Une approche d'analyse du code de la cospécificaion et la description des composants architecturaux pour la conception des systèmes mixtes

Fouaz BERRHAIL, Abdelhafid BENOUDA, Mohamed MOSTEFAI
Laboratoire d'automatique et d'informatique industrielle
Université FERHAT ABBAS
Sétif, ALGERIE
bfouaz02@hotmail.com, ahbenaouda@gmail.com, Mostefai@univ-setif.dz

Résumé-- **Nous présentons dans ce travail une approche automatique de haut niveau pour l'analyse de code de la cospécification java, et pour la description des composants qui composent l'architecture cible sur laquelle l'application s'exécute dessus. Dans ce contexte, nous avons développé deux outils pour extraire toutes les informations pertinentes concernant les entités matérielles et logicielles. Ces informations sont utilisées pour alimenter les étapes suivantes du processus de conception des systèmes mixtes tels que le partitionnement et la cosynthèse des interfaces de communication.**

Mots-clés: Codesign, Cospecification Analysis, Architecture cible.

I. INTRODUCTION

Le processus de conception des systèmes mixtes (codesign) procède par étapes. Il commence par la spécification et la modélisation du comportement et des fonctionnalités du système, une étape d'analyse et de description de l'architecture cible. Ensuite, une étape de partitionnement sert à partitionner les entités du système aux différents processeurs de l'architecture cible, suivie par une étape de synthèse de : la partie matérielle, la partie logicielle, et les interfaces de communication entre ces deux partie. La dernière étape du processus est la validation pour garantir que le système conçu réalise de façon correcte les fonctionnalités attendues tout en respectant les contraintes fixées.

L'étape de cospécification est l'étape clé du processus de conception [5]. C'est une représentation abstraite du système que l'on veut concevoir.

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) [7]. 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.

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 dessus. 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, et la quantité d'informations échangées durant la communication, etc.).

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.
- 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.

II. ETAT DE L'ART

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 [11], car elle permet d'extraire un certain nombre de paramètres, qui ont aidés les concepteurs lors des étapes ultérieurs du processus de conception.

L'architecture cible choisie pour supporter et exécuter le système mixte est d'une importance primordiale pour les performances futures de l'application une fois implantée [5].

Les architectures cible utilisées dans les travaux de codesign appartiennent, en général, à l'un des types suivants : composant unique (VLSI[1]), carte dédiée, carte autonome embarquée [7]. Une classification des architectures présentées dans la littérature, permet de distinguer deux approches :

La première classe englobe les travaux qui fixent l'architecture cible avant de procéder au codesign de l'application qui doit être exécutée dessus. L'architecture la plus utilisée consiste en un processeur à jeu d'instructions et un ou plusieurs circuits matériels (de type ASIC[2], FPGA[3] ou autre) pour implanter les fonctions nécessitant plus de rapidité de calcul [1, 2, 3, 4,12].

La seconde classe d'approches consiste à déduire l'architecture cible qui résulte de l'étape de partitionnement [6, 8, 9].

Dans la Méthodologie SpecSyn [6,10]. L'architecture cible est une machine mono-processeur (les processeurs à mémoire

[1] VLSI *:* Very-large-scale integration.
[2] ASIC: Application-Specific Integrated Circuit.
[3] FPGA: Field Programmable Gate Array

cache et/ou avec instructions pipelinées ne sont pas pris en considération).

III. SPECIFICATION DU SYSTEME MIXTE

L'étape de spécification de l'application constitue l'étape clé du processus de conception. Nous avons fait le choix sur un formalisme de cospécification orienté objet. Notre cospécification est effectuée à l'aide du langage orienté objet Java, ce dernier permet de décrire toutes les entités de l'application à concevoir.

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 [7]. 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), la figure 1 présente un modèle générique utilisé.

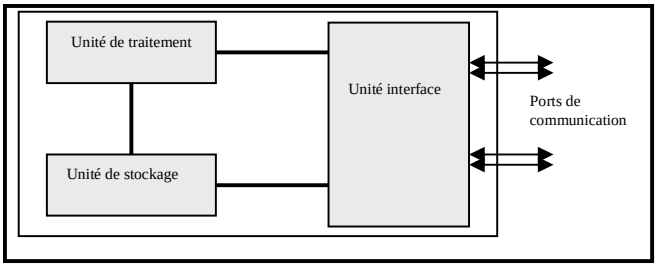


Figure 1: Modèle générique d'un composant [7].

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. Notre formalisme de cospécification 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, tels 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.

IV. ANALYSE DU CODE DE LA COSPECIFICATION

Nous avons développé 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 qui représente la hiérarchie des classes de la cospécification; et graphe d'instance qui représente les liens de communication entre objets. Les informations à extraire de la spécification Java sont des informations sur les classes, les méthodes, les appels aux méthodes et les objets.

A. Différentes étapes de l'analyse

L'analyse de la cospécification Java se déroule en deux étapes, comme illustre la figure 2. Le module d'analyse reçoit en entrée 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.

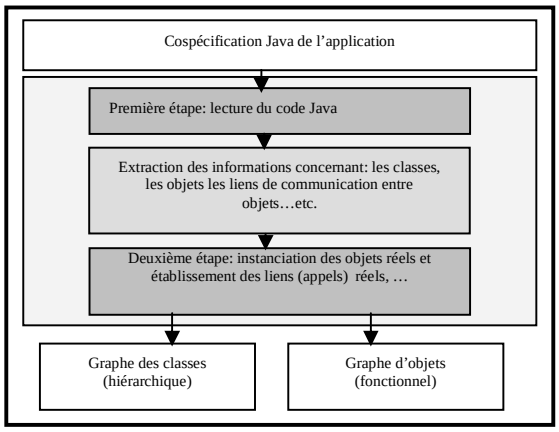


Figure 2: Etapes de l'analyse de la cospécification Java

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. Dans notre approche on s'intéresse à l'aspect communication entre objets (extraire des informations relatives à la communication entre objets)

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 leurs noms, 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).

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.

B. 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 (fonctionnel).

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

2) Graphe d'instances d'objets (fonctionnel)

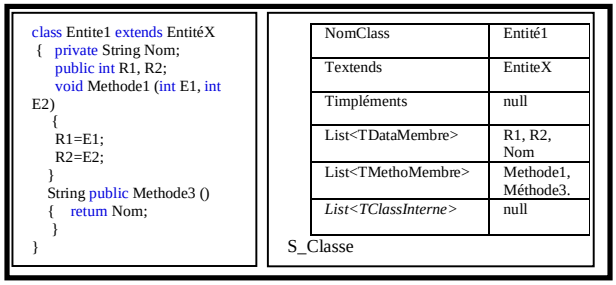
Ce graphe représente toutes les instances d'objets qui sont les instances des classes du graphe hiérarchique. Les nœuds de ce graphe représentent les instances d'objets de l'application et les relations entre objets sont représentées par les arcs du graphe. Ce graphe permet de déterminer les quantités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

V. 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.

A. Détection des classes et de leurs caractéristiques

La première phase de l'analyse consiste à extraire toutes les informations sur les classes. La structure de données *S_Classe* proposée encapsule toutes les informations sur une classe de l'application: le nom, les classes ascendantes, les interfaces implémentées, les objets et les méthodes membres, etc. La figure 3 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

S_Classe

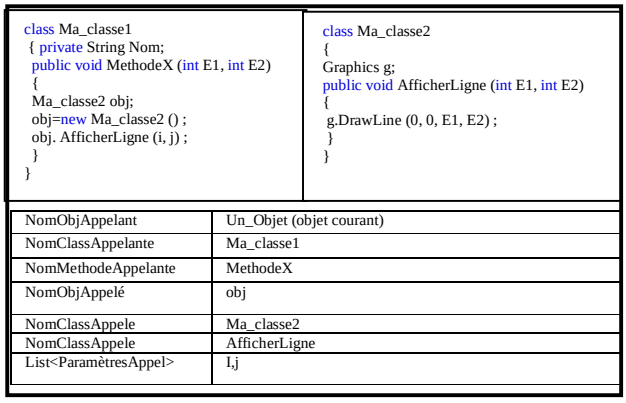
Figure 3: Exemple de détection d'une classe et la structure de données correspondante.

B. 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.

C. 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 appelée, 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 4 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) ;
 }
}
```

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

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

Figure 4**:** Exemple de détection des appels de méthodes et les structures de données proposées.

D. Problèmes rencontrés et cas particuliers

Durant l'analyse de la spécification du système, certains cas et problèmes particuliers peuvent se présenter. Pour cela nous avons tenté de proposer des solutions Dans ce qui suit, nous en présentons un exemple de problème de la cohérence de la mémoire.

-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, On suppose que la classe A est affectée au processeur Pr_1, et la classe B est affecté au processeur Pr_2.

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.

La solution que nous proposons consiste à insérer un mécanisme 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 table 1. La table 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.

Table 1: Table de mécanisme de cohérence de la mémoire

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 …

E. Présentation d'une partie de l'outil développé pour l'analyse

La figure 5 illustre la partie de l'environnement permettant de représenter graphiquement les résultats d'analyse, sous forme un graphe (les nœuds représentent les objets du système et les arcs représente les liens de communication).

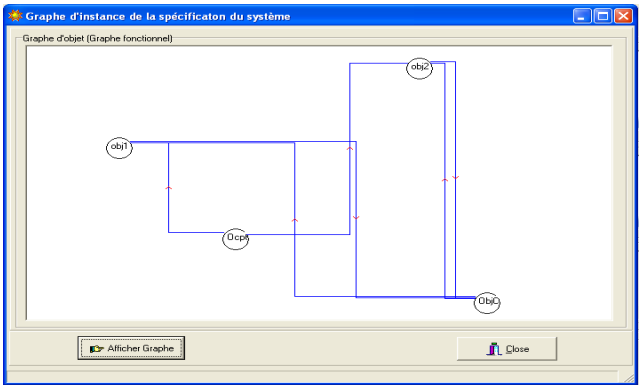


Figure 5: Exemple d'un graphe d'instance

VI. DESCRIPTION DES COMPOSANTS ARCHITECTURAUX ET DE PROTOCOLES DE COMMUNICATION

La spécification des composants de l'architecture cible constitue une activité très importante, 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 [13]. 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 leurs besoins d'une manière interactive.

A. Principales fonctionnalités de l'environnement de description de l'architecture cible

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 [11]. Il permet d'abord, de sélectionner des composants processeurs (processeurs à usages générales ou 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 [11], 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.

B. Composition de l'architecture cible

1) Description des composants processeurs

Les processeurs matériels et logiciels (FPGA, ASIC, Microprocesseurs à usages générales) sont utilisés pour exécuter tous les traitements 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.

2) Description des composants de communication

Pour mettre en œuvre 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

La figure 6 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.

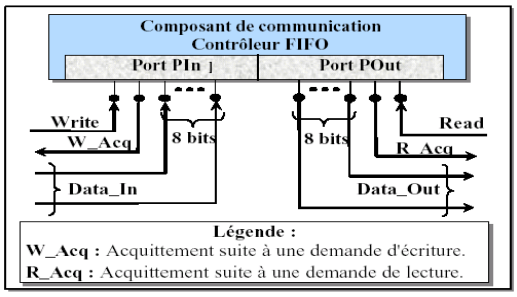


Figure 6: Exemple d'un composant de communication contrôleur FIFO [kou02]

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.

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. 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.

C. Modèle de protocole de communication

Le protocole de communication est un composant virtuel au même titre que les composants architecturaux [7]. Il est modélisé par un composant encapsulant des données et des traitements. La figure 8 illustre le modèle générique de protocole de communication de cette approche. Ce modèle permet de décrire les protocoles standards ou spécifiques.

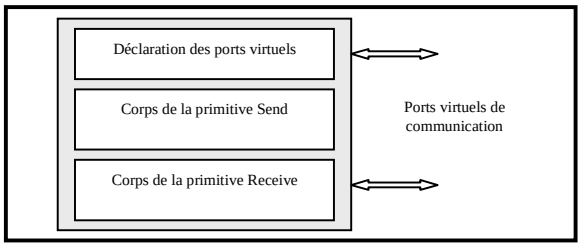


Figure 8: Modèle de protocole de communication

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.

La figure 9 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 9: Exemple d'un port virtuel.

D. Bibliothèque des protocoles de communication

La bibliothèque des protocoles de communication regroupe les protocoles de communication qui sont utilisés pour assurer la communication. Ces protocoles sont sélectionnés et instanciés lors de la synthèse de la communication. La structure de la bibliothèque de communication de notre approche est illustrée en tableau 2.

TABLE 2 : Structure de la bibliothèque de protocoles.

	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
…	…	…	..

VII. CONCLUSION

Nous avons présenté dans ce travail l'approche proposée et l'environnement développé pour l'analyse du code et la description de l'architecture cible.

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. 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 (partitionnement, cosynthèse d'interfaces, etc.).

L'environnement développé pour la description de l'architecture cible est basé sur des bibliothèques de composants architecturaux et de protocoles de communication réutilisables. 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. Elle permet également de décrire de nouveaux composants architecturaux et protocoles de communication de manière très simple, de les stocker dans la bibliothèque, et de les réutiliser en les paramétrant sur de nouveaux composants.

En effet, l'avantage de cette approche est que l'architecture cible résultante n'est pas figée comme dans de nombreux travaux.

Nos perspectives portent sur la continuité de ce travail afin d'aborder le problème de partitionnement et de cosynthèse d'interfaces.

REFERENCES

[1] GUPTA R.K. AND DE MICHELI G., "Hardware/Software cosynthesis for digital systems", IEEE Design and Test of Computers, Vol.10, N°3, pp.29-41, Sept. 01993.

[2] THOMAS D.E., ADAMS J.K. AND SCHMIT H., "A model and methodology for hardware-software codesign", IEEE Design and Test of Computers, Vol.10, N°3, pp.6-15, Sept. 1993.

[3] JANTSCH A., ELLERVEE P., OBERG J., HEMMANI A. AND TENBUNEN H., "Hardwaresoftware partitioning and minimizing memory interface traffic", Proc. of the European Design Automation Conference on Compiler Construction, CC- 94, pp.93-102, Edimburgh, Scotland, 1994.

[4] KALAVADE A. AND LEE E.A., "A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem", Proceedings of the Third International Workshop on Hardware/Software Codesign, pp.42-48, 1994.

[5] M. Koudil, Support de cours de magister, INI, 2004.

[6] GAJSKI D.D. AND VAHID F., "Specification and Design of Embedded Hardware-Software systems", IEEE Design and Test of Computers, pp.53-57, Spring 1995.

[7] M. Koudil, "Une approche orientée objet pour le Codesign", Thèse de Doctorat d'Etat, Institut National d'Informatique, 2002.

[8] BENDER A., "Design of an Optimal Loosely Coupled Heterogeneous Multiprocessor System", Proc. ED&TC, pp.275-281, 1996.

[9] DOURS D. ET AL., "Conception concurrente matérielle -logicielle de systèmes temps-réel-strict distribués", Technique et science informatique, Vol.18, N°10, pp.1137-1165, 1999.

[10] 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

[11] F.BERRHAIL, M. KOUDIL, Résolution du Problème de Cosynthèse D'interfaces de Communication pour le Codesign, Conférence Internationale des Technologies de l'Information et de la Communication sétif CITIC2009.

[12] HOU J. AND WOLF W., "Process partitioning for distributed embedded systems", Proc CODES'96, pp.70-75, Pittsburgh, USA, 1996.

[13]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.