Introduction

« In object-oriented programming a program execution is regarded as a physical model, simulating the behaviour of either a real or imaginary part of the world. Objects are viewed as computerized material that may be used to construct physical models. », O.L.*Madsen, 1988.*

1- Le problème.

Dans le domaine du traitement du signal, les architectures matérielles deviennent très rapidement obsolètes. Les environnements d'analyse, de conception et de développement doivent être disponibles très rapidement et s'adapter aux architectures qui répondront à la demande toujours croissante et toujours plus pressante du marché.

L'évolution des architectures est continue et les possibilités très rapidement accrues de la technologie[1] permettent d'intégrer toujours plus d'unités fonctionnelles, d'augmenter le parallélisme et ainsi de réaliser des architectures de plus en plus performantes mais aussi de plus en plus complexes. C'est pourquoi les méthodes et outils de conception doivent sans cesse être améliorés.

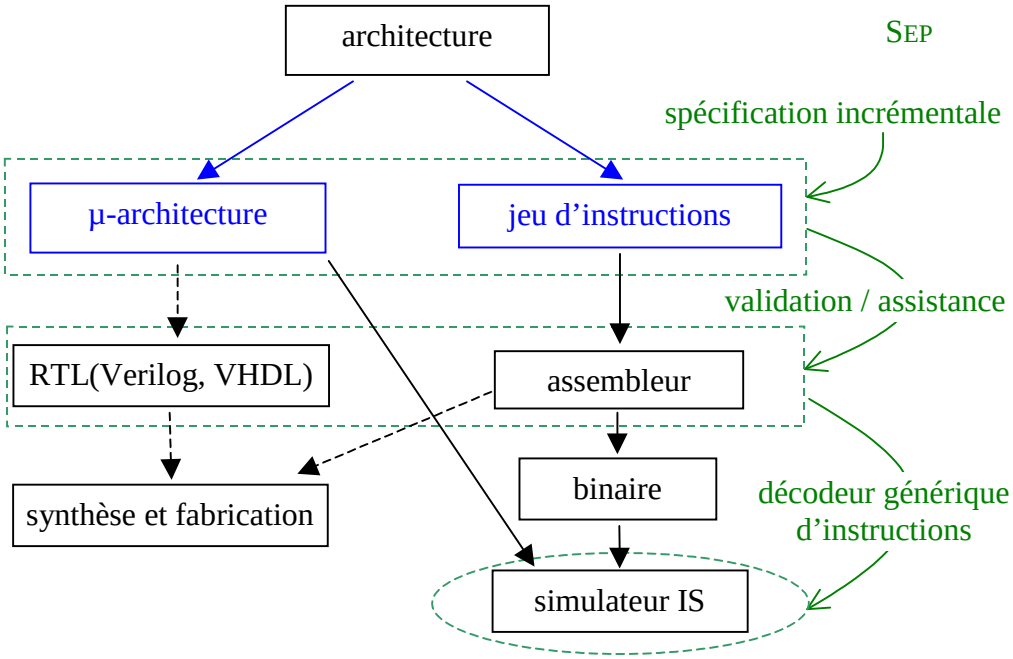

Figure 1 – Conception d'une architecture.

La conception d'une architecture matérielle consiste à définir sa micro-architecture et son jeu d'instructions (cf. Figure 1). La micro-architecture devra être modélisée dans un langage synthétisable et le jeu d'instructions permet de modéliser le décodeur d'instructions, de concevoir les outils d'analyse et de développement d'applications pour cette architecture : assembleur, compilateur, simulateur.

[1] Loi de Moore : En 1965, Gordon Moore, co-fondateur d'INTEL, observait que le nombre de transistors par centimètre carré de circuit intégré doublait chaque année depuis l'invention du circuit intégré en 1950. Il prédit alors que cette tendance se maintiendrait les années suivantes. En fait, les spécialistes évaluent maintenant que pour la prochaine décennie le taux d'intégration serait au minimum doublé tous les 18 mois.

