du partitionnement et de synthèse d'interfaces, etc.). Les structures de données permettant de conserver toutes les informations extraites sont présentées.

Le processus d’analyse d'un langage orienté objet pose un certain nombre de problèmes et de cas particuliers à résoudre. Nous les avons présentés ainsi que les solutions proposées pour les résoudre.

Enfin nous pouvons conclure que l’analyse de la communication du système est une activité complexe qui permet de déterminer un certain nombre de paramètres indispensables aux étapes ultérieures, en particulier l’étape du partitionnement et la synthèse d’interfaces.

CHAPITRE 7
DESCRIPTION DES COMPOSANTS ARCHITECTURAUX ET DES PROTOCOLES DE COMMUNICATION

7.1 Introduction

La spécification des composants de l'architecture cible constitue une activité très importante, car elle représente l'architecture du système sur laquelle l'application s'exécute. Le choix des composants de l'architecture cible dépend fortement de l'analyse du code de la cospécification de l'application et des différentes informations extraites (le temps d'exécution de chaque entité sur chaque processeur, l'espace occupé, la quantité d'informations échangées durant la communication, etc.).

L'utilisation d'un environnement dédié pour la description de l'architecture cible (ex: outil commercial), rend difficile l'extraction des différentes informations et leur transmission à l'outil de partitionnement [46]. Ceci met en évidence l'intérêt de disposer d'un environnement unique pour modéliser à la fois les entités de l'application et les composants architecturaux (processeurs logiciels, processeurs matériels, composants de communication…), afin de disposer de toutes les informations qui servent aux étapes ultérieures du processus de codesign.

Dans cette partie, Nous proposons une technique et un environnement pour la description des composants de l'architecture cible. Pour cela, nous avons fait le choix d'offrir à l'utilisateur la possibilité de composer ses architectures cibles en même temps que l'application qui s'exécute dessus. L'objectif de cette idée est de simplifier la circulation du flux d'informations à travers toutes les étapes du processus de codesign. L'environnement développé est basé sur l'utilisation de bibliothèques de composants architecturaux et de protocoles de communication réutilisables. Une interface graphique permet aux concepteurs des systèmes mixtes de sélectionner ses composants et ses protocoles de communication et de les paramétrer suivant les besoins d'une manière interactive.

7.2 Principales fonctionnalités de l'environnement de description de l'architecture cible

Comme nous l'avons cité précédemment, l'environnement que nous avons développé est basé sur la notion de bibliothèque de composants et de protocoles de communication. Il fonctionne d'une manière interactive. Il permet d'abord, de sélectionner des composants processeurs (processeurs à usages générales et processeurs matériels) à partir de la bibliothèque, et de les paramétrer selon les besoins de l'utilisateur. Pour notre approche de description de l'architecture cible, nous nous intéressons essentiellement aux ports physiques des composants architecturaux. Ces ports constituent les points de communication des composants architecturaux. Pour ce faire, l'outil développé comporte des interfaces interactives permettant aux concepteurs de paramétrer les composants (le nom du processeur, le type, le nombre des ports, la taille maximale des ports, le nom du protocole de communication dans le cas de composants de communication).

Lorsque les processeurs de l'architecture cible sont spécifiés, on doit sélectionner les composants de communication à partir de la bibliothèque des composants de communication, ensuite, connecter les composants en branchant ses ports physiques..

Le choix du protocole de communication permet de gérer la communication entre les ports des composants qui doivent communiquer. Dans cette approche, le protocole de communication est intégré dans le composant de communication (chaque composant implémente un protocole). Donc le choix du protocole est basé sur le choix du composant de communication.

La figure 7.1 présente un exemple de composition d'une architecture cible dans notre environnement de description des composants architecturaux.

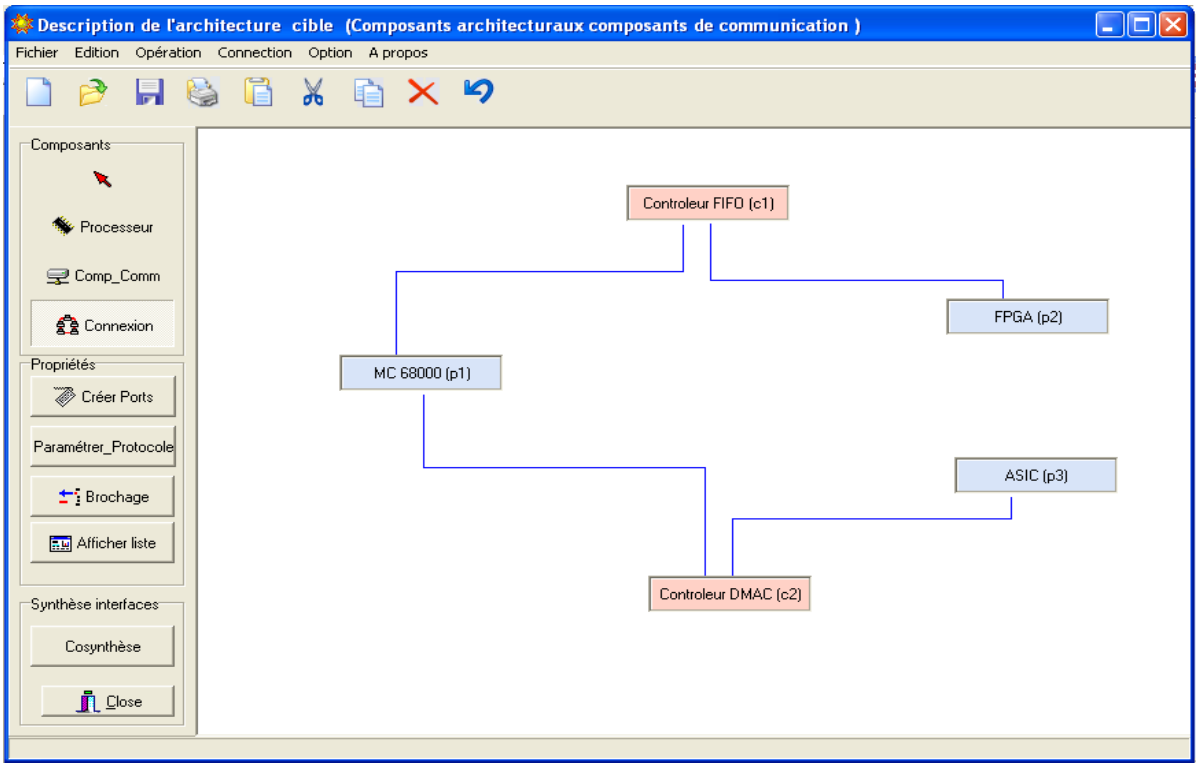

Figure 7.1: Exemple de description d'une architecture cible à l'aide de l'environnement développé

Pour la description des composants architecturaux, nous nous intéressons essentiellement à la description des ports physiques des composants, car notre approche de cosynthèse d'interface consiste à adapter les ports physique des composants aux appels des méthodes des différentes entités de l'application suivant un protocole de communication.

7.3 Composition et description de l'architecture cible

7.3.1 Description des composants processeurs

Les processeurs matériels et logiciels (FPGA, ASIC, Microprocesseurs, Processeur DSP…) sont utilisés pour exécuter tous les traitement de l'application à concevoir. Généralement, chaque processeur contient une unité de traitement, une unité de stockage, une unité interface pour la communication avec d'autres composants, et un dictionnaire de données (pour les processeurs logiciels). Ce dernier regroupe les informations technologiques du processeur auquel il est rattaché.

L'environnement que nous avons développé permet de sélectionner un composant processeur à partir de la bibliothèque. Cette bibliothèque contient les composants processeurs préalablement créés par le concepteur et qu'il pourra réutiliser. Il est ensuite possible de paramétrer le processeur choisi, selon les besoins du concepteur.

La figure 7.2 illustre un exemple de création d'un processeur Matériel à partir de la bibliothèque des composants processeurs (paramètres de création: Nom, Type du processeur, Nombre des ports, Taille des ports). Le même processus se déroule pour la création des processeurs logiciels.

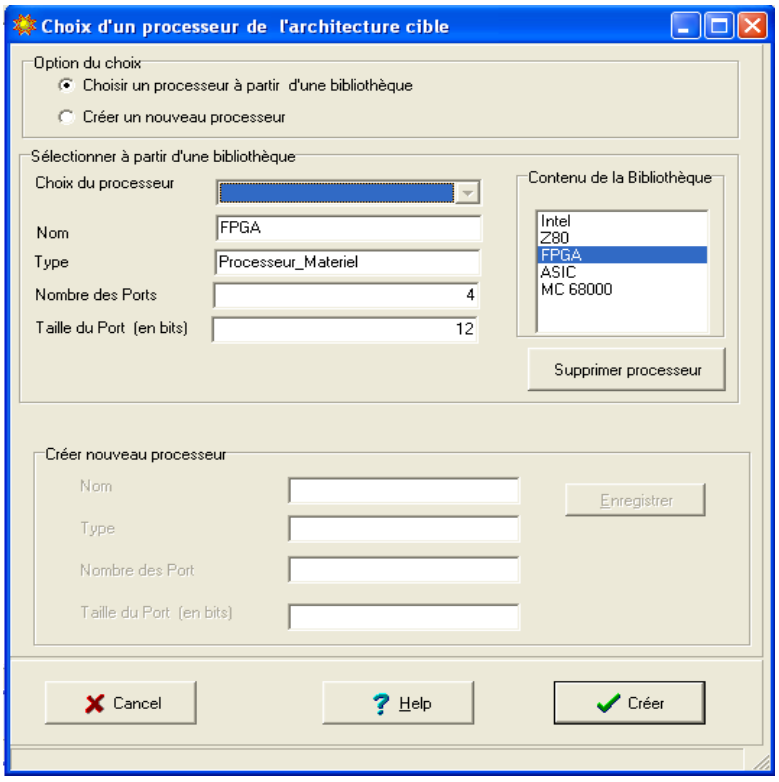

Figure 7.2: Exemple pour la création d'un processeur Matériel

- Description d'un nouveau composant processeur de l'architecture cible

L'outil de composition de l'architecture cible permet également la description de nouveaux processeurs. Grâce à une interface qui offre la possibilité de décrire toutes les caractéristiques du composant à créer, et de les mémoriser dans la bibliothèque des composants afin de les réutiliser.

7.3.2 Description des composants de communication

Pour mettre en oeuvre la communication et l'échange d'informations entre les composants processeurs de l'architecture cible, les unités interfaces des composants processeurs ne suffisent pas. Il est donc nécessaire de disposer de composants spécialisés matérialisant le canal de communication (arbitres, bus, contrôleurs…etc.). Ces composants intègrent des protocoles qui définissent l'ensemble des règles pour mettre en œuvre la communication.

7.3.2.1 Exemple de composant de communication (contrôleur FIFO)

La figure 7.3 illustre un exemple de composant qui peut être utilisé pour mettre en œuvre la communication entre les différents composants processeurs de l'architecture cible. Il s'agit d'un contrôleur FIFO.

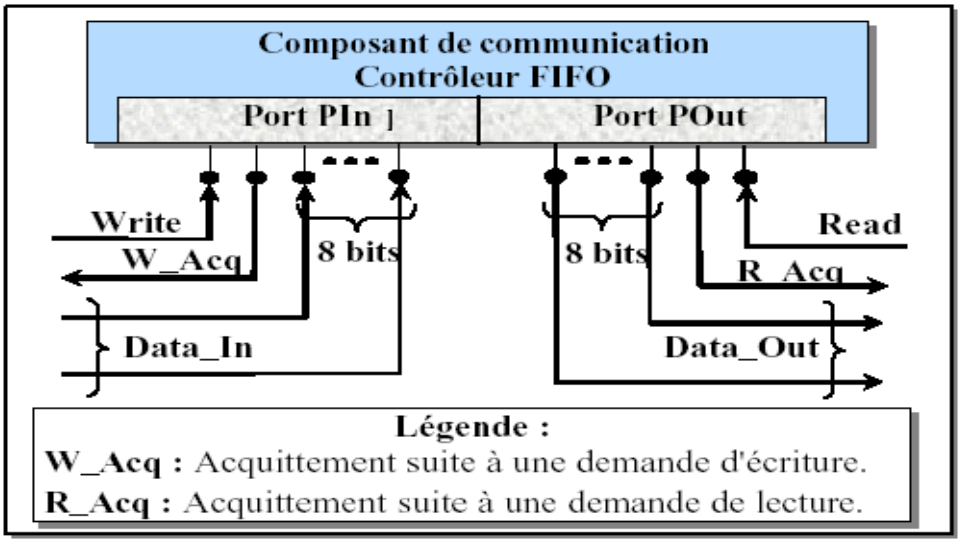

Figure 7.3: Exemple d'un composant de communication contrôleur FIFO [46]

L'unité interface de ce composant contient la spécification de ses ports de communication (PIn, POut). L'unité de traitement contient les primitives du protocole décrivant les opérations de fonctionnement du contrôleur FIFO (Read, Write). Les signaux Write, W_Acq, Read, R_Acq, Data_In et Data_Out sont utilisés pour assurer le dialogue entre les composants qui utilisent ce composant pour communiquer.

7.3.2.2 Création d'un composant de communication

Nous avons fait le choix d'intégrer, sur chaque composant de communication, un protocole qui décrit les règles de communication à travers les différents ports du canal de communication. Notre outil de description des composants architecturaux permet la création des composants de type "composant de communication" à partir d'une bibliothèque de composants réutilisables.

La création des composants se fait de manière interactive et graphique. Elle consiste à sélectionner le composant de communication (selon les besoins du concepteur), afficher ses paramètres, positionner le composant à l'emplacement spécifié afin de le connecter avec les composants processeurs. La figure 7.4 illustre un exemple de création d'un composant contrôleur FIFO: avec 2 ports, une taille de chaque port de 10 bits et un protocole de communication correspondant (FIFO).

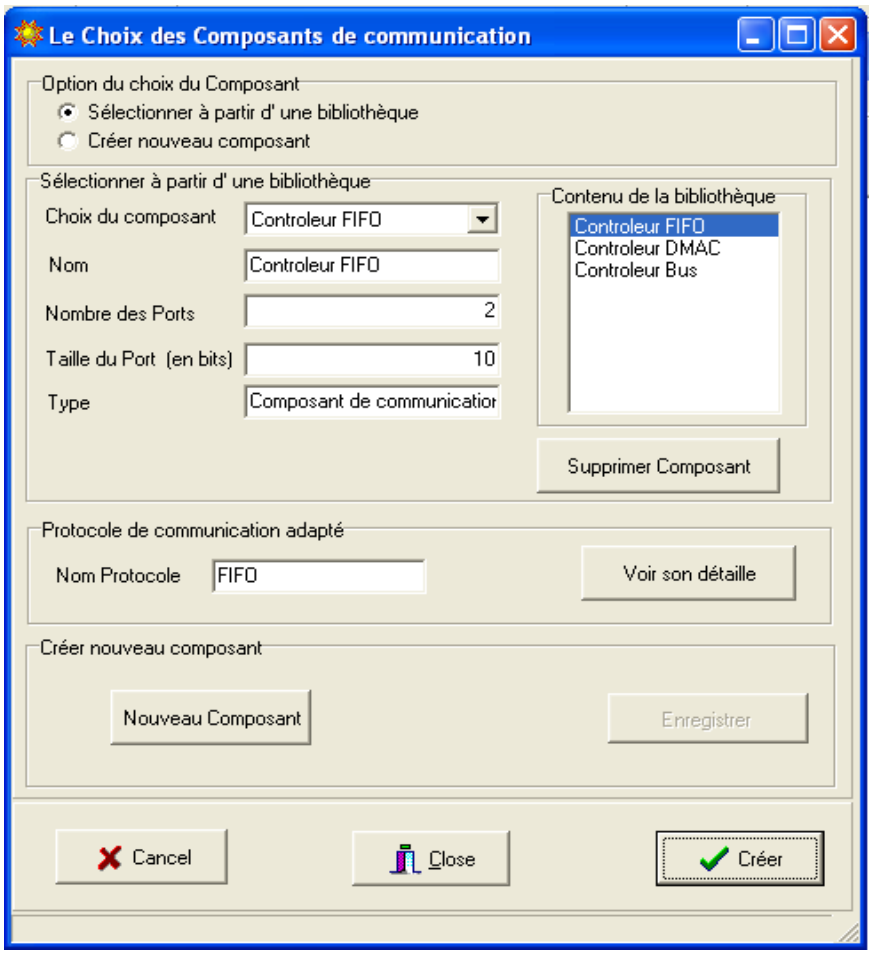

Figure 7.4: Exemple de création d'un composant de communication FIFO.

7.3.3 Choix d'un protocole de communication et connexion des composants

Pour que les composants processeurs échangent des informations entre eux, il est nécessaire de disposer de protocoles de communication qui gèrent la communication. Ces protocoles décrivent les règles de communication à travers les ports d'un canal de communication. Le protocole correspondant est intégré dans chaque composant de communication.

Lorsque les composants qui constituent l'architecture cible sont spécifiés, il faut les connecter en choisissant les ports de connexion, et en y associant les protocoles de communication. Il est important de disposer des informations sur les composants de l'architecture cible, et d'accéder à toutes les informations sur les protocoles, pour pouvoir mener à bien l'opération de synthèse des interfaces de communication.

7.3.3.1 Modèle de protocole de communication

Le protocole de communication est un composant virtuel au même titre que les composants architecturaux [46]. Il est modélisé par un composant encapsulant des données et des traitements. La figure 7.5 illustre le modèle générique de protocole de communication de cette approche. Ce modèle permet de dériver les protocoles standard ou spécifiques.

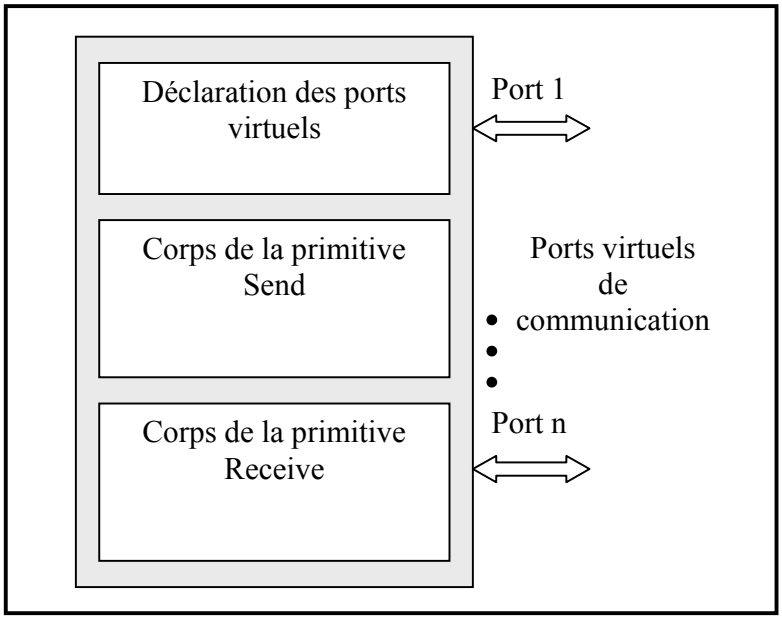

Figure 7.5: Modèle de protocole de communication [46]

L'échange des informations et des données est réalisé exclusivement grâce à deux primitives de communication: (*Send, Receive)*, afin d'éviter d'avoir à traiter de manière particulière toutes les possibilités offertes par Java des différents combinaisons des architectures cibles que l'utilisateur peut composer.

Les primitives de communication agissent sur les ports virtuels. La notion de port virtuel permet de rendre le protocole indépendant des composants qui l'utilisent. L'avantage de l'utilisation des ports virtuels est de répondre à la grande diversité des objets (les types des objets, les tailles, etc.) qui peuvent être transférés comme paramètres d'appels dans une méthode quelconque. Cela permet également d'éviter de réécrire les corps des mêmes primitives *Send/Receive* pour tous les types de données.

Un autre avantage des ports virtuels vient du fait qu'ils permettent de modéliser les types des variables complexes alors que les ports physiques des composants sont matérialisés par des chaînes des bits.

La figure 7.6 illustre une classe générique pour un port virtuel. Cette dernière peut être spécialisée pour décrire des ports en entrées, en sorties, ou bidirectionnels.

```
class CPortVirtuel
{

  private string Type;   //type du port « XXXX »
  private int Size;      // taille du port.
  private boolean Data;  // zone de données.

  public CPortVirtuel (string PortType, int PortSize)
  {

   Type = PortType;
    Size = PortSize;
    Data = new boolean [Size]
  }
}
```

Figure 7.6: Exemple d'un port virtuel.

La figure 7.7, illustre un exemple générique de protocole de communication.

```
class CProtocoleXY
{
 private PIn CPortVirtuel ("Port_in ", 0) ; // une taille 0, signifie que la taille du port
 private POut CPortVirtuel ("Port_out", 0)  // doit être définie dynamiquement

 public void Send (boolean Data_to_send)
 {
  // corps de la primitive
 }

 public void Receive (boolean Data_to_receive)
 {
 // corps de la primitive
 }
}
```

Figure 7.7: Exemple d'un Protocole de communication.

La figure 7.8, présente le détail d'un protocole FIFO rangé dans la bibliothèque des protocoles de communication.

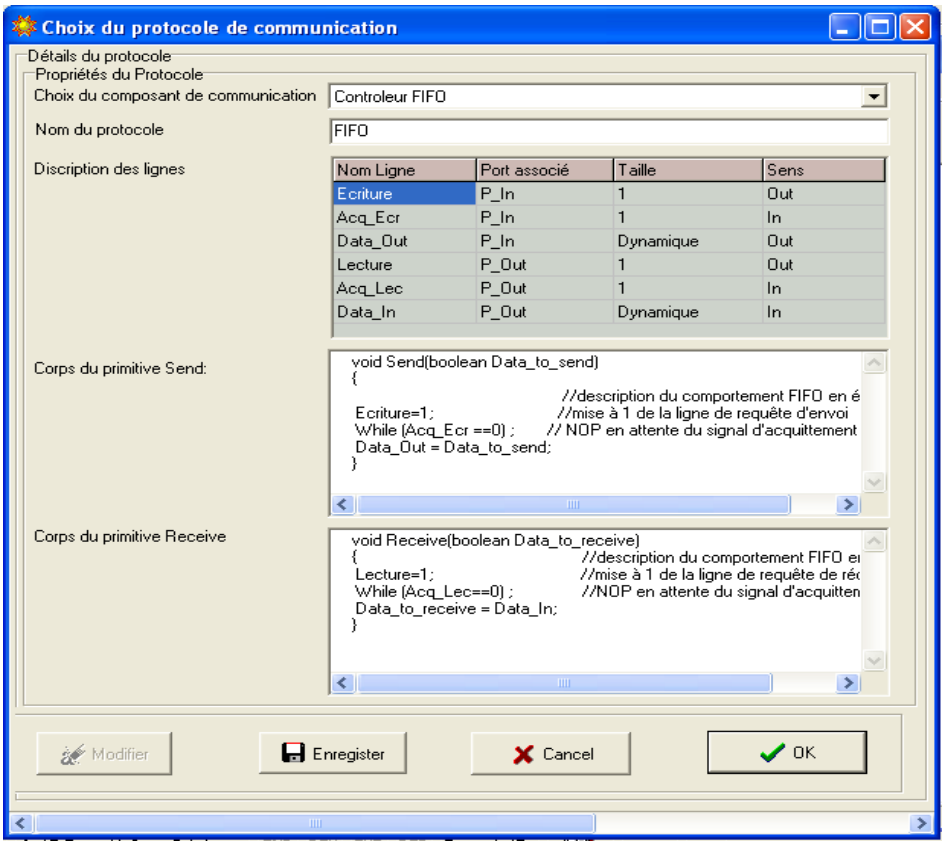

Figure 7.8: Détail du protocole FIFO rangé dans la bibliothèque

7.3.3.2 Bibliothèque des protocoles de communication

La bibliothèque des protocoles de communication regroupe les protocoles de communication qui sont rangés dans la bibliothèque sélectionnés et instanciés lors de la synthèse de la communication. La structure de la bibliothèque de communication de notre approche est illustrée en tableau 7.1.

	Nom du composant de communication	**Primitive Send**	**Primitive Receive**
Protocole xxx	…	Corps de la primitive send	Corps de la primitive receive
Protocole yyy	…	Corps de la primitive send	Corps de la primitive receive
…	…	…	..

Tableau 7. 1: Structure de la bibliothèque de protocoles.

En effet, l'utilisation de la bibliothèque des protocoles de communication permet de rendre l'opération de synthèse d'interface indépendante de l'architecture cible et du système d'exploitation. Elle permet aussi de réutiliser les composants et les primitives existants. Elle dissocie totalement les composants de communication de la politique de transfert des informations entre composants architecturaux.

7.3.3.3 Description de nouveaux protocoles

La création d'un composant de communication nécessite d'instancier un protocole à partir de la bibliothèque. Ce protocole définit la politique d'échange d'informations entre composants architecturaux. Il est paramétré au fur et à mesure de la synthèse de la communication. Notre environnement de description des composants architecturaux et des protocoles de communication permet de créer et de décrire de nouveaux composants et protocoles de communication. La figure 7.9 présente une partie de l'outil consacrée à la création de nouveaux composants de communication.

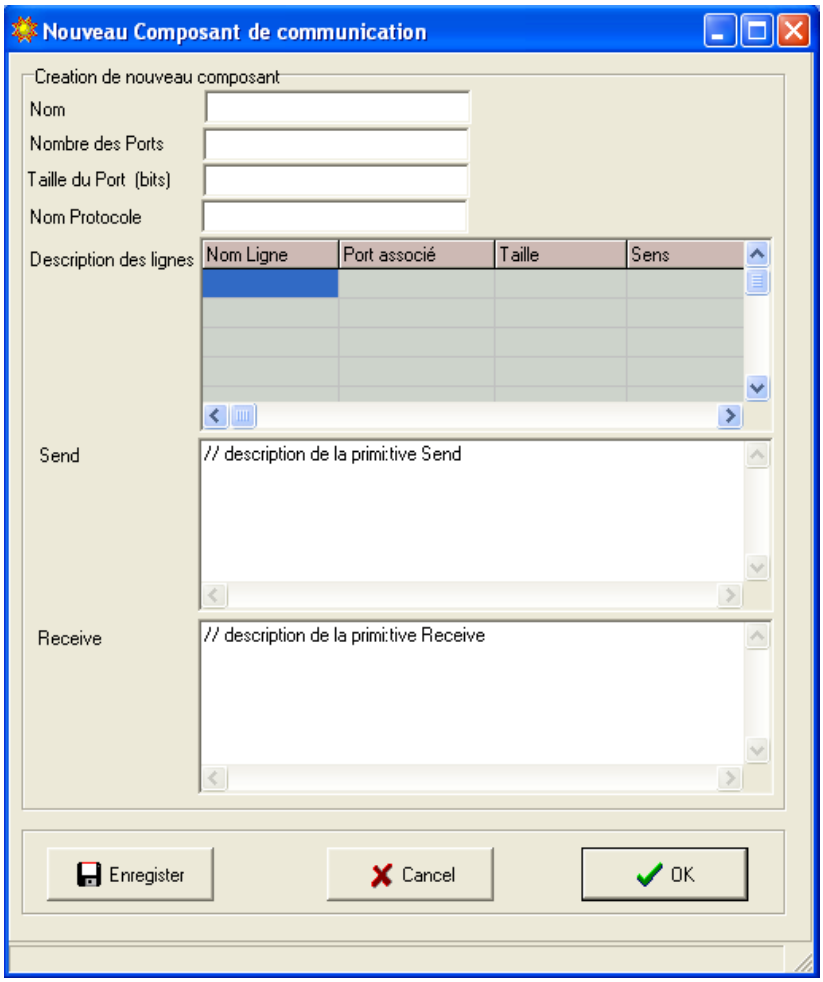

Figure 7.9: Description de nouveaux composants et protocoles de communication.

7.3.4 Description des lignes des ports physiques des composants architecturaux

Lorsque les composants architecturaux sont spécifiés, il faut donner pour chaque composant la description de ses ports physiques (lignes). La description des ports physique consiste à donner pour chaque ligne du port: le nom (br1, br2, …), le type de la ligne (In, Out, InOut) et la taille de la ligne. La figure 7.10 illustre un exemple de description des lignes physiques d'un composant processeur.

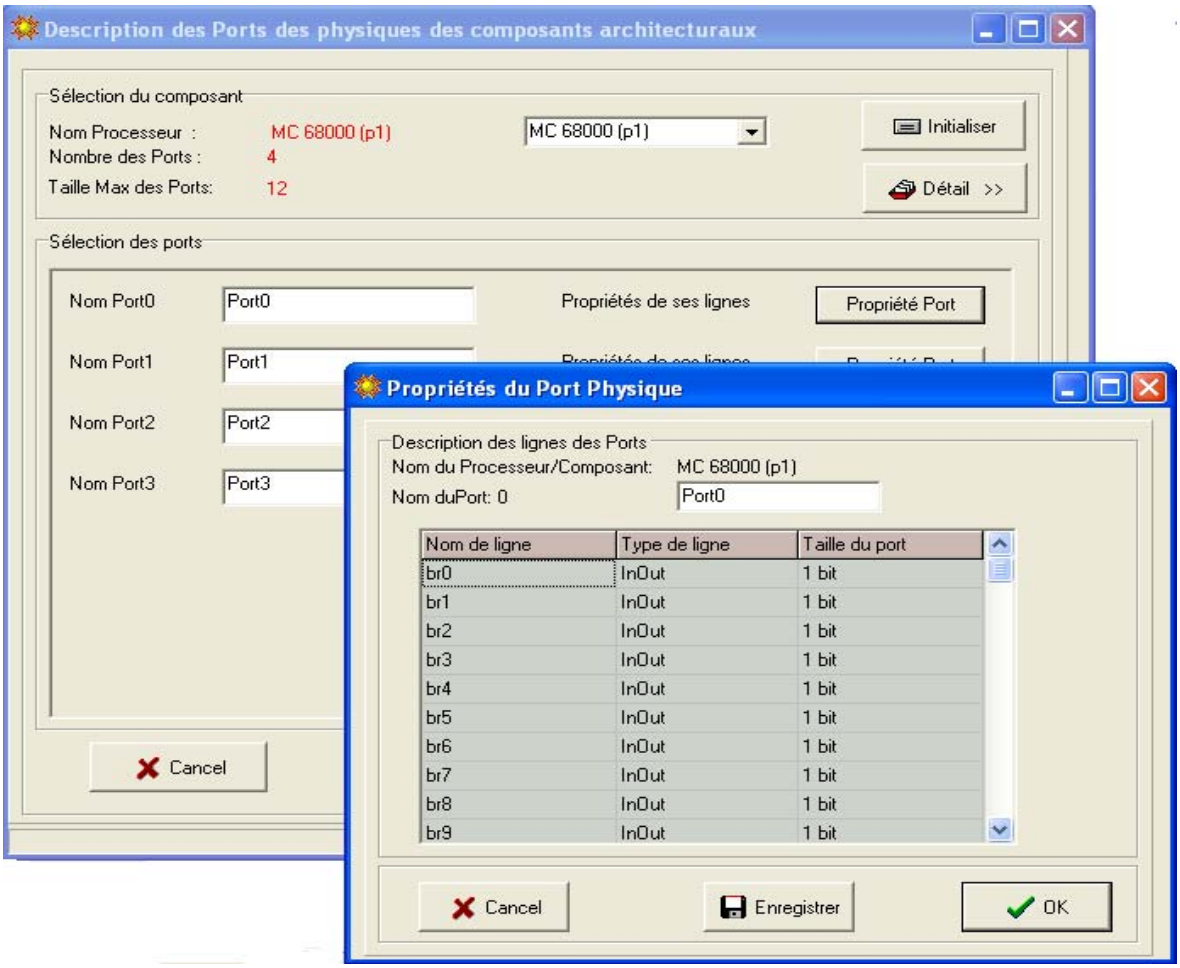

Figure 7.10: Exemple de description des ports physiques des composants architecturaux.

7.3.5 Connexion et brochage des lignes des ports physiques des composants

Pour connecter et brocher les lignes des ports physiques, il faut d'abord identifier les composants qui participent à une connexion donnée, c'est-à-dire déterminer les deux processeurs et le composant de communication qui le relie. Le brochage des lignes des ports physiques détermine la manière de relier les ports des unités interface des composants, en connectant leurs broches deux à deux. A cette étape, on peut effectuer un contrôle de compatibilité des broches des ports reliés. Par exemple, on ne peut pas relier deux ports du même type (In/In, Out/Out.)

La figure 7.11 présente un exemple de brochage des ports des composants architecturaux. Une sélection des composants est effectuée, puis les ports et les lignes des ports sont spécifiés afin de les relier.

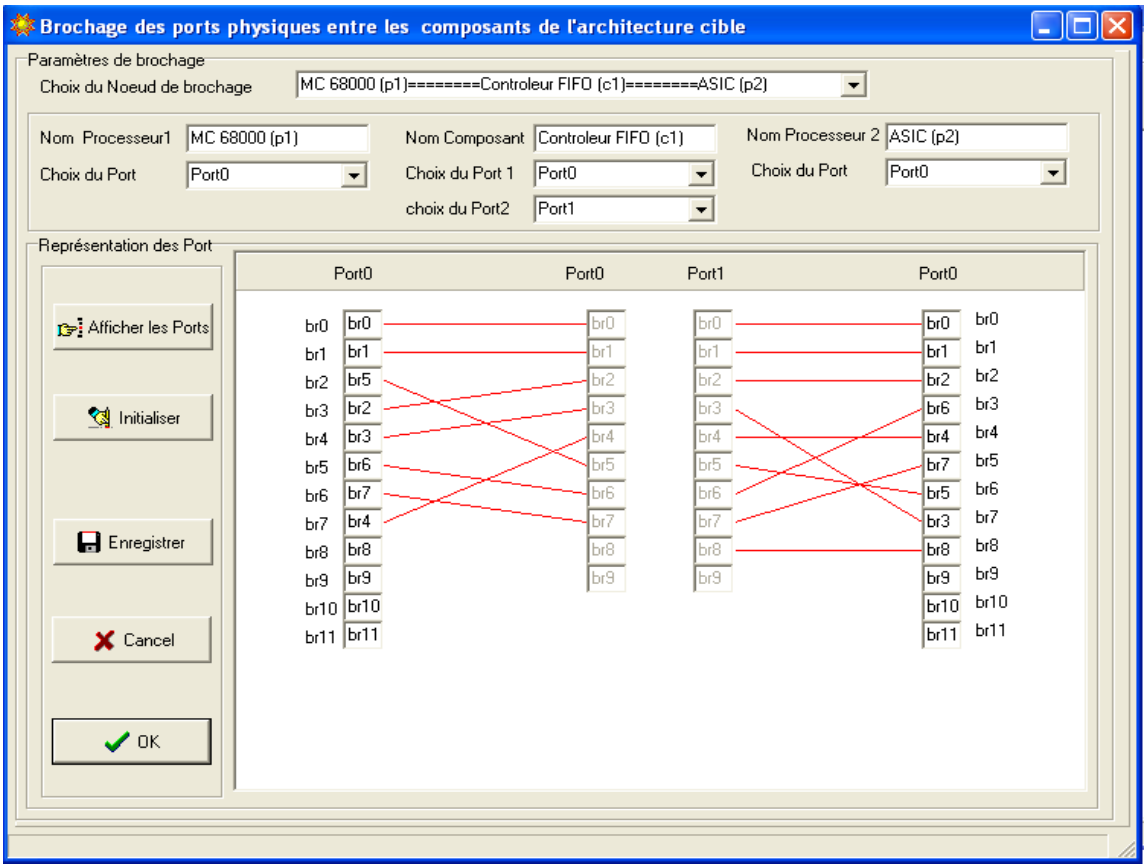

Figure 7.11: Brochage des ports des composants de l'architecture cible.

7.4 Conclusion

Le travail introduit dans ce chapitre présente une technique et un environnement pour la description des composant architecturaux. Il aide les concepteurs des système mixtes à décrire ses architectures cibles. Nous avons décrit les principales activités, en commençant par le choix des composants processeurs à partir des bibliothèques de composants; puis le choix des composants de communication qui intègrent chacun un protocole de communication; et enfin le paramétrage et le brochage des ports physiques des composants.

L'architecture cible peut être décrite d'une manière graphique à l'aide de l'environnement que nous avons développé. Ce dernier permet au concepteur de composer lui-même son architecture en utilisant les composants et les protocoles rangés dans les bibliothèques conçues à cet effet. Cette technique et cet outil permettent la description de nouveaux composants, de nouveaux protocoles de communication et leur réutilisation en les stockant dans la bibliothèque.

L’avantage de cette approche est que l’architecture cible résultante n’est pas figée comme dans de nombreux travaux. De plus, la circulation de l’information entre les différentes activités du processus de codesign est aisée.

CHAPITRE 8
APPROCHE POUR LA COSYNTHESE D'INTERFACES DE COMMUNICATION

8.1 Introduction

L'étude bibliographique des différentes approches de cosynthèse des interfaces de communication souligne, bien que de nombreux travaux soient rapportés dans la littérature, qu'aucune approche ne s'impose comme parfaite ou meilleure que les autres.

Dans beaucoup de travaux, la synthèse des interfaces de communication est réalisée à un niveau de spécification trop bas. Ce dernier ne permet pas d'utiliser des mécanismes de communication évolués (primitives de communication *Send*, *Receive*). Il ne permet pas non plus d'utiliser des mécanismes de systèmes d'exploitation, ni le transfert d'informations par accès direct DMAC. D'autres approches décrivent le système d'une manière complexe [18] ce qui rend difficile l'opération de synthèse d'interfaces. Un certain nombre de travaux utilisent des techniques d'affinement manuel et semi automatique des mécanismes de communication.

Nous présentons dans ce chapitre l'approche proposée pour la synthèse des interfaces de communication dans le processus de conception des systèmes mixte, en détaillant les différentes étapes de cette opération. L'approche développée consiste principalement à partir d'une spécification Java du système à concevoir et d'une description des composants architecturaux, de procéder à des affinements successifs automatiquement et de manière transparente au concepteur. Cette approche dispense l'utilisateur de connaître les détails de bas niveau de l'interface. Elle permet de sélectionner une architecture cible, un protocole de communication grâce à un outil interactif. Ensuite, et à l'issue de l'étape partitionnement, de lancer l'opération de cosynthèse d'interface.

8.2 Présentation générale de l'approche de cosynthèse d'interfaces de communication

Comme nous l'avons cité précédemment, le problème que nous avions à résoudre dans ce travail est la communication entre les objets qui sont affectés aux différents processeurs à l'issue

de l'étape de partitionnement. Dans cette approche, l'échange des informations entre les différents composants de l'architecture cible et les composants de communication se fait à travers des ports de communication et suivant un protocole sélectionné par l'utilisateur. Les protocoles définissent les règles de communication à travers un port. Ils sont modélisés par des couples de primitives « *Send* et *Receive*» qui décrivent la politique d'échange d'informations.

L'approche développée de synthèse d'interfaces dans ce travail est basée sur la notion de la communication à l'aide des primitives virtuelles sur des ports virtuels. Nous présentons dans la figure 8.1 un schéma qui illustre les quatre étapes de cosynthèse d'interfaces de la communication.

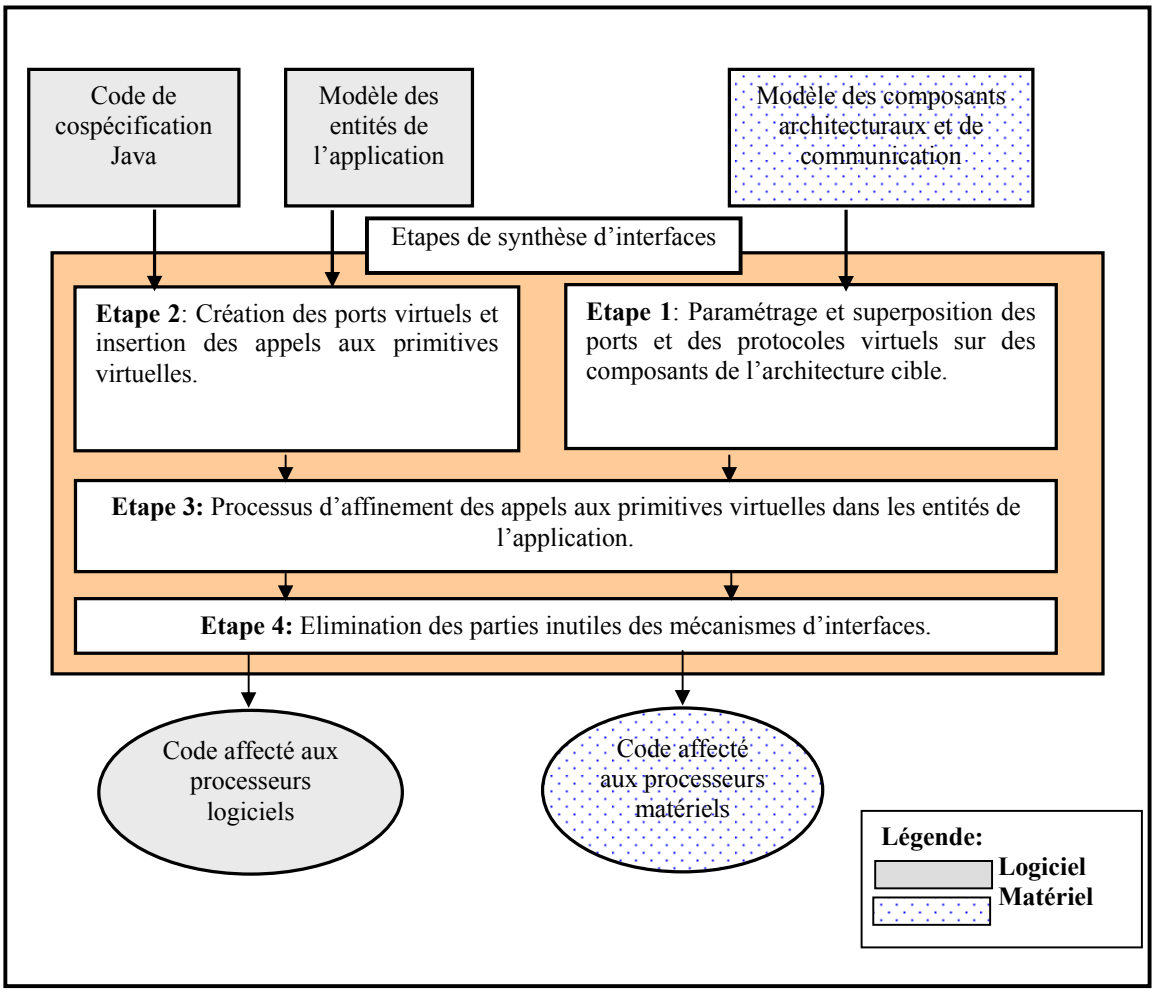

Figure 8.1: Etapes de l'approche de cosynthèse d'interfaces de communication

8.3 Avantages de l'approche proposée pour la synthèse d'interfaces

Parmi les avantages de l'approche proposée nous pouvons citer les suivants:

- L'architecture cible sur laquelle l'application s'exécute est paramétrable (indépendance vis-à-vis de l'architecture et des entités de l'application).
- L'utilisation de deux procédures standard de transfert des informations (*Send*, *Receive*) pour modéliser tous les échanges d'informations, permet de normaliser la communication et d'éviter d'avoir à traiter toutes les possibilités offertes par Java pour l'échange des données en fonction des différentes combinaisons des architectures cible. Ces primitives de communication (*Send*, *Receive*) sont paramétrées au fur et à mesure du processus d'affinement de la communication.
- L'utilisation de ports virtuels permet de répondre à la grande diversité des objets ou des variables (le type des variables, la taille des variables..). Elle permet aussi d'éviter de redéfinir ou de réécrire les mêmes procédures *Send*/*Receive* pour tous les types de données. Les ports virtuels utilisés dans les primitives de communication peuvent être remplacés par des données réelles au fur et a mesure du processus d'affinement.
- Les notions de bibliothèque de composants processeurs, de composants de communication, de protocoles de communication, et de primitives de communication permettent de rendre l'application indépendante de l'architecture cible et du système d'exploitation.

8.4 Etapes de la cosynthèse d'interfaces de communication

Les objets de l'application dans cette approche qui sont affectés aux processeurs logiciels après l'opération de partitionnement, communiquent entre eux grâce à des méthodes publiques (assurant le passage des messages et des informations entre objets de l'application). Les objets qui sont affectés aux processeurs matériels communiquent entre eux grâce à des ports physiques. L'approche proposée consiste à établir la correspondance entre les appels des méthodes des objets de l'application (communication virtuelle de haut niveau) et les ports physiques des composants de l'architecture cible.

L'approche proposée se déroule en quatre principales étapes:

- Etape 1: Paramétrage et superposition des ports et des protocoles virtuels sur des composants de l'architecture cible;
- Etape 2: Création des ports virtuels et insertion des appels aux primitives virtuelles dans le code source de l'application;
- Etape 3: Processus d'affinement des appels aux primitives virtuelles dans les entités de l'application;
- Etape 4: Elimination des parties inutiles des mécanismes d'interfaces.

8.4.1 Paramétrage et superposition des ports et des protocoles virtuels sur les composants de l'architecture cible

Cette tâche est effectuée durant la phase de cospécification des composants de l'architecture cible et des protocoles de communication. Elle permet d'adapter chaque protocole de communication au composant de l'architecture cible qu'il est chargé de relier. Lorsque l'architecture cible du système est spécifiée (description des composants processeurs, des composants et des protocoles de communication), les ports et les protocoles virtuels sont paramétrés et superposés sur les composants architecturaux. Cette étape est effectuée en quatre tâches:

8.4.1.1 Mise en correspondance entre les ports physiques des composants architecturaux et les ports virtuels des protocoles

La mise en correspondance entre les ports virtuels des protocoles de communication et les ports physiques des composants architecturaux est réalisée de manière interactive par le concepteur. C'est le concepteur qui décrit les caractéristiques des lignes des ports (physiques et virtuels), afin d'établir la correspondance entre les lignes des ports (physique et virtuels).

Prenons un exemple de correspondance entre les ports d'une architecture cible illustrée dans la figure8.2, qui est composée de deux processeurs et d'un composant de communication FIFO.

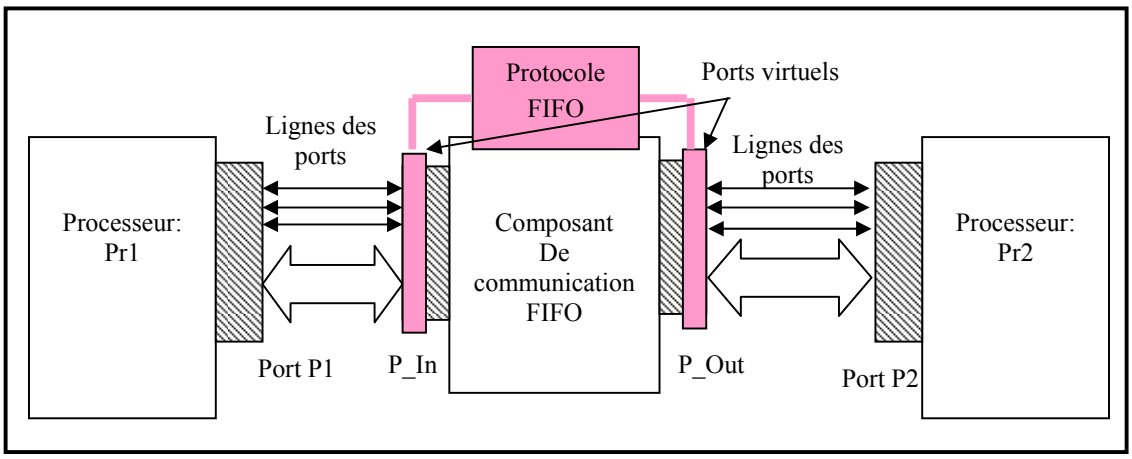

Figure 8.2: Exemple de correspondance entre les lignes des ports virtuels du protocole FIFO et les ports physiques des composants processeurs

Le protocole de communication FIFO utilise plusieurs lignes pour échanger les informations:

Ecriture: demande d'écriture (ligne de contrôle pour Port IN).

Acq_Ecr: acquittement de l'écriture (ligne de contrôle pour Port IN).

Data_Out: émission de la donnée (Port IN).

Lecture: demande de lecture (ligne de contrôle pour Port Out).

Acq_Lec: acquittement de la lecture (ligne de contrôle pour Port Out).

Data_Out: réception de la donnée (Port Out).

Le processeur émetteur utilise les signaux suivants:

W: requête d'écriture.

WA: acquittement d'écriture.

Le processeur récepteur utilise les signaux suivants:

R: requête de lecture.

RA: acquittement de lecture.

La correspondance entre les différentes lignes des composants processeurs et composants de communication est représenté par le tableau 8.1. Le nom du port virtuel, le nom de la ligne virtuel, le composant processeur de mise en œuvre, le nom du port physique, la taille du port, et le sens de la ligne (en entré ou en sortie) sont précisés.

Nom du port virtuel	Nom de la ligne virtuelle	Composant de mise en œuvre	Nom du port physique	Taille en bits	Sens (In, Out)
PIn	Ecriture	Pr1	W	1	Out
PIn	Acq_Ecr	Pr1	WA	1	In
PIn	Data_Out	Pr1	D_Out	8	Out
POut	Lecture	Pr2	R	1	Out
POut	Acq_Lec	Pr2	RA	1	In
POut	Data_In	Pr2	D_IN	8	IN

Tableau 8.1: Table de correspondance entre les ports physiques et les ports virtuels

8.4.1.2 Paramétrage des ports virtuels selon la taille des ports physiques des composants architecturaux

Cette tâche est effectuée suite à la phase de mise en correspondance des ports physiques et des ports virtuels. Elle consiste principalement à affecter la taille du port physique du composant de l'architecture cible au port virtuel du protocole de communication qui lui est associé dans la table de correspondance.

8.4.1.3 Remplacement des ports virtuels des protocoles par les ports physiques des composants dans les primitives de communication

Lorsque la correspondance entre les ports virtuels et les ports physiques est effectuée, on doit remplacer les ports virtuels des protocoles par des ports physiques des composants dans les primitives *Send*, *Receive*. Cette tâche procède au remplacement du nom du port virtuel par le nom du port physique du composant dans le code des primitives *Send* et *Receive*. Les tailles des ports qui étaient déclarés dynamiquement (tant que la taille physique n'était pas connue) sont mis à jour.

Exemple: Supposons qu'on dispose d'une primitive *Send* d'un protocole FIFO. Sa déclaration est illustrée en figure 8.3.

```
void Send(boolean Data_to_send)
{    //description du comportement FIFO en émission
  Ecriture=1;          //mise à 1 de la ligne de requête d'envoi
  While (Acq_Ecr ==0) ;   // NOP en attente du signal d'acquittement
  Data_Out = Data_to_send;
 }
//-------------------------------------------------------------------------------------------------
 Après remplacement des ports virtuels par les ports physiques, la spécification de
la primitive Send devenue comme suit:
//-------------------------------------------------------------------------------------------------
 void Send(boolean Data_to_send)
{   // declaration
    //description du comportement FIFO en émission
  W=1;          //mise à 1 de la ligne de requête d'envoi
  While (WA ==0) ;   // NOP en attente du signal d'acquittement
  D_Out = Data_to_send;
}
```

Figure 8.3: Exemple de remplacement des ports virtuels par les ports physiques.

8.4.1.4 Affectation automatique du code des primitives Send, Receive aux unités de traitement des composants architecturaux

C'est la dernière tâche du processus de superposition des protocoles virtuels sur les composants de l'architecture cible. Elle consiste à affecter le code des primitives *Send* et *Receive* aux unités de traitement des composants architecturaux.

Prenons l'architecture présentée dans l'exemple de la figure 8.2. Le composant (Pr1) joue le rôle d'émetteur. Il reçoit donc le code encapsulé dans la primitive *Send*. Le processeur (Pr2) joue le rôle de récepteur. Il reçoit le code encapsulé dans la primitive *Receive*. Si les composants sont de type processeurs logiciels, le code Java de la primitive est inséré tel quel. A la fin de la phase d'affinement, il est traduit pour d'écrire les ports physiques[2]. Sinon, dans le cas des processeurs matériels, le code de la primitive nécessite une traduction en VHDL pour la synthèse du matériel.

[2]A la fin de la phase d'affinement, une traduction du code java est effectuée, afin de d'écrire les ports physiques (utilisation de l'interface JNI)

8.4.2 Création des ports virtuels et insertion des appels aux primitives virtuelles

Avant la création des ports virtuels et l'insertion des appels aux primitives virtuelles, il faut que nous disposions des informations sur la spécification Java de l'application (surtout, des informations sur les opérations de transferts de données). Ces informations sont obtenues par l'environnement d'analyse que nous avons développé (présenté en chapitre 6).

L'analyse de la spécification Java permet d'extraire un certain nombre d'informations afin de construire les deux graphes (graphe de communication entre objets, et graphe hiérarchique). Elle permet aussi de recenser toutes les opérations de la communication. Les appels des méthodes entre les objets de l'application sont répertoriés et l'analyse des méthodes appelantes et appelées permet d'extraire toutes les caractéristiques de la communication. Ces caractéristiques sont: le sens des données (en entrée ou en sortie) et la taille des données émises et reçues.

8.4.2.1 Création des ports virtuels

Cette tâche consiste à associer à chaque variable qui est référencée dans l'entête de chaque méthode un port virtuel. Afin de déterminer le type et la taille du port, on doit effectuer une analyse sur les appels des méthodes, cette analyse nous permettant de savoir:

- La taille du port: la connaissance du type de la variable à transférer (integer, boolean, string….), permet de déterminer la taille du port à créer (nombre de bits nécessaires au transfert).

- Le sens ou le type de port: le sens du port à créer dépend du type de la variable qui lui est associée. Si la variable est en entrée, le port à créer est en entrée (Port_In). Si la variable est en sortie, alors le port est en sortie (Port_Out). S'il est dans les deux sens, le port est déclaré en entrée/sortie (Port_InOut).

Exemple: Prenons l'exemple illustré en figure 8.4. L'objet appelant de la classe *CEnAppelante* possède une méthode *MEappelante*, qui appelle la méthode *MEappele* de l'objet de la classe *CEnAppele*. Il fournit des paramètres en entrée, et attend un ou plusieurs paramètre(s) de retour. L'objet appelé reçoit des paramètres en entrée. Ensuite il renvoit éventuellement des paramètres de retour.

```
class CEnAppelante
{
 private int P1, P2 ;

 public void MEappelante(P1,P2)
{
 CEnAppele.MEappele(P1, P2);
 //le reste des traitements...
}
}
```

```
class CEnAppele
{
private  int V1, V2 ;
public int MEappele(V1, V2)
 {
  //le reste des traitements...
 }
}
```

Figure 8.4: Exemple de création de ports virtuels

Dans l'exemple précédent, deux ports virtuels sont crées pour chaque classe dont les caractéristiques sont les suivantes:

- Le nom du nouveau port: c'est la concaténation du nom de la variable associée avec la chaîne du caractère « PV » (afin de désigner qu'il s'agit d'un port virtuel). Les variables P1, P2, V1, V2 deviennent respectivement PVP1, PVP2, PVV1, PVV2.
- La taille du port : est celle allouée à la variable associée (type int, 24 bits).
- Le type (sens) est Port_InOut

• Deux tables sont utilisées pour la création et l'insertion des ports virtuels:

Deux tables sont créées pour l'insertion des ports virtuels. La première comporte tous les noms des entités de l'application et les composants processeurs sur lesquels les entités sont affectées à l'issue de l'étape du partitionnement. La structure de cette table est illustrée en tableau 8.2.

La deuxième table effectue la correspondance entre les variables et les ports virtuels. Elle regroupe également les caractéristiques de ces ports, qui sont mises à jour au fur et à mesure du processus de cosynthèse d'interfaces. Sa structure est illustrée dans le tableau 8.3.

Nom_Entité	Nom_Comp
Appelante	Pr1
Appelée	Pr2
…	…

Tableau 8.2: Table des entités et composant d'affectation après partitionnement

Nom_Entité: Nom de l'entité de l'application.

Nom_Comp: Nom du composant processeur sur lequel l'entité est affectée durant le partitionnement.

Num	Noc	N_Ens	N_Ms	N_Vs	N_Pvs	N_Ps	TyPo	N_Endi	N_Md	N_Vd	N_Pvd	N_Pd	Tpv	Tpp
001 ….	261 ….	Obj1 ….	M1 ….	P1 ….	PVP1 …..	Var1 ….	InOut ….	Obj2 ….	M2 ….	V1 ….	PVV1 ….	Var2 ….	32 ….	08 ….

Tableau 8.3: Table de correspondance et caractéristiques des ports

La désignation des champs du tableau 8.3 est la suivante:

Num: Numéro (clé) du port virtuel, qui permet d'identifier d'une manière unique un port.

Noc: Numéro de l'opération de la communication.

N_Ens: Nom de l'entité appelante (émettrice).

N_Ms: Nom de la méthode appelante (source).

N_Vs: Nom de la variable source.

N_Pvs: Nom du port virtuel source associé à la variable source.

N_Ps: Nom du port physique source associé au port virtuel.

TyPo: Type de port (le sens: In, Out, InOut)

N_Endi: Nom de l'entité appelée (destinataire).

N_Md: Nom de la méthode appelée (destinataire).

N_Vd: Nom de la variable destinataire.

N_Pvd: Nom du port virtuel destinataire.

N_Pd: Nom du port physique destinataire.

Tpv: Taille du port virtuel.

Tpp: taille du port physique.

8.4.2.2 Insertion des appels aux primitives virtuelles dans le code de la cospécification

Cette tâche est effectuée en parallèle avec la création des ports virtuels. Elle consiste à insérer des appels aux primitives virtuelles « *Send*/ *Receive* » dans le code source de la cospécification Java de l'application. En remplaçant les appels aux méthodes classiques qui constituent les opérations d'échange d'informations, par des appels aux primitives qui normalisent l'échange de l'information, et facilitent la superposition sur les primitives de protocole de communication.

Durant l'analyse du code source de la cospécification Java de l'application, les appels aux méthodes sont identifiés par un numéro unique. A chaque détection d'un appel de méthode, une entée est créée dans la table des primitives (sa structure est illustrée en tableau 8.4).

Numéro de la primitive	**Type de la primitive**	**Nom de l'entité appelante**	**Nom de l'entité appelée**	**Liste des numéros des paramètres**	**Nombre de paramètres**	**Identificateur du protocole dans la bibliothèque**	**Méthode Copie**	**Classe d'implé-mentation**
001	Send	Class_i	Class_j	…..	…..	…..		
002	Receive	Class_k	Class_l	…..	….	…..		
…..	…..	…..	….	…..	…..	…..		

Tableau 8.4: Structure de la table des primitives

8.4.2.2.1 Principe d'insertion des primitives Send/Receive

Le principe d'insertion des primitives dans le code de la cospécification de l'application est le suivant:

- A chaque rencontre d'un appel à une méthode publique, une entrée dans la table des primitives lui est créée avec son identificateur et ses caractéristiques. D'un autre côté, l'appel est remplacé par un appel à la primitive *Send*, en spécifiant son numéro dans la table des primitives et les numéros de ses ports dans la table des ports (s'il existe des paramètres à envoyer).

- A chaque rencontre d'une entête de méthode appelée dans le code de la cospécification, une entrée dans la table des primitives est créée. Un appel à la primitive *Receive* dans le corps de la méthode appelée est ensuite inséré, pour recevoir les paramètres en entrée. Les paramètres qui sont déclarés dans l'entête de la méthode appelée sont déclarées à l'intérieur du corps de la méthode.

• Exemple:

Prenons un exemple présenté en figure 8.5, qui illustre l'insertion des appels aux primitive *Send* et *Receive*.

```
class Cexemple
{
 private float balance1;
 private float balConsulter;

public void Deposer(float Mts)
{
 balance1+=(Mts) ;
 ObjCompte.MethodeX(E1,E2);
 }
}
// Après remplacement des primitives
class Cexemple
{
 private float balance1;
 private float balConsulter;

public void Deposer(float Mts)
{
   balance1+=(Mts) ;
   //  ObjCompte.MethodeX(E1);  // Num: numéro de la méthode dans la table des primitives.
    Send(Num, NumE1);          // NumE1: numéro du paramètre E1 dans la table des ports.
}
}
```

Figure 8.5: Exemple d'insertion des appels aux primitives *Send*/*Receive*

❖ Problème

L'étape de création des ports virtuels et d'insertion des appels aux virtuelles est effectuée juste après l'analyse de code Java des entités de la cospécification. Il est alors impossible de connaître le type d'implémentation de chaque objet de l'application, tant que les résultats de l'étape du partitionnement ne sont pas encore connus.

❖ Solution proposée

A cet effet, notre hypothèse consiste, dans un premier temps, à supposer que chaque objet se trouve dans un processeur différent. Cette hypothèse va entraîner l'insertion de mécanismes de communication qui peuvent s'avérer inutiles après l'étape de partitionnement. Par exemple, il est inutile d'insérer des mécanismes de communication entre deux objets qui se trouvent sur le même processeur. A la fin de l'étape de cosynthèse, une tâche d'élimination des parties de mécanismes de communication qui sont inutiles est effectuée.

Les méthodes publiques des objets de l'application constituent les points d'accès et de communication entre les objets. Un objet (*Obj1*) situé sur un processeur (*Pr1*) peut être appelé par un autre objet (*Obj2*) qui se trouve sur le même processeur (*Pr1*), ou sur un autre processeur (*Pr2*). Le mécanisme de communication proposé consiste à créer une copie de la méthode de l'objet en effectuant des modifications sur cette copie. Ces modifications sont présentées en section suivante. La méthode d'origine (méthode initiale) sert à une communication sur le même processeur, alors que la méthode copie sert à une communication entre deux processeurs différents (s'il existe un appel d'un objet situé dans un autre processeur, à travers le canal de communication).

8.4.2.2.2 Modifications effectuées sur la copie de la méthode

Il est possible de distinguer deux cas:

a. Cas des objets appelés

Dans ce cas, la copie de la méthode à "surdéfinir" subit les modifications suivantes:

- Modifier le nom de la méthode, en concaténant son ancien nom avec une version
  Exemple: nouveau nom: ''MethodeX''+'V1'.
- Eliminer les paramètres qui sont déclarés dans l'entête de la nouvelle copie de la méthode, et les déclarer localement.
- Insérer un appel à une primitive *Receive* pour les paramètres en entrée de la méthode Receive (NumM, NP1, NP2, NP3,…) où:

  NumM: est le numéro de la méthode dans la table des primitives.

  NPi: numéro du paramètre i dans la table des ports.

- Insérer un appel à une primitive Send pour les paramètres en sortie de la méthode avant le mot clé « return » et éliminer les paramètres du « return », s'il en existe:

Send (NumM, NP0, NP1,…).

Les champs des tables des ports et des primitives sont mis à jour au fur et à mesure que les valeurs des numéros des paramètres et des méthodes sont connues.

- Exemple:

La figure 8.6 présente un exemple qui illustre à son tour les modifications effectuées sur une méthode d'un objet appelé.

```
class Homme
 { private int age;
  public int GetAge(string nom, int y )
   {
    //reste du code
    return(age)
   }
 }

//La méthode est recopiée et sa copie après modification devient comme suit

class Homme
 { private int age;
  public int GetAge_V1( )
 { string nom, int y ;      // déclarations privées et locales des paramètres de l'entête de
                            //la méthode
  Receive(NM1,Nnom,Ny) ; //NM1: numéro de la méthode dans la table des primitives.
                             // Nnom, Ny: numéros des paramètres Nnom, Ny dans la table
                            //des ports.
  Send(NM2, Nage)           // Nm2: numéro da la primitive dans la table des primitives
                            // Nage: numéro du paramètre age dans la table des ports.
   return() ;
   }
 }
```

Figure 8.6: Exemple des modifications effectuées sur une méthode d'un objet appelé.

b. Cas des objets appelants

Dans ce cas, on doit remplacer l'appel à la méthode par une primitive *Send*, en spécifiant le numéro de la méthode dans la table des primitives, et le numéro des paramètres dans la table des

ports. Les deux tables sont mises à jour au fur et à mesure que les valeurs des numéros sont connues.

Exemple:

ObjCompte.MethodeX(E1,E2), devient ***Send (NumM, NE1, NE2).***

Tel que: NumM, NE1, NE2, les numéros des paramètres et de la primitive dans les deux tables.

❖ Problème de déclenchement des méthodes

Un problème se pose pour déclencher la méthode appelée. Comment procéder pour déclencher une méthode appelée située sur un processeur différent que le processeur de la méthode appelante ?

Ce problème ne se pose pas lorsque l'objet appelant et l'objet appelé se trouvent sur le même processeur. Prenons l'exemple d'un processeur à jeu d'instruction. Après la compilation du code de la cospécification, le module d'édition des liens établit automatiquement les liens d'appels aux méthodes. Ce n'est pas le cas lorsque les deux entités sont sur des processeurs différents (éventuellement dans des langages différents: Java pour le logiciel et VHDL pour le matériel).

❖ Solution proposée:

La solution que nous proposons consiste à envoyer un signal *déclencheur* de l'objet appelant vers l'objet appelé. Ce signal est reçu par l'objet appelé, et sur la base de la valeur de ce dernier, la méthode appelée est déclenchée.

Prenons l'exemple de la figure 8.7 où une architecture cible composée de deux processeurs (Pr1, Pr2) qui communiquent entre eux grâce à un composant de communication contrôleur FIFO.

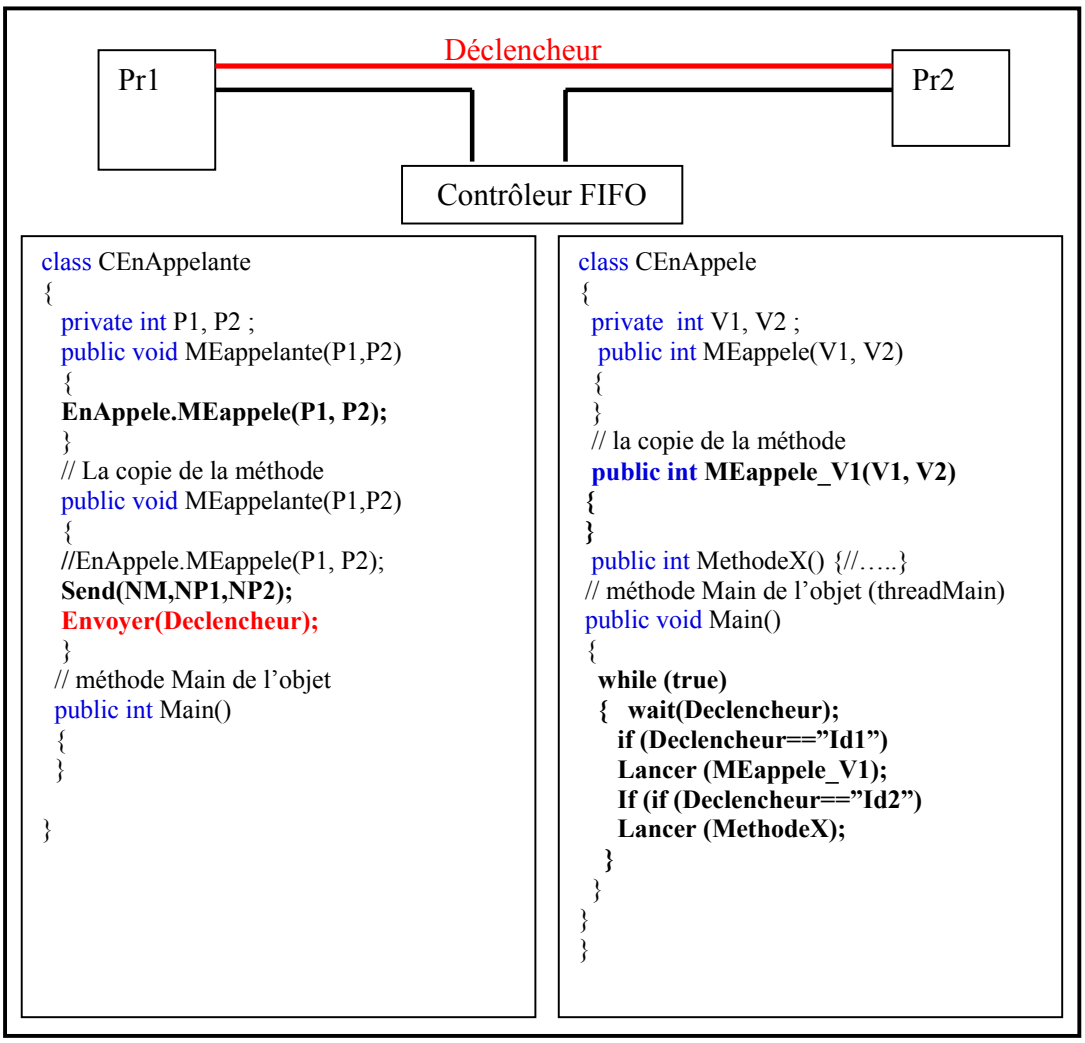

Figure 8.7: Exemple de déclenchement d'une méthode appelée

La figure 8.7 illustre un exemple de déclenchement d'une méthode appelée. Notre solution consiste, à la fin de chaque envoi de tous les paramètres d'appels depuis l'objet appelant à l'objet appelé, à envoyer un signal vers l'objet appelé. Ce dernier possède une méthode main (Thread Main exécutée indéfiniment) qui attend la réception du signal déclencheur. Lorsque la valeur du signal passe à '1', la méthode appelée correspondante est lancée.

8.4.3 Processus d'affinement des appels aux primitives virtuelles dans les entités de l'application

Les résultats de l'étape de création des ports virtuels et de l'insertion des appels aux primitives *Send*/ *Receive***,** est de remplir le modèle de communication par les différentes informations collectées (sur les ports et les appels aux primitives virtuelles). Ces informations sont regroupées dans des tables qui assurent l'échange d'information entre objets. Une fois que les résultats de l'étape de partitionnement sont connus (toutes les informations concernant l'affectation des objets aux processeurs de l'architecture cible), le schéma de communication entre les objets de l'application est connu, ce qui permet de passer à l'étape d'affinement.

Le processus d'affinement consiste à remplacer les appels aux primitives qui sont insérées dans le code des méthodes de la spécification, par les corps de primitives de protocoles paramétrés. Elle se déroule en trois tâches:

- Remplacement des appels de primitives avec paramètres multiples par des appels de primitives simples.
- Insertion des corps de primitives des protocoles dans la cospécification.
- Mise en correspondance des ports virtuels (des appels de primitives de l'application) avec les ports physiques, et adaptation des primitives aux composants processeurs.

8.4.3.1 Remplacement des appels de primitives avec des paramètres multiples par des appels des primitives simples

Le but de la synthèse d'interface de communication est de permettre l'échange d'informations entre les différents composants de l'architecture cible. Cet échange est assuré par les canaux de communication qui sont les bus de l'architecture cible. La taille du bus physique de l'architecture cible limite la quantité d'informations transférée à la fois. Il est donc difficile de transmettre plusieurs paramètres simultanément sur le même bus. Pour ce cas, nous proposons de transmettre un seul paramètre à la fois.

Les appels aux primitives *Send/Receive* insérés dans les corps des méthodes, et qui comportent plusieurs paramètres, sont décomposés en des appels à des primitives élémentaires (transmettant un seul paramètre à la fois).

Exemple:

L'appel à la primitive Send(NM, Npar1, Npar2, Npar3, Npar4) est remplacé par:

```
Send(NM, Npar1);
Send(NM, Npar2);
Send (NM, Npar3);
Send (NM, Npar4);
```

Un problème peut être rencontré quand la taille du paramètre à transmettre est plus grande que la taille du bus physique correspondant. Nous discutons de ce problème dans le prochain chapitre.

8.4.3.2 Insertion des corps de primitives des protocoles dans la cospécification

L'insertion des corps de primitives correspondant au protocole sélectionné et affecté aux unités de traitement des composant processeurs de l'architecture cible, constitue une tâche importante. Elle consiste, à insérer les corps de primitives des protocoles de communication dans les entités qui sont affectées aux composants architecturaux après le partitionnement.

- Exemple:

Prenons un exemple d'insertion du corps de la primitive *Send* pour un protocole FIFO, comme l'illustre la figure 8.8.

Le processus consiste à insérer le corps de la primitive *Send* récupérée dans la bibliothèque de protocoles, et qui est calquée sur les ports physiques du composant processeur sur lequel l'entité est affectée, à l'issue du partitionnement.

```
class Processeur
 {
  private int taille;
  public int GetSize( )
    { ….
  //Obj.lireSize(int size );
  Send(NM1, Nsize)
  // Le reste du code
   return() ;
  }
//Après insertion du corps de la primitive Send récupérée dans la bibliothèque
class Processeur
 { private int taille;
  public int GetSize( )
    {
  //Obj.lireSize(int size );
  Send(NM1, Nsize);
  // Le corps de la primitive Send
 // void Send(boolean Data_to_send)
 // {
   W=1;                          //mise à 1 de la ligne de requête d'envoi
  While (WA ==0) ;               // NOP en attente du signal d'acquittement
  D_Out = Data_to_send;
 // }
 // Le reste du code
   return() ;
  }
}
```

Figure 8.8: Exemple d'insertion du corps de la primitive *Send* d'un protocole FIFO

8.4.3.3 Mise en correspondance des ports virtuels des appels de primitives de l'application avec les ports physiques

Le but de cette tâche est de mettre en correspondance les ports virtuels des primitives insérées dans les entités de l'application, et les ports physiques des primitives des protocoles qui sont affectées aux unités de traitement des composants processeurs à la fin de l'étape de paramétrage. Autrement dit, cette correspondance est la superposition des appels aux primitives insérées dans les entités et les entêtes des primitives affectées aux composants architecturaux.

La superposition (correspondance) est effectuée grâce à toutes les informations stockées dans les deux tables (table des ports et table des primitives). A la fin de cette tâche, les numéros des primitives sont éliminés pour ne garder que les ports.

Quelques problèmes peuvent être rencontrés, à cause des types et des tailles des ports physiques et des ports virtuels. Nous traitons ces problèmes dans le prochain chapitre (Test et mettre en œuvre de l'approche).

Exemple:

```
Send (bool Data_to_send[8]);   //Le port physique est un bus de 8 bits
Send (N°Param);   // N°Param: numéro du paramètre virtuel dans la table des ports.
```

❖ Problème

A la fin de cette étape, les primitives paramétrées qui sont insérées dans les méthodes des objets affectés aux processeurs matériels, sont ensuite traduites en VHDL pour synthétiser le circuit correspondant. Un problème est posé pour les primitives insérées dans les objets qui sont affectés aux processeurs logiciels. Le problème est que le langage de spécification Java ne permet pas de d'écrire ou de définir les ports physiques, alors que nous avons inséré des primitives paramétrées à l'aide des ports physiques du composant processeur sur lequel elles s'exécutent.

Java ne permet pas d'accéder aux détails de bas niveau de l'architecture. Par exemple, il n'est pas possible avec Java de manipuler des cartes d'acquisition, les ports physiques, etc. Cela peut être considéré comme une limite du langage Java.

❖ Solution proposée

Pour d'écrire des ports physiques des composants processeurs, nous proposons une solution qui consiste à mélanger du code écrit dans un autre langage (C++ par exemple) à l'intérieur du code Java. C'est-à-dire qu'on doit traduire les primitives paramétrées à l'aide des ports physiques dans un autre langage C++. Ce dernier permet de d'écrire les ports physiques des composants. Il sera interfacé avec Java pour le reste de l'application. L'interface Java permettant de lier du code natif[3] (implémenté dans un autre langage comme C++) au code Java. Cette interface est désignée par le terme JNI (**J**ava **N**ative **I**nterface).

8.4.4 Elimination des parties inutiles des mécanismes d'interfaces

L'étape de création des ports et d'insertion des primitives virtuelles est effectuée avant l'étape de partitionnement. A ce stade, les résultats du partitionnement ne sont pas connus. Ces

[3] Code natif: c'est un code interfacer dans le langage Java, il a une implémentation dans un autre langage.

résultats permettent de connaître le type d'implémentation de chaque entité de l'application. Il est donc trop tôt pour connaître le type de communication entre les différentes entités de l'application.

Comme nous l'avons cité précédemment, le mécanisme proposé pour résoudre ce problème consiste à créer des copies de toutes les méthodes appelantes et appelées des objets, et d'effectuer des modifications sur ces copies pour assurer la communication entre objets. Certains mécanismes ainsi inclus peuvent s'avérer inutiles après le partitionnement (par exemple si la copie et la méthode initiale sont affectées par le partitionnement au même processeur). Il est par conséquent nécessaire de recourir à une étape qui élimine les parties insérées et qui s'avèrent inutiles afin de minimiser la taille du code source et la quantité de matériel nécessaire pour mettre en œuvre la communication.

Cette étape prend en compte les résultats du partitionnement, afin d'éliminer les branches inutile et les redondances. L'algorithme d'élimination dépend fortement de trois paramètres:

Le nom de l'entité appelante, le nom de l'entité appelé et le processeur sur lequel l'entité est affectée après l'étape de partitionnement. La figure 8.9 présente l'algorithme du processus d'élimination des branches inutiles des mécanismes d'interface.

```
Si le processeur est le même pour toutes les entités appelantes et l'entité appelée Alors
   La communication est interne et seule la version originale de la méthode est conservée;

Sinon

   Si toutes les entités appelantes se trouvent en dehors du processeur de l'entité appelée Alors
         Garder seulement la version de la méthode modifiée;

   Sinon
         Si une partie des entités appelantes se trouvent sur le processeur de l'entité appelée et une
               autre partie est en dehors de ce processeur  Alors

           Garder les deux versions de la méthode (les deux méthodes d'appels sont présents);

         Finsi
   Finsi
Finsi
```

Figure 8.9: Algorithme du processus de réduction des mécanismes d'interfaces.

8.5 Conclusion

Le résultat de la cosynthèse d'interfaces de communication est un système contenant du logiciel, du matériel et des composants de communication.

Nous avons présenté dans ce chapitre une approche qui est entièrement automatique. Cette approche part d'une spécification Java de l'application et de la description des composants architecturaux et des protocoles de communication. Elle remplace les appels aux méthodes publiques des entités de l'application par des appels aux primitives virtuelles *Send*/*Receive*. Ensuite, elle effectue des affinements successifs sur la spécification Java des objets de l'application. L'avantage principal de cette approche est qu'elle procède de manière transparente au concepteur (le concepteur n'a pas besoin de connaître le détail de bas niveau de ces interfaces).

Nous avons essayé de détailler toutes les étapes et les sous-étapes de cette approche de cosynthèse des interfaces, afin de mettre en œuvre l'approche. Le prochain chapitre présente la mis en œuvre de l'approche proposée, avec les problèmes et les cas particuliers rencontrés.

CHAPITRE 9
MISE EN OEUVRE ET TEST DE L'APPROCHE DE COSYNTHESE D'INTERFACE PROPOSEE

9.1 Introduction

Ce chapitre est consacré à l'implémentation et la mise en oeuvre de l'approche proposée pour la cosynthèse de la communication. Nous présentons le jeu de test utilisé pour la mise en œuvre des différentes phases de l'approche proposée. C'est un exemple de cospécification Java d'une application avec une architecture cible sur laquelle elle s'exécute.

Nous commençons par la spécification de l'application, l'analyse de la cospécification, la description des composants architecturaux et des protocoles de communication, le partitionnement et nous terminons par la cosynthèse d'interface.

9.2 Présentation du jeu de test

9.2.1 Cospécification de l'application

La spécification consiste à décrire les fonctionnalités du système à concevoir et les contraintes qu'il doit respecter. Notre approche consiste premièrement à entrer le code de la cospécification Java du système, qui est supposée correcte d'un point de vue syntaxique, lexical, et sémantique. Cette partie est réalisée grâce un éditeur texte (qui permet d'entrer le code de cospécification voir figure 9.1).

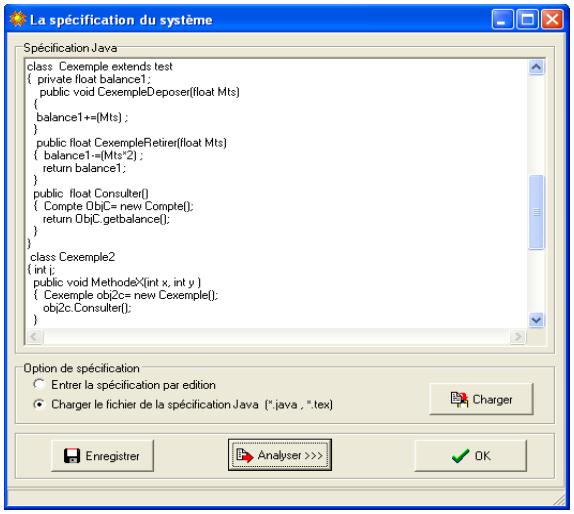

Figure 9.1: Editeur permet d'entrer la cospécification Java de l'application

Prenons un exemple de cospécification Java (figure 9.2)

```
class Compte
 {
  private float balance ;
  Cexemple obj1;
  Cexemple obj2;
  float Val;
  int indice ;

  public void deposer(float montant)
  {
   obj1 = new Cexemple();
   obj1.CexempleDeposer(montant,indice);
   obj2 = new Cexemple();
   obj2.MethodeXY(Val);
   }
 }
class Cexemple
{
  private float balance1;
  public void CexempleDeposer(float Mts, int indice)
  {
  balance1+=Mts ;
  }
  public void MethodeXY(float Qte )
  {
  balance1-=(Qte) ;
  }
}
  public class test
  {
  public static void main (String []args)
  {
  Compte ObjMain= new Compte();
  }
}
```

Figure 9.2: Exemple de cospécification Java d'un système

L'exemple illustré dans la figure 9.2 comporte trois instances d'objets (*ObjMain*, *obj1* et *obj2*). L'objet *ObjMain* qui appartient à la classe Compte, appel les deux objets *obj1* et *obj2* de la classe *Cexemple* à travers la méthode *deposer()*. Il envoie pour le premier objet les deux paramètres (montant, indice) et pour le deuxième objet le paramètres Val.

9.2.2 Analyse du code de la cospécification

Une phase d'analyse de code de cospécification est effectuée (elle est décrite en chapitre 6). Cette phase permet de lire toutes les unités du code source afin d'extraire un certain nombre

d'information sur les classes, les objets et les appels de méthodes. Elle a pour buts d'identifier toutes les classes de l'application, d'identifier toutes les instances d'objets déclarés dans le programme (objets membres d'une classe ou d'une méthode, objets globaux, etc), d'identifier toutes les méthodes membres, et enfin d'identifier tous les appels aux méthodes. Le résultat de cette phase et la représentation des informations extraites à partir du code source sous forme deux graphes (graphe de d'objet et graphe de classe). La figure 9.3 présente le graphe d'instances d'objets (ou de communication) correspondant à l'exemple précèdent (figure 9.2) réalisé par l'environnement d'analyse développé.

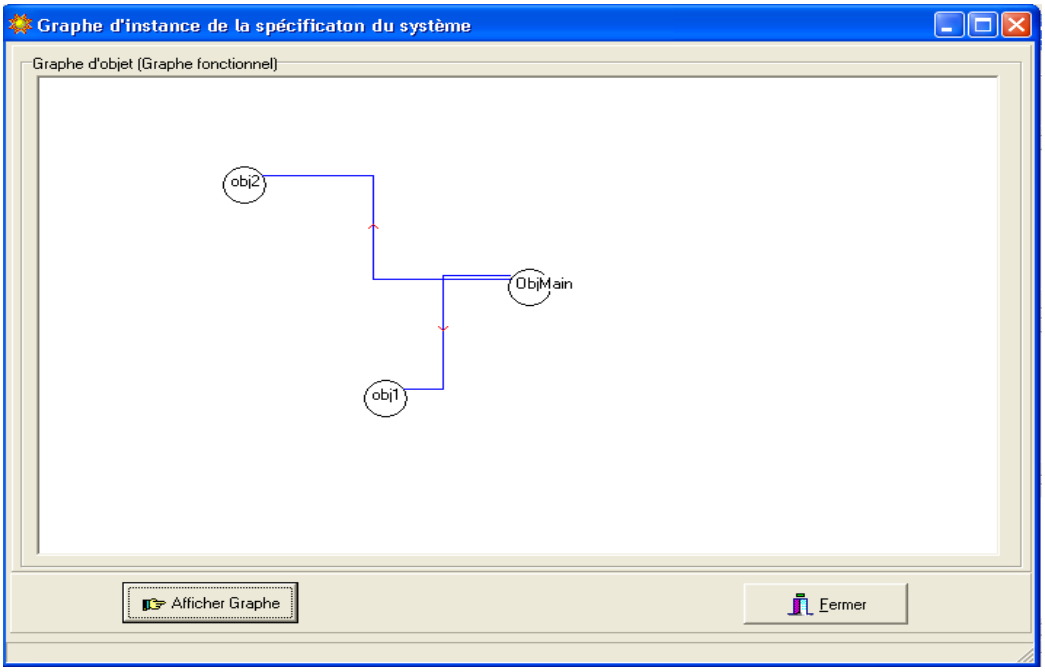

Figure 9.3: Graphe d'instance de l'exemple de cospécification précédent (figure 9.2)

9.2.3 Description de l'architecture cible

Le but de la description de l'architecture cible est de décrire, pour un système donné, le nombre de composants architecturaux, le type de chaque composant, le type des composants de communication et le choix des protocoles de communication.

Dans notre approche, la composition de l'architecture cible permet au concepteur de choisir les composants processeurs, la manière de les relier, et la politique de communication entre eux (protocole de communication).

Pour mettre en œuvre l'exemple précédent de cospécification, nous avons composé une architecture simple grâce à l'outil de description de l'architecture cible développé en chapitre 7.

La figure 9.4 illustre un exemple d'architecture simple qui sert à exécuter l'application du système. Elle est composée de deux composants processeurs: le premier est un processeur à usage général (Intel) qui sert à exécuter la partie logicielle; le second est un processeur matériel ASIC qui implémente la partie matérielle. Ces processeurs sont reliés par un composant de communication contrôleur FIFO.

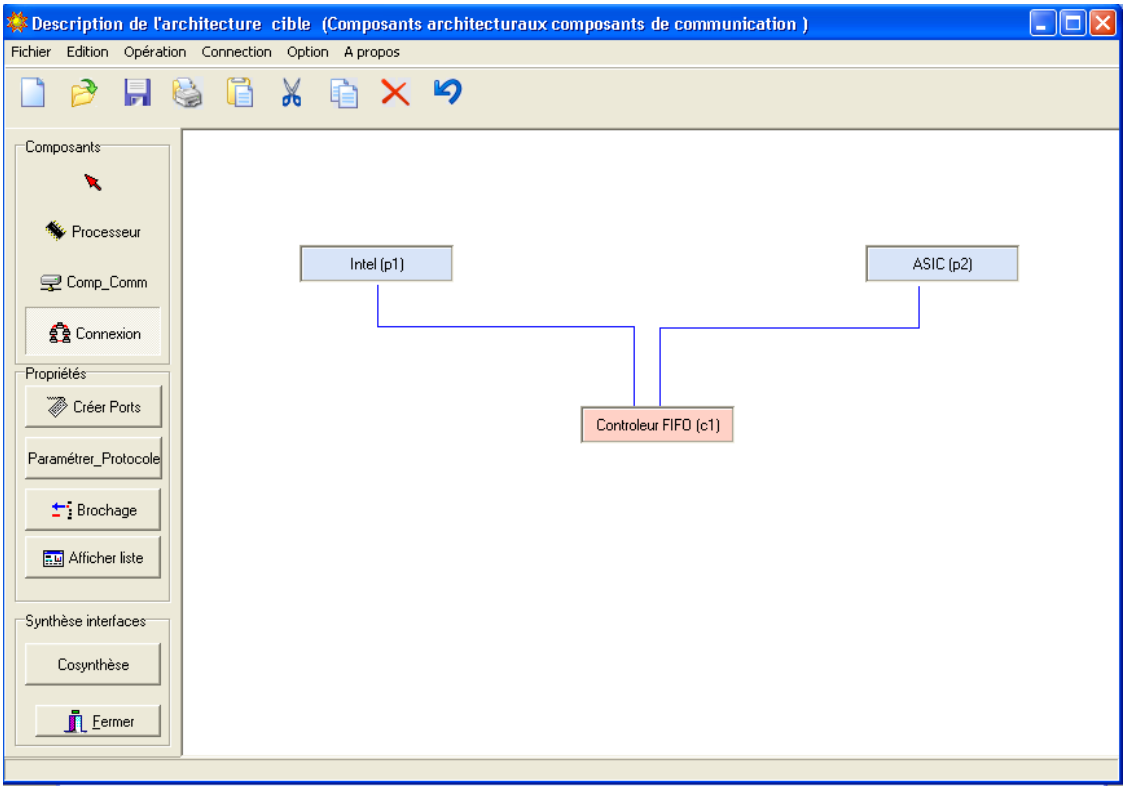

Figure 9.4: Exemple de composition d'une architecture cible

9.2.4 Choix d'un protocole de communication

La figure 9.5 illustre le protocole implémenté dans le composant de communication *Contrôleur FIFO (c1)* de l'architecture cible précédente (figure 9.4).

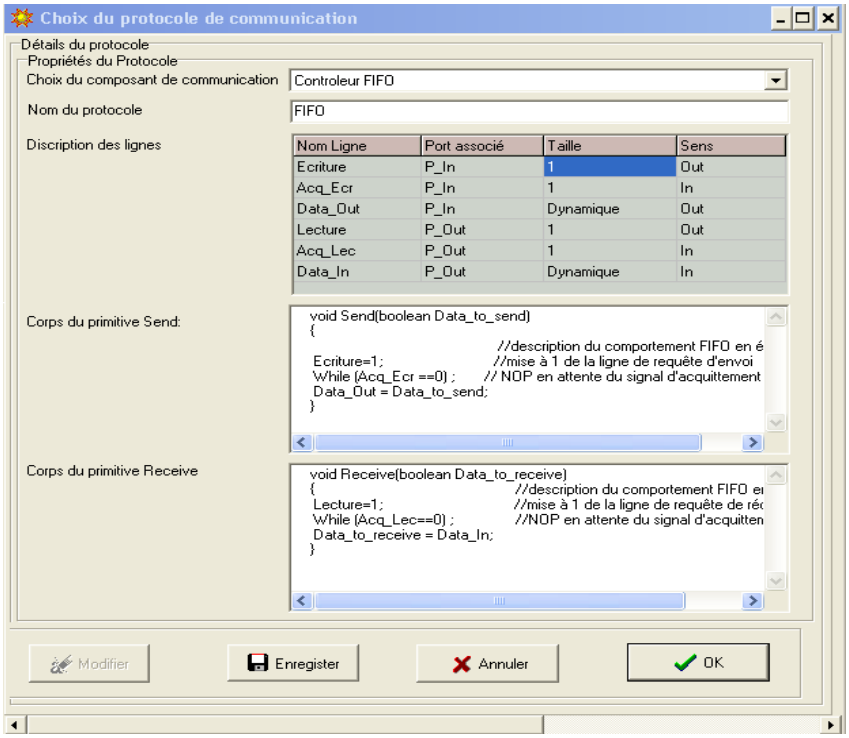

Figure 9.5: Choix d'un protocole de communication dans une bibliothèque

Le composant de communication Contrôleur FIFO intègre un protocole de communication FIFO. Ce dernier définit les règles de la politique d'échange d'informations entre les ports des deux processeurs reliés. Dans notre approche, un protocole est identifié par son nom, le composant de communication qui l'implémente, la description de ses lignes virtuelles, et la description de ses deux primitives de communication *(Send/Receive)* qui normalisent la communication entre les composants processeurs.

9.2.5 Superposition des protocoles sur les composants processeurs de l'architecture cible

C'est la première étape du processus de cosynthèse d'interface de communication. Elle permet d'adapter chaque protocole de communication au composant de l'architecture cible qu'il est chargé de relier. Lorsque l'architecture cible du système est spécifiée, on doit paramétrer et superposer les ports et les protocoles virtuels sur les composants architecturaux.

Pour l'exemple de notre jeu de test, il faut paramétrer le protocole de communication FIFO implémenté dans le composant *contrôleur FIFO (c1)* sur les ports physiques des composants

processeurs de l'architecture (*Intel(p1), ASIC(p2)*). La figure 9.6 illustre les quatre opérations effectuées dans la phase de paramétrage des protocoles de communication qui sont: la sélection des nœuds de communication (processeur/composant de communication/ processeur), la mise en correspondance entre les lignes des ports virtuels des protocoles et les lignes des ports physiques (voir rectangle 1 figure 9.6), le remplacement des ports virtuels par les ports physiques dans les corps des primitives *Send/Receive* (voir rectangle 2 figure 6).

Cette étape permet au concepteur du système de sélectionner ses ports de brochage, de renommer ses lignes, d'établir la correspondance, etc.

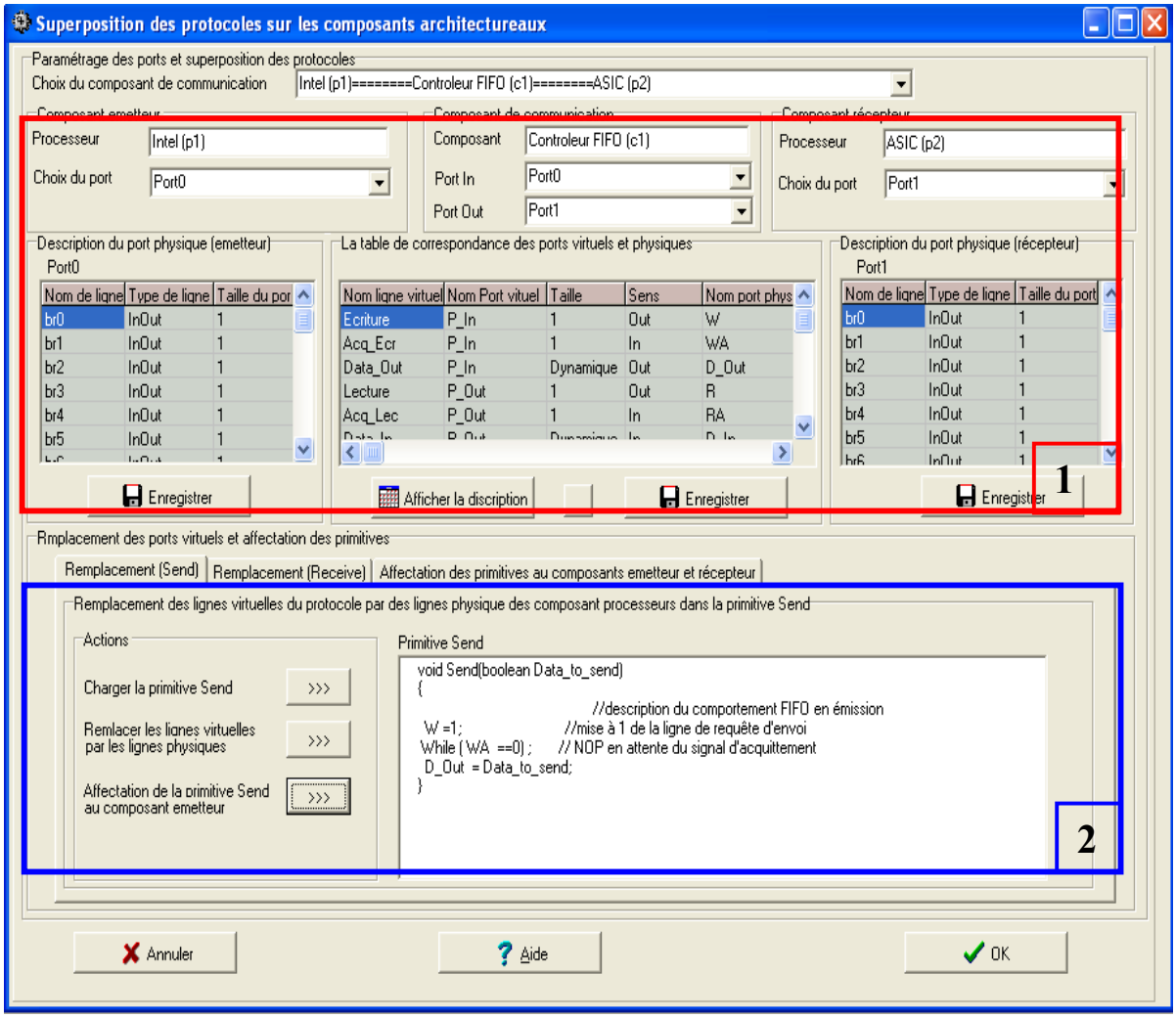

Figure 9.6: Paramétrage et superposition des protocoles sur les composants architecturaux.

9.2.6 Création des ports virtuels et insertion des appels aux primitives virtuelles

La création des ports virtuels et l'insertion des appels aux primitives virtuelles sont deux étapes qui sont effectuées après l'analyse du code de la cospécification. Elles permettent d'extraire toutes les informations pertinentes concernant les opérations classiques de communication afin de remplir les champs des deux tables (table des ports et des primitives).

Comme nous l'avons cité en chapitre 8, le principe proposé pour la communication consiste à créer, pour chaque méthode, une copie et de procéder à des modification afin d'insérer les appels des primitives.

Nous présentons en figure 9.7 une partie de l'application qui réalise la création des ports virtuels et leur insertion dans la table des ports. Elle réalise aussi l'insertion des primitives virtuelles dans le code de cospécification et dans les tables.

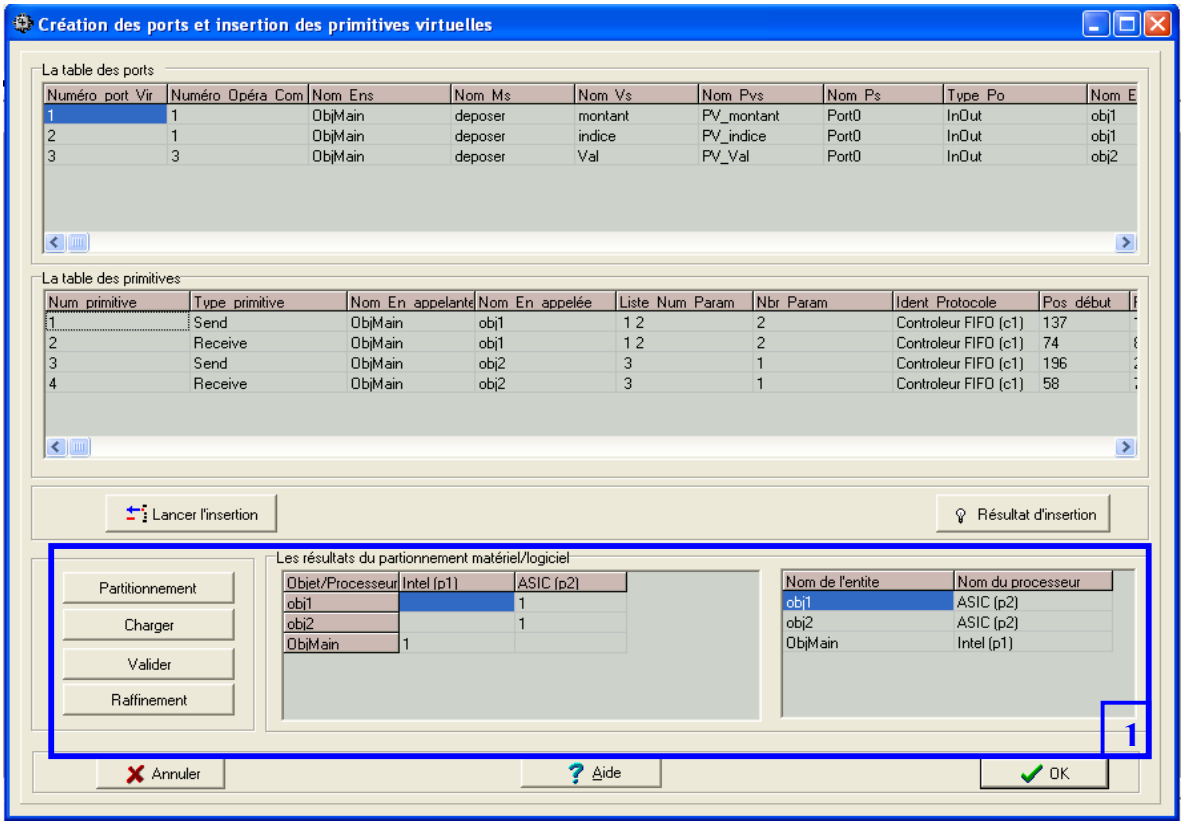

Figure 9.7: Exemple de création des ports virtuels et insertion des primitives de l'exemple précédent (figure 9.2)

La figure 9.7 illustre les différentes informations extraites de tous les appels des méthodes. Certains champs ne sont pas connus à cette étape. Ils sont connus dynamiquement (à l'issue de

l'étape du partitionnement). Ces champs sont: le nom des processeurs source et destination, le nom des ports physiques source et destination, la taille des ports physiques source et destination, et l'identification du protocole de communication entre les processeurs.

Le résultat de cette étape est représenté en figure 9.8. Il comporte le code de la cospécification du système après l'insertion des copies des méthodes appelantes et appelées. Les appels aux méthodes classiques pour la communication sont remplacés par des appels aux primitives standard qui normalisent la communication entre les différents objets de l'application.

```
class Compte
 {
  //declaration
  public void deposer(float montant)
  {
   obj1 = new Cexemple();
   obj1.CexempleDeposer(montant,indice);
   obj2 = new Cexemple();
   obj2.MethodeXY(Val);
  }
  public void deposer_V1(float montant) //copie de la méthode
    {
     obj1 = new Cexemple();
     //obj1.CexempleDeposer(montant,indice);
     Send(1,1,2);
     obj2 = new Cexemple();
     // obj2.MethodeXY(Val);
     Send(3,3);
    }

}
 class Cexemple
{
  private float balance1;
  public void CexempleDeposer(float Mts, int indice)
 {
 balance1+=Mts ;
}
  public void CexempleDeposer_V1() //copie de la méthode
    {
     float Mts; int indice; // déclaration local
     Receive(2,1,2);
     balance1+= Mts ;
    }


   public void MethodeXY(float Qte )
 {
 balance1-=(Qte) ;
 public void MethodeXY_V1()  //copie de la méthode
 {
  float Qte ;     // déclaration local
  Receive(4,3);
  balance1-=(Qte) ;
 }
```

Figure 9.8: Résultat de l'insertion des primitives de l'exemple de la figure 9.2

9.2.7 Le partitionnement

L'étape de partitionnement constitue le processus qui détermine les objets du système qui doivent être implantés en matériel et ceux qui doivent être en logiciel. Pour notre exemple de jeu de test, l'objectif n'est pas d'effectuer un partitionnement des objets du système. Pour illustrer l'exemple, nous procédons à un partitionnement manuel effectué par l'utilisateur à travers un outil interactif. La figure 9.7 (rectangle 1) illustre la partie de l'outil qui réalise le partitionnement des objets du système. C'est à ce moment que les valeurs des champs des deux tables sont connus (nom des processeurs source et destination, nom des ports physiques source et destination, taille des ports physiques source et destination, et identification du protocole de communication).

Objet/ Processeur	**Intel (p1)**	**Asic (p2)**
Obj1		1
Obj2		1
ObjMain	1	

Tableau 9.1: Partitionnement des objets de l'application.

9.2.8 Affinement des appels de primitives virtuelles dans les entités de l'application

Comme nous l'avons cité précédemment, cette étape permet d'affiner les appels des primitives insérées, en remplaçant ces appels par le corps des primitives du protocole de communication. La correspondance doit être établie entre les ports virtuels des primitives insérées et les ports physiques des primitives des protocoles. Prenons comme exemple la méthode copie (*deposer_V1*) de l'exemple de la figure 9.8.

Pour l'exemple illustré dans la figure 9.9, la correspondance entre les appels des primitives de l'application et les primitives des protocoles est réalisée par l'affectation de la donnée occupée dans le port virtuel associé à la variable *"montant"* dans l'entrée '1' de la table des ports virtuels, et l'affectation de la donnée occupée dans le port virtuel associé à la variable *"indice"* dans l'entrée '2' de la table des ports.

Deux problèmes concernant la correspondance se posent:

- Le premier provient de la différence entre les types de la donnée manipulée dans les deux ports (virtuel et physique).
- Le second problème est la quantité d'information à transférer.

Nous discuterons ces problèmes en détail dans la section suivante.

```
public void deposer_V1(float montant) //copie de la méthode
  {
   obj1 = new Cexemple();
   //obj1.CexempleDeposer(montant,indice);
   Send(1,1,2);

 }
// après affinement
public void deposer_V1(float montant) //copie de la méthode
 {
   obj1 = new Cexemple();
   //obj1.CexempleDeposer(montant,indice);
   //Send(1,1,2);
   //Send(1,1);
   //void Send(boolean Data_to_send)
   // { déclaration des lignes physiques
     W=1;                  //mise à 1 de la ligne de requête d'envoi
    While (WA ==0) ;           // NOP en attente du signal d'acquittement
    D_Out = Data_to_send;
   // }
   //Send(1,2);
   //void Send(boolean Data_to_send)
  // { déclaration des lignes physiques
     W=1;                  //mise à 1 de la ligne de requête d'envoi
    While (WA ==0) ;           // NOP en attente du signal d'acquittement
    D_Out = Data_to_send;
   // }
  }
```

Figure 9.9: Exemple d'affinement des appels aux primitives de l'application (résultat de l'étape d'affinement)

9.3 Traitement de quelques problèmes et cas particuliers

9.3.1 Cas d'une fonction renvoyant un résultat

Ce cas peut se manifester dans l'insertion des appels aux primitives virtuelles

Prenons un exemple: *Une_Var= Un_Obj.MethodeX(Par1, Par2) ;*

Les modifications effectuées pour la copie de la méthode seront comme suit:

Send (NM, NPar1, NPar2);

Receive (NM, NUne_Var);

9.3.2 Cas de transmission de variables complexes

L'étape d'analyse doit identifier correctement le type de la variable transmise. Pour le cas de types de variables complexes, il faut étendre les paramètres transmis en remplaçant le nom de la variable complexe par ses différents champs à la fois dans l'objet appelant et appelé. Il est nécessaire d'insérer les paramètres dans le même ordre dans les deux primitives (*Send* et *Receive*) des entités appelantes et appelées.

- Exemple:

Prenons l'exemple d'une variable à transmettre de type *Point,* elle est composée de deux coordonnés X et Y.

La primitive insérée est la suivante:

Send(NM, NX, NY);

Avec *NM:* numéro de la primitive; N*X,* NY*:* numéros des ports virtuels associés aux paramètres *X* et *Y*.

9.3.3 Problème d'un composant processeur possédant plus d'un composant de communication

La communication entre les différentes parties du système mixte (les objets du système), consiste à recopier la méthode avec ses modifications, et de l'insérer dans le code de la cospécification pour assurer la communication externe (l'échange d'informations entre les objets qui sont situés dans des processeurs différents). Ce mécanisme n'est plus valide quand un processeur possède plus d'un composant de communication.

Prenons l'exemple illustré dans la figure 9.10. Le processeur *Pr1* communique avec le processeur *Pr2* grâce à un composant de communication *Contrôleur_FIFO*, et il communique avec le composant processeur *Pr3* grâce à un composant de communication *Contrôleur_DMAC*.

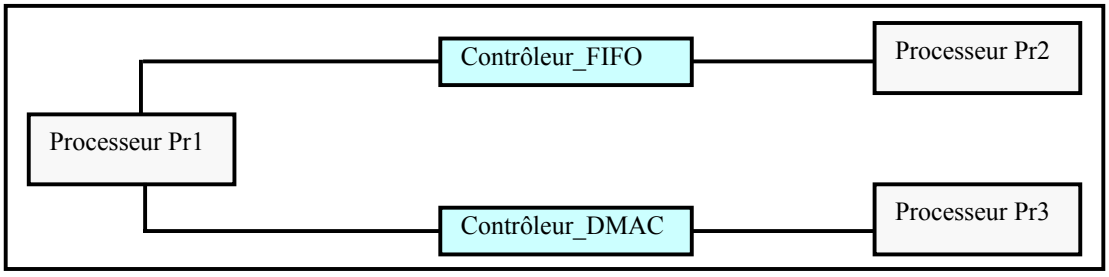

Figure 9.10: Exemple de communication avec deux composants de communication

Pour assurer la communication entre le processeur *Pr1* et les deux processeurs *Pr2*, *Pr3*, il faut que le processeur Pr1 possède autant de copies de la méthode modifiée qu'il y a de composants de communication, c'est-à-dire:

-Une méthode originale qui sert à une communication interne.
-Une méthode copie modifiée et paramétrée pour la communication externe grâce au composant *Contrôleur_FIFO*.
-Une méthode copie modifiée et paramétrée pour la communication externe grâce au composant *Contrôleur_DMAC.*

9.3.4 Problème de correspondance de type entre les ports physiques et les ports virtuels dans le processus d'affinement

Il est utile de rappeler que les ports physiques des composants architecturaux sont des chaînes de bits ayant une taille fixe et mettant en œuvre le canal de communication. D'un autre côté, les objets de l'application possèdent des ports virtuels mettant en œuvre les types plus complexes permis par le langage de cospécification. Le problème qui se pose est que, du fait que nous ayons adopté un langage de haut niveau pour la spécification, les données peuvent être de types très différents. Pour cela, Il est donc nécessaire d'insérer une procédure de conversion (casting) du type de la variable (entier) vers la chaîne binaire matérialisant le port physique. Chaque procédure de conversion a alors pour tâche de convertir le type de variable correspondant en chaîne d'octets (et inversement en réception).

En Java, les procédures de conversion de type sont assez simples. Pour chaque type de données, il existe une procédure. Le polymorphisme permet de garder la même forme pour l'entête de la procédure, quelque soit le type de la variable à transmettre.

9.3.5 Problème de correspondance de taille entre les ports physiques et les ports virtuels dans le processus d'affinement

Ce problème est soulevé par la différence entre la taille du port virtuel et la taille du port physique correspondant (la taille du paramètre à envoyer est différente de la taille du port physique).

Prenons un exemple d'envoi sur un port physique de taille 8 bits, d'un paramètre de type entier (24 bits).

La solution proposée consiste à recourir à une procédure d'adaptation des tailles des paramètres aux ports physiques.

Nous pouvons distinguer deux cas:

- La taille du paramètre à transférer est plus grande que celle du bus

Dans ce cas, il est nécessaire de fractionner la donnée en paquets de taille inférieur ou égale celle du bus et d'envoyer les paquets l'un après l'autre. A la réception des paquets, il est nécessaire d'effectuer l'opération inverse à l'autre bout du canal et de reconstituer la donnée. Les primitives d'envoi et de réception (*Send/Receive*) sont transformées pour prendre en considération la conversion des types et l'envoi par paquets. La figure 9.11 illustre les primitives transformées.

```
Send(Paramètre)
{
 Convert (Param, Chaine, Taille_Port_Physique, Nb_Paquets);
 // convertit la variable "Param" en une chaîne de bits "Chaine " contenant un nombre
//"Nb_Paquets" de mots dont chacun a la taille du port physique

Int Indice=0 ; // variable qui parcourt le nombre d'octets à transférer
While Indice < Nb_Paquets
Send(Chaine [Indice]);
// appel à la primitive du protocole avec une cha îne de bits de taille égale à celle du port physique
}


//*************************************************************************
Receive (Paramètre)
{
Int Indice =0 ; // variable qui parcourt le nombre d'octets transférés
While Indice < Nb_Paquets //le nombre de paquets est le même que pour la primitive send
Receive(Chaine[Indice]); // appel à la primitive du protocole avec une chaîne de bits
// de taille égale à celle du port physique

ConvInv(Chaine, Paramètre, Taille_Port_Physique, Nb_Paquets);
// convertit la chaîne d'octets "Chaine" contenant un nombre "Nb_Paquets" de mots
// dont chacun a la taille du port physique, en la variable Paramètre
}
```

Figure 9.11: Transformation de la primitive "*Send/Receive*" pour le transfert par paquets.

- La taille du paramètre à transférer est plus petite que celle du bus

Il suffit de compléter les bits restants par des zéros, en prenant soin, à la réception, de les éliminer pour ne garder que l'information significative.

9.4 Conclusion

Dans ce chapitre, nous avons présenté les différentes phases de la mise en œuvre de la nouvelle approche de cosynthèse d'interface de communication. Notre jeu de test commence par l'analyse du code de la cospécification qui est effectuée automatiquement grâce à l'environnement d'analyse développé. Elle permet d'extraire toutes les informations sur les objets et la communication entre eux.

La second tâche est la description des composant architecturaux. Elle permet au concepteur de choisir ses composant selon ses besoins.

La dernière étape est la cosynthèse d'interface.

L'automatisation des différentes tâches de ce processus constitue le point essentiel de notre approche. Elle permet au concepteur de considérer un plus grand nombre de solutions et libère ce dernier de la gestion des détails de bas niveaux.

L'avantage principal de notre approche est qu'elle rend l'opération de synthèse d'interfaces transparente au concepteur, le dispensant ainsi de l'une des tâches les plus sujettes à erreurs du processus de codesign du fait de sa complexité. Toutes les étapes d'affinement sont automatiques et restent transparentes pour le développeur d'applications logicielles qui ne désire pas connaître les détails de mise en œuvre.

Durant les étapes de cosynthèse des interfaces, nous avons rencontré quelques problèmes et quelques cas particuliers auxquels nous avons tenté d'apporter des solutions.

L'incompatibilité des primitives du système d'exploitation reste un problème très ouvert. Le problème est que chaque système d'exploitation dispose de primitives différentes et donc les méthodes offrant un même service peuvent avoir des noms différents selon le système d'exploitation référencé.

CONCLUSION

Dans ce travail, nous avons abordé le problème de cosynthèse d'interfaces de communication dans le processus de conception des systèmes mixtes. Notre objectif était le développement d'une approche automatique de haut niveau pour la cosynthèse d'interface, basée sur un formalisme de cospécification et un modèle générique orienté objets des composants. Cette approche permet aux concepteurs des systèmes de synthétiser ses interfaces automatiquement et d'une manière transparente (libérer les concepteurs la gestion des détails de bas niveaux qui sont souvent sources d'erreurs), elle permet aussi une meilleure exploration de l'espace de solutions.

L'approche de cosynthèse développée est basée sur la notion de ports virtuels et de primitives virtuelles de communication (*Send*, *Receive)*. Elle consiste à procéder à des affinements successifs sur le code de cospécification. Il a fallu, pour cela, prendre en charge la quantité importante d'informations hétérogènes manipulées tout au long du processus de cosynthèse. Nous avons tenté de résoudre un certain nombre des problèmes rencontrés durant le processus de cosynthèse d'interfaces. Des solutions sont proposées pour l'analyse du code de la cospécification, la description des composants architecturaux et le déroulement des différentes étapes de cosynthèse d'interfaces.

Formalisme de cospécification et modèle des composants utilisé :

L'étape de cospécification est basée sur l'approche orientée objet. La partie application est décrite en langage Java, et la partie architecture cible est décrite d'une manière graphique à l'aide de composants et de protocoles stockés dans des bibliothèques. La notion d'orienté objet est de plus en plus utilisée dans la spécification de systèmes complexes du fait qu'elle simplifie le développement de logiciels plus gros et plus complexes [62]. Le formalisme de cospécification orienté objet offre plusieurs mécanismes, tel que l'héritage, la composition et le polymorphisme. Ces mécanismes permettent d'offrir une plus grande souplesse de développement qui favorise

l'amélioration et la maintenance du système, réduisant ainsi le temps de conception et les risques d'erreurs.

Le formalisme de cospécification utilisé possède les avantages suivants :

- Indépendance entre la spécification de l'application et la description de l'architecture cible sur laquelle l'application s'exécute.
- Utilisation de mécanismes d'abstraction (hiérarchisation, encapsulation, héritage, agrégation, etc.) qui sont adaptés à la description de systèmes complexes.
- La communication est clairement définie à travers les appels de méthodes classiques grâce à l'approche objet.
- La réutilisabilité des composants reste valable pour les systèmes mixtes en codesign.
- Garantie d'une cospécification de haut niveau, le concepteur n'a ainsi pas à se préoccuper de la manière dont son système sera implanté.
- L'utilisation du Langage Java comme langage de cospécification permet de décrire : le parallélisme à travers l'exécution de processus multiples (threads), la synchronisation entre threads est assurée à travers les sections critiques.

Analyse du code de la cospécification de l'application :

Nous avons développé une technique et un environnement d'analyse du code de la spécification Java de l'application, ce qui constitue notre première contribution dans ce projet. Cet environnement n'est pas un outil d'aide à la programmation, mais plutôt un outil qui permet au concepteur d'introduire le code de la cospécification en Java. Le code est supposé correcte d'un point de vue syntaxique, sémantique et lexical, du fait que Java offre un avantages certain: la disponibilité d'outils fiables pour la mise au point. La cospécification est automatiquement analysée afin d'extraire les différentes informations relatives à la communication entre objets de l'application. Ces informations permettent, dans un premier temps, de construire les deux graphes (graphe fonctionnel et graphes hiérarchique), et dans un deuxième temps d'alimenter le modèle et les étapes ultérieures du processus de conception (étape de modélisation, étape du partitionnement et de synthèse d'interfaces, etc.). Nous avons proposé des structures de données pour l'extraction des informations concernant le code de cospécification.

Nous avons cependant rencontré quelques problèmes et des cas particuliers auxquels nous avons tenté d'apporter des solutions.

Description des composants architecturaux et des protocoles de communication

La description des composants de l'architecture cible dépend fortement de l'analyse du code de la cospécification de l'application, et des différentes informations extraites (le temps d'exécution de chaque entité sur chaque processeur, l'espace occupé, la quantité d'informations échangés durant la communication, etc.).

L'utilisation d'un environnement dédié pour la description de l'architecture cible (ex: outil commercial), rend difficile l'extraction des différentes informations et leur transmission à l'outil de partitionnement et au processus de cosynthèse d'interface. Nous avons développé un outil pour la description des composants de l'architecture cible et les protocoles de communication. Pour cela, nous avons fait le choix d'offrir à l'utilisateur la possibilité de composer ses architectures cibles en même temps que l'application qui s'exécute dessus. L'objectif est de simplifier la circulation de flux d'informations à travers toutes les étapes de processus de codesign. L'outil développé est basé sur des bibliothèques de composants architecturaux et de protocoles de communication réutilisables. Une interface graphique permet aux concepteurs des systèmes mixtes de sélectionner ses composants et ses protocoles de communication et de les paramétrer selon ses besoins d'une manière interactive.

Le protocole de communication est un composant virtuel au même titre que les composants architecturaux. Il est modélisé par un composant encapsulant des données et des traitements. Les traitements sont représentés par des primitives virtuelles (*Send*/*Receive*) agissent sur des ports virtuels. Ces protocoles sont regroupés et stockés dans la bibliothèque. Notre outil de description des composants et des protocoles de communication permet également de décrire de nouveaux protocoles entre les composants architecturaux, de manière très simple, de les stocker dans une bibliothèque, et de les réutiliser en les paramétrant sur de nouveaux composants.

L'avantage de l'utilisation de bibliothèques de composants architecturaux et de protocoles de communication, est qu'elles permettent de rendre l'application indépendante de l'architecture (architecture non figée ou paramétrable). Notre outil permet d'utiliser les composants définis dans la bibliothèque, de créer de nouveaux composants, de les insérer, et de les réutiliser.

Test et mise en œuvre de l'approche de cosynthèse d'interfaces

Notre approche de cosynthèse d'interfaces part d'une spécification Java de l'application de haut niveau, et d'une description des composants architecturaux et des protocoles de

communication. Elle remplace les appels aux méthodes publiques des entités de l'application par des appels à des primitives virtuelles *Send*/*Receive*. Ensuite, elle effectue des affinements successifs sur la spécification Java des objets de l'application.

Nous avons développé un environnement qui permet aux concepteurs d'automatiser les différentes étapes de cosynthèse d'interfaces (superposition des protocoles sur les composants architecturaux, création des ports et insertion des appels aux primitives virtuelles, processus d'affinement et élimination des mécanismes inutiles). Cet environnement est basé sur les informations fournies par le module d'analyse (informations sur les objets, communications, etc.) et les informations fournies par le module de description des composants architecturaux (information sur les composants architecturaux et les protocoles).

Un jeu de test est présenté sur un exemple de cospécification Java d'un système et une architecture cible. Un partitionnement manuel est effectué par l'utilisateur à travers un outil interactif, pour identifier les objets qui doivent être implantés sur un processeur matériel et ceux qui doivent être sur un processeur logiciel. L'approche est ensuite testée sur le jeu de test présenté.

L'avantage principal de cette approche est qu'elle procède d'une manière transparente au concepteur (le concepteur n'a pas besoin de connaître le détail de bas niveau de ces interfaces). L'utilisation de deux procédures standard de transferts des informations (*Send*, *Receive*) pour modéliser tous les échanges d'informations, permet de normaliser la communication et d'éviter d'avoir à traiter toutes les possibilités offertes par Java pour l'échange des données en fonction des différentes combinaisons des architectures cibles. Ces primitives de communication (*Send*, *Receive*) sont paramétrées au fur et à mesure du processus d'affinement de la communication. Un autre avantage de l'utilisation des ports virtuels dans cette approche, est qu'ils permettrent de répondre à la grande diversité des objets ou des variables (le type des variables, la taille des variables..). Elle permet aussi d'éviter de redéfinir ou de réécrire les mêmes procédures *Send*/*Receive* pour tous les types de données.

Perspectives

Un travail n'est jamais accompli à cent pour cent. Cela implique que de nombreuses perspectives peuvent être suggérées.

L’approche proposée ne résout pas tous les problèmes posés durant le processus de cosynthèse. Nos perspectives portent sur la poursuite de ce travail afin de permettre de:

- L’évolutivité de l’approche pour tenir compte des primitives du système d’exploitation. De nombreux concepteurs ont recours à des primitives de systèmes d'exploitation, y compris les développeurs temps-réels qui utilisent des noyaux temps-réel (RTOS : Real-Time Operating Système) pour les systèmes embarqués. Un problème d’incompatibilité des primitives est posé du fait que chaque système dispose de primitives différentes, et donc des fonctions offrant un même service peuvent avoir des noms différents.
- Développer un outil de traduction du code de la cospécification Java en VHDL. Cet outil permettra d’aider les concepteurs à synthétiser les portions du code Java en circuits dédiés (ASIC, FPGA…).
- Utiliser les mécanismes d’interruptions: le recours au système d’interruption et à la procédure de traitement associée peut s'avérer très important et très intéressant: par exemple, lorsque deux entités doivent communiquer de manière asynchrone. Le signal d'interruption peut servir à déclencher une méthode de l'entité appelée, ou à exécuter une portion de code.

REFERENCES

1. M. Auguin, "Co-Conception de Systèmes Spécialisés sur Composant", Les Algorithmes / Bâtiment Euclide B, 2000 route des Lucioles, BP 121, 06903 Sophia-Antipolis, France, 2000.
2. Auguin M., Capella L., Cuesta F., Gresset E., "CODEF: a system level exploration tool", IEEE International Conference on coustics, Speech, and Signal Processing, Salt Lake City, p. 1031-1034, May 2001.
3. Ivan Augé, Rajesh K. Bawa, Pierre Guerrier, Alain Greiner, Ludovic Jacomme, and Frédéric Pétrot, "User guided high level synthesis", In Ricardo Reis and Luc Claensen, editors, VLSI: Integrated Systems on Silicon, pages 464–475, Gramado, Brazil, IFIP, Chapman & Hall, August 1997.
4. M. Azizi, "Covérification des Systèmes Intégrés", Thèse de Doctorat, Faculté des Études Supérieures, Université de Montréal, Décembre 2000.
5. F. Balarin, H. Hsieh, A. Jurecska, L. Lavagno and A. Sangiovanni-Vincentelli, "Formal Verification of Embedded Systems based on CFSM Networks", in Proc. DAC, pp. 568-571, 1996.
6. F. Balarin et al., "Hardware-Software Codesign of Embedded Systems", The POLIS Approach, Kluwer Academic Publishers, 1997.
7. T. Ben Ismail, M. Abid, K. O'Brien, A. Jerraya, "An Approach for Hardware-Software Codesign", RSP'94, Grenoble, France, June 1994.
8. T. Ben Ismail, "Synthèse au niveau système et conception de systèmes mixtes logiciels/matériels", Thèse Doctorat Institut National Polytechnique De Grenoble 1996
9. G. Bois, "Spécification et conception des systèmes embarqués", Support de cours INF6501, guy. bois@polymtl. ca http: //www. cours. polymtl. ca/inf6501 HIVER 2005.
10. G. Boriello, K. Buchenrieder, R. Camposano, E. Lee, R. Waxman and W. Wolf, "Hardware/Software Codesign", IEEE Design & Test of Computers, Round Table, pp. 83-91, March 1993.
11. M. Boubekeur, "Validation de Spécifications de Circuits Asynchrones: Méthodes et Outils", Thèse de Doctorat, TIMA, Grenoble, Octobre 2004.
12. J. Buck, J. Ha, E. A. Lee and D. G. Messerschmitt, "Ptolemy: a framework for simulating and prototyping heterogeneous systems", International Journal of Computer Simulation, Vol. 4, pp. 155-182, 1991.

13. J. Buck, J. Ha, E. A. Lee and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems", Special issue on simulation software development, Int. J. Computer Simul., Vol. 4, No155, pp. 155-182, Avril 1994. http://ptolemy. eecs.berkeley.edu

14. J. P. Calvez, "spécification et conception des système", Editions Masson, 1990.

15. R. Camposano and J. Wilberg, "Embedded System Design", Design Automation for Embedded Systems, Vol.1, N°1-2, pp.5 -50, January 1996.

16. M. Chiodo, P. Giusto, A. Jurecska, C. Hsieh, A. Sangiovanni-Vincentelli and L. Lavagno, "Hardware-Software codesign of embedded systems", IEEE Micro, pp. 26-36, Aug. 1994.

17. P. Chou, R. B. Ortega, G. Borriello, "Interface co-synthesis techniques for embedded systems", International Conference on Computer Aided Design Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design.

18. P. Chou, R. Ortega, K. Hines, K. Partridge and G. Borriello, "IPCHINOOK: An integrated IP-based design framework for distributed embedded systems", In Proceedings of the 36th Design Automation Conference, pages 44–49, 1999.

19. P. Coste, F. Hessel, P. Le Marrec, Z. Sugar, M. Romdhani, R. Suescun, N. Zergainoh and A.A. Jerraya, "Multilanguage design of heterogeneous systems", CODES'99, pp. 54-58, 1999.

20. J. M. Daveau, G. F. Marchioro, T. Ben Ismail and A. Jerraya, "Protocol Selection and Interface Generation for HW-SW Codesign", IEEE Transaction on Very Large Scale Integration, Vol. 5, N°1, pp. 136-144, 1997.

21. J. M. Daveau, "Spécifications systèmes et Syntèse de la communication pour le codesign logiciel-Matériel", Thèse de Doctorat, Institut Nationale Polytechnique de Grenoble, spécialité microélectronique, mars 1997.

22. G. DeMicheli and R. Gupta, "Hardware/Software Co-design", Proceedings of the IEEE, Vol. 85, N°3, pp. 349-365, 1997.

23. A. Diagne, "Systèmes répartis et coopératifs: une approche multi-formalismes de spécification de systèmes répartis: transformation de composants modulaires en réseaux de petri", Thèse de Doctorat, Université Paris 6, 1997.

24. M. D. Edwards, J. Forrest and A.E. Whelan, "Acceleration of software algorithms using hardware/software co-design techniques", Journal of Systems Architecture, Vol. 42, N°9, Feb. 1997.

25. ERNST R. AND BENNER T., "Communication, constraints and user directives in COSYMA", technical Rreport CY-94-2, Institüt für DV-Anlagen, Technische Universit?t Braunschweig, June 1994.

26. D. Gajski, F. Vahid, and S. Narayan, "A System-Design Methodology: Executable-Specification Refinement", Proc. European Design & Test Conference (EDAC-ETC-EuroASIC), Paris, France, IEEE CS Press, pp. 458-463,

February 1994.

27. D. Gajski, and F. Vahid, "Specification and Design of Embedded Systems, " IEEE Design & Test of Computers, pp. 53-67, Spring 1995.

28. D. Gajski, F. Vahid, S. Narayan and J. Gong "SpecSyn: An Environment Supporting the Specify-Explore-Refine Paradigm for Hardware/Software System Design", 84th IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 6, N° 1, March 1998.

29. L. Gauthier, "Génération de système d'exploitation pour le ciblage de logiciel multitâche sur des architectures multiprocesseurs hétérogènes dans le cadre des systèmes embarqués spécifiques", Thèse de Doctorat, Laboratoire TIMA, Ecole Doctorale EEATS, 2001.

30. A. Grasset, F. Rousseau, A. A. Jerraya, "Génération des Interfaces de Communication pour les Systèmes Multiprocesseurs Monopuces: de la Spécification des Services de Communication vers l'Implémentation RTL", TIMA, Grenoble, 2002

31. K. Ten Hagen, H. Meyer, "Timed and Untimed Hardware/Software Cosimulation: Application and Efficient Implementation", International Workshop on Hardware-Software Codesign, Cambridge, October 1993.

32. J. L. Hennessy, and D. A. Patterson, "Organisation et conception des ordinateurs: L'interface Matériel/Logiciel, " Édition française par P. Klein, DUNOD Publishers, Paris, 1994, 660 pages

33. J. Henkel and R. Ernst, "High level estimation techniques for usage inhardware/software co-design", 1998, www.ida.ing.tu-bs.de

34. F.P. Hessel, "Conception des systèmes hétérogènes multilangages", Thèse de Doctorat, UNIVERSITE JOSEPH FOURIER, 2000.

35. C.A.R. Hoare, "Communicating Sequential Processes", Communications of the ACM 21, vol. 8, pp. 666-677, Aug 1978.

36. C.A.R. Hoare, Communicating Sequential Processes, 1985, Prentice Hall.

37. D. Hommais, "Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés Matériel-logiciel", Thèse de Doctorat de l'université Paris VI spécialité informatique, Septembre 2001.

38. J.P. Jamont, "Une approche pour la conception de systèmes multi-agents embarqués", Thèse de Doctorat Laboratoire de Conception et d'Intégration des Systèmes, dans le cadre de l'Ecole Doctorale, 29 Septembre 2005.

39. A. Jantsch A., J. Oberg, P. Ellervee, A. Hemmani and H. Tenbunen, "A software oriented approach to hardware-software codesign", Proceedings of the Poster Session of International Conference on Compiler Construction, CC- 94, pp.93-102, Edimburgh, Scotland, 1994.

40. A.A. Jerraya and K. O'Brien, "SOLAR: an intermediate format for system-level modeling and synthesis", Computer Aided Software/Hardware Engineering, Ed. J.

Rozenblit, IEEE Publisher, Chap. 10, 1994.

41. J.M.P. Cardoso and H.C. Neto, "An Approach to Hardware Synthesis from a system Java Specification", IEEE COPYRIGHT In Proceedings of the 5th IEEE International Conference on Electronics, Circuits and Systems, Lisbon, Portugal, September 7-10, 1998.

42. M.J.C. Hamon, "Méthodes et outils de la conception amont pour les systèmes et les microsystèmes", Thèse de Doctorat de l'Institut National Polytechnique de Toulouse, 1 février 2005.

43. A. Kalavade and E.A. Lee, "A hardware-software codesign methodology for DSP applications", IEEE Design and Test of Computers, Vol. 10, N°3, pp. 16-28, Sept. 1993.

44. A. Kalavade, "System-Level Codesign ofMixedHardware-Software Systems", PhD Thesis, Electronics Research Laboratory, Berkeley, 1995.

45. M.J. Knieser and C.A. Papachristou, "COMET: a hardware-software codesign methodology", Proceedings of the European Design Automation Conference with Euro-VHDL, pp. 178-183, September 1996.

46. M. Koudil, "Une approche orientée objet pour le Codesign", Thèse de Doctorat d'Etat, Institut National d'Informatique, 2002.

47. M. Koudil, Support de cours de magister, INI, 2004.

48. E.A. Lee, "Specification and Design of Reactive SystemsOverview of the Prolemy Project", A dissertation submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences, Graduate Division of the University of California, Berkeley, 2000.

49. E.A. Lee, "Overview of the Ptolemy Project, " Technical Memorandum UCB/ERL M03/25, July 2, 2003, University of California, Berkeley, CA, 94720, USA.

50. E.A. Lee and S. Neuendorffer, "Classes and Subclasses in Actor-Oriented Design", Invited paper: Conférence on Formal Méthods and Models For Codesign (MEMOCODE) EECS Department University of California at Berkeley Berkeley, CA 94720, U. S. A. June 22-25, 2004

51. E.M. Lester, Rapport d'activité, UPRES EA n° 3372 - CNRS FRE 2734 au 1° janvier 2004 LESTER- Université de Bretagne Sud Centre de recherche, Lorient, France, 2004.

52. B. Lin and S. Vercauteren, "Synthesis of Concurrent System Interface Modules with automatic Protocol Conversion Generation", In: Proc. Of ICCAD, pp. 395-399, 1994.

53. MINT GROUP PROJECT UPC, "Architecture Description Languages", 2000, http://mint.cs.man.ac.uk/Projects/UPC/Reviews/ArchitecturalDescriptionLanguages.htm

54. V.J. Mooney, C. N. Coelho , T. Sakamoto and G. DeMicheli, "Synthesis From Mixed Specifications", Proceedings of the European Design Automation Conference with Euro-VHDL, pp. 114-119, September 1996.

55. S. Narayan and D. Gajski, "Synthesis of System-Level Bus Interfaces", In: Proc. of EDAC with Euro-VHDL, pp. 395-399, February 1994.

56. S. Narayan and D. Gajski, "Interfacing Incompatible Protocols Using Interface Process Generation", In: Proc. of the IEEE Design AutomationConference, pp. 157-164, 1995.

57. E.G.N. Nicolescu, "Spécification et validation des systèmes hétérogènes embarqués", Thèse de Doctorat, Laboratoire TIMA dans le cadre de l'Ecole Doctorale, 2002.

58. M. O'nils, "Specification, synthesis and validation of hardware/software interfaces", Technical Doctor Thesis, School of Electrical Engineering, Royal Institute of Technology, Sweden, 1999.

59. R. Ortega and K. Boriello, "Communication synthesis for distributed embedded systems", Proceedings of the International Conference on Computer Aided Design, 1998.

60. Y. Paviot, "Partitionnement des services de communication en vue de la génération automatique des interfaces logicielles matérielles", Thèse de Doctorat, laboratoire TIMA dans le cadre de l'école doctorale, 2004.

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

62. D.B. Powell,"Real-Time Object Orientation", I-Logix, 1997, http://www.ilogix.com

63. PTOLEMY Project, http: //www.liflfr/west/courses/SoCdesign/biblio0405/ptolemy/rapporok. htm.

64. R.K. Gupta and G. DeMicheli, "A co-synthesis approach to embedded systemdesign automation", Design Automation for Embedded Systems, 1(1-2):69–120, June 1996.

65. E. Stoy and Z. Peng, "A design representation for hardware/software cosynthesis", Proceedings of the 20th EUROMICRO Conference,. EUROMICRO 94. System Architecture and Integration, pp. 192-199, 1994.

66. E. Stoy and Z. Peng, "An integrated modelling technique for hardware/software systems", IEEE International Symposium on Circuits and Systems, Vol. 1, pp. 399-402, 1994.

67. E. Stoy, "A petri net based unified representation for hw/sw codesign", Masters thesis, Dept. of Computer and Information Science, Linköping University, Sweden, S-581 83, 1995.

68. SYNOPSYS Eagle, available on line at

http://www.synopsys.com/products/hwsw/eagle_ds.html. 2002

69. C.A. Valderrama, "Prototype virtuel pour la génération des architectures mixtes logicielles/matérielles", Thèse de Doctorat, Laboratoire TIMA, Grenoble, 1998.

70. D. Verkest, K. Van Romaey, I. Bolsen and H. De Man, "CoWare - A design environment for heterogeneous hardware-software systems", Design Automat. Embedded Syst., Vol.1, N°4, pp.357-386, Oct.1996.

71. J. Wilson, "Hardware/software selected cycle solution", Proceedings of the Third International Workshop on Hardware/Software Codesign, pp.190-194, 1994.

72. X. Xiong, P. Gutberlet and W. Rosentiel, "Automatic Generation of Interprocess Communication in the PARAGON System", Proceedings of IEEE, 1996

73. T. Yen and W. Wolf, "Communication Synthesis for Distributed Embedded Systems", In: Proc. of ICCAD, pp. 288-294, 1995.

74. J. Young,"JavaTime",august,1996, http://www-cad.eecs.berkeley.edu/~jimy/java/index.html

75. J. Young, J. Macdonald, M. Shilman, A. Tabbara, P. Hilfinger and A.R. Newton, "Design and specification of embedded systems in Java using successive formal refinement", 35th Design Automation onference, 1998.

76. L. Zaffalon and P. Breguet, "CONCEPTION DE SYSTEMES REACTIFS", Haute Ecole Spécialisée de Suisse Occidentale Visions / Revue scientifique de l'EIVD / 2001.

77. NILSEN K., "Isues in the design and implementation of Real-Time Java", July 1996, http://www.newmonics.com/webroot/technologies/java/RTJL.ps.

78. Edward Lee and Stephen Neuendorffer Classes and Subclasses in Actor-Oriented Design. Invited paper: Conference on Formal Methods and Models for Codesign (MEMOCODE), EECS Department, University of California at Berkeley, CA 94720, U.S.A. June 22-25, 2004