Le problème ainsi défini en collaboration avec VLSI Technology – filiale de Philips Semiconductors, nous nous proposions de définir une méthode qui permette aux concepteurs de circuits intégrés d'évaluer les performances d'architectures matérielles numériques par rapport à des familles d'applications critiques, le plus tôt possible dans la phase de conception.

Notre objectif est alors de proposer une méthode et un environnement de conception et d'analyse non spécifique d'une architecture. Cette environnement est adapté à l'architecture visée à partir d'un modèle construit par une méthode incrémentale. Le niveau d'abstraction du modèle est choisi en fonction des besoins de l'analyse et de la conception. Cette méthode doit faciliter non seulement la modélisation interactive des architectures matérielles et des applications, mais aussi la réutilisation des modèles réalisés.

2- Les approches existantes : faiblesses et améliorations.

Notre proposition s'est inspirée du travail qui a été fait dans le cadre du programme RASSP (Rapid Application Specific Signal Processing) lancé en 1993 par la DARPA (Defense Advanced Research Projects Agency) et terminé en 1996. La DARPA a été créée par le ministère de la défense américain (DoD) afin de susciter et d'investir dans des projets à haut risques dont les résultats doivent permettre de conserver une avance technologique sur les autres nations[2]. Le programme RASSP a pour objectif de réinventer les méthodes de développement des processeurs de traitement du signal afin de réduire d'un facteur au moins égal à quatre le temps de conception ou d'amélioration d'architectures matérielles. Ce programme s'intéresse aussi à l'amélioration du cycle de vie et à la documentation relative au produit. Il s'est terminé par la présentation de quatre 'benchmarks' qui présentent les résultats obtenus avec et sans le RASSP Development Environment (RDE)[3].

Il existe actuellement deux langages normalisés pour la description et la synthèse du matériel : VHDL, Very High Speed Integrated Circuit (VHSIC), Hardware Description Language et Verilog-HDL. Les travaux du programme RASSP utilisent essentiellement VHDL car il est syntaxiquement très proche du langage ADA qu'ils utilisaient déjà par ailleurs. Dans ce programme deux approches se distinguent. La première consiste à utiliser une description entièrement fonctionnelle du système à modéliser, généralement avec des langages comme SPW de Cadence ; le VHDL est alors généré afin d'effectuer les simulations. Nous n'avons pas retenu cette approche, bien qu'elle présente de l'intérêt car les spécifications qui nous concernent ne sont pas de nature fonctionnelle, mais utilisent une notation plus proche des préoccupations pratiques du concepteur : schémas bloc d'assez haut niveau, c'est-à-dire d'un niveau supérieur ou égal au niveau transfert de registres (RTL).

La deuxième approche consiste à écrire directement du VHDL avec un niveau d'abstraction le plus haut possible. Les applications sont généralement décrites en ADA ou en C. Etant données les contraintes fortes de réutilisation et l'importance accrue des langages orienté-objets pour la conception de systèmes logiciels, des propositions d'adaptation du paradigme objet au langage VHDL ont été faites (**[AMS95]**, **[BaB96]**, **[Swa95]**, **[Put&co98]**). Cependant, VHDL est un langage fortement typé, basé sur la syntaxe du langage ADA et conçu avec des objectifs de spécification mais aussi et surtout de synthèse. L'ajout de concepts objets à un tel langage pose alors des problèmes techniques **[ScN95]** et conduit à notre avis à un langage très complexe, difficile à utiliser et à réutiliser.

De plus, ces langages ***ne sont pas aisément ouverts à l'intégration d'autres formalismes***, ce qui est de plus en plus nécessaire pour la modélisation des architectures modernes.

[2] http://www.darpa.mil/
[3] http://rassp.scra.org/

C'est pourquoi nous proposons une approche différente baptisée SEP : Simulation et Evaluation de Performances. Les contraintes de réutilisation et de modularité imposées nous conduisent aussi vers une approche basée sur la technologie objet. Il s'agit alors d'ajouter à un langage orienté objet pur comme Java les concepts nécessaires à la modélisation des architectures matérielles. Cependant, l'utilisation d'une approche objet n'est pas suffisante pour assurer la réutilisation. En effet, celle-ci fournit les méthodes pour réutiliser un objet, pas pour réutiliser une combinaison d'objets.

Nous proposons une approche orientée composants pour la modélisation et l'évaluation des performances des architectures matérielles. Ce choix nous rapproche des langages de description des architectures (ADL). Récemment, une enquête a été menée pour définir les propriétés communes à ce type de langage **[MeT00]**, le chapitre I met en évidence les aspects de SEP qui le rapprochent de la notion d'ADL.

3- La solution proposée.

La spécification des architectures visées est principalement constituée de schéma-blocs et les techniques de preuve et de validation formelles ne permettent pas actuellement l'analyse exhaustive des architectures de taille industrielle afin d'identifier les limitations et les goulots d'étranglement. Nous nous orientons alors vers une méthode d'évaluation de performances basée sur la simulation.

De plus, nous nous intéressons à des architectures numériques dont les applications manipulent des données discrètes. C'est pourquoi nous utilisons une technique de simulation à événements discrets adaptée, afin de permettre l'expression de propriétés spécifiques aux architectures visées. Notre environnement de simulation nous permet d'obtenir des résultats très rapidement, d'observer les éventuels défauts de l'architecture et de modifier de façon interactive le modèle.

Nous proposons une approche incrémentale qui permet le choix d'un niveau de modélisation adapté aux besoins des évaluations. Les modifications sont apportées très rapidement par le concepteur afin de valider les différents choix architecturaux par rapport à la famille d'applications visée. Les applications sont modélisées dans un langage proche de l'assembleur. Cette contrainte qui peut sembler excessive ne l'est pas forcément. En effet, les architectures étudiées sont principalement à base de processeurs de traitement du signal (DSP). Or, il n'existe actuellement aucune méthode efficace de compilation vers ce type d'architectures **[Leu97]**. C'est pourquoi dans ce domaine les concepteurs d'architectures ont l'habitude de travailler en assembleur. De plus, il ne s'agit de modéliser que les parties critiques des applications, ce qui réduit en général fortement la taille du code à écrire. Enfin, pour la modélisation d'architectures classiques plus orthogonales, il suffit d'utiliser les outils de compilation existants afin d'obtenir le code assembleur.

Les autres approches utilisent soit des générateurs de code à partir de langages de haut niveau et ne peuvent donc pas être bien adaptées aux DSP, soit des estimations assez grossières sans écrire de code assembleur. Cette dernière approche ne nous concerne pas puisque ce sont précisément des estimations fiables que nous voulons obtenir.

système

composant

Les composants sont définis par la DARPA en utilisant une définition récursive et hiérarchique. Un système est constitué de plusieurs parties et réalise une fonction ou un ensemble de fonctions. La plupart des systèmes sont des composants de systèmes plus importants. Les composants sont eux définis comme n'importe quelle partie du système, un composant peut être instancié une ou plusieurs fois et combiné avec d'autres composants pour

réaliser un système. Le comportement d'un composant peut être réalisé par une définition fonctionnelle ou structurelle.

Dans SEP, les composants sont assemblés par l'intermédiaire de connecteurs. La manipulation consiste à modéliser le comportement des composants élémentaires, à les assembler pour former des systèmes plus complexes nommés *modules*, à s'assurer par simulation que le comportement composite correspond aux attentes du concepteur, à observer les éventuels dysfonctionnements et à permettre de rapides modifications jusqu'à l'obtention des performances souhaitées. Le code Java équivalent est alors généré puis compilé, le module ainsi créé est archivé, il sera utilisé comme composant pour la construction d'un module.

4- Le fonctionnement de SEP.

Le fonctionnement de SEP est résumé par la Figure 2.

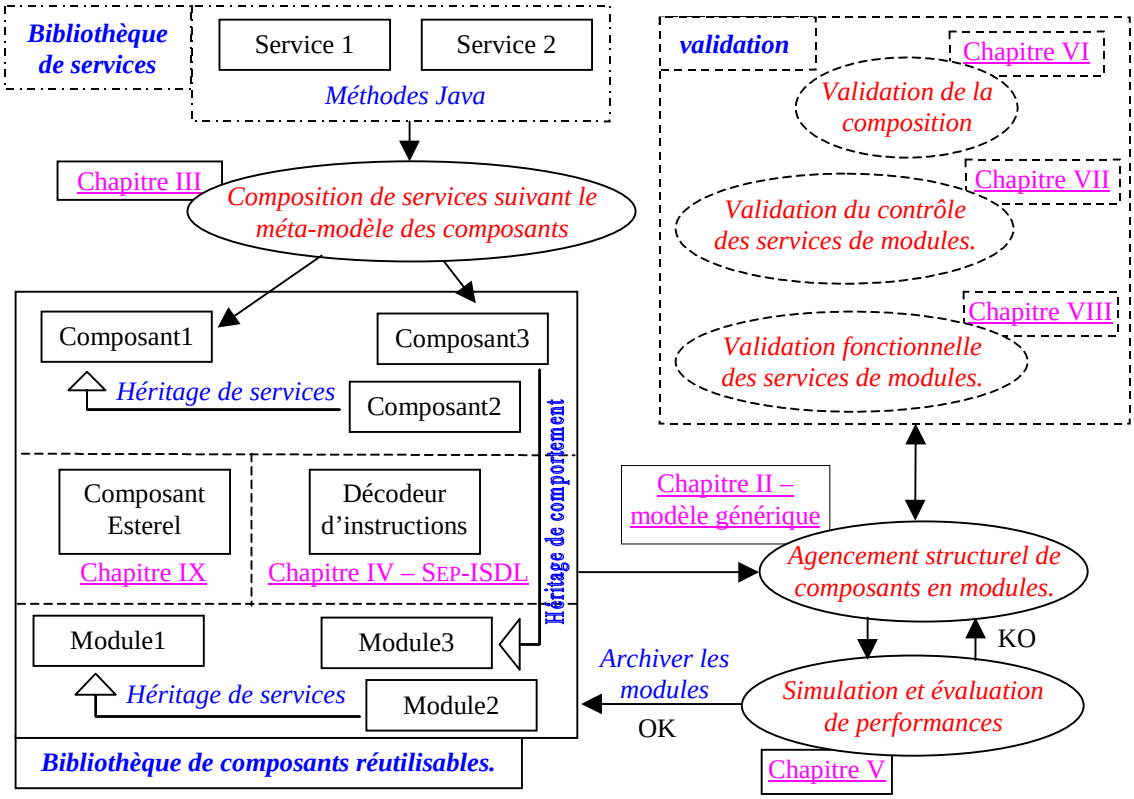

Figure 2 – Schéma de principe du fonctionnement de SEP.

L'environnement construit n'est pas spécifique d'un type d'architectures, il permet de modéliser des architectures câblées (co-processeurs), mais aussi des architectures de type DSP, RISC, CISC, VLIW, des architectures micro-programmées ou encore des ASIC. Pour atteindre cet objectif, nous avons élaboré un modèle objet représentant la structure générique des architectures visées. Ce **modèle objet générique**, décrit dans le langage Unified Modelling Language (UML [4]) la définition récursive et hiérarchique du composant et permet l'agencement de composants pour constituer des modules. Il est présenté par le chapitre II.

Il met en évidence quatre familles de composants élémentaires décrites par quatre classes : *Edge*, *Level*, *Time*, *Input*. Les composants sont donc des objets, instances d'une de ces quatre

[4] Une description succincte de ce langage de modélisation est fournie en annexe.

classes. L'héritage permet une plus grande réutilisation. L'encapsulation garantit l'autonomie des composants ce qui nous permet d'intégrer plusieurs paradigmes (cf. chapitre IX).

Le modèle établit la façon dont le comportement est joué, le mode de communication des composants ainsi que la hiérarchie de composition des différents composants. Il décrit les composants comme un ensemble de services concurrents et communicants. Les composants communiquent par envoi de messages, ils s'échangent des valeurs de types très hétérogènes.

La description du comportement d'un composant commence par la définition des services fournis. Ces services sont décrits par les méthodes d'une classe « fournisseur de services ». Il faut modéliser les contraintes d'exécution des différents services. Puis, le concepteur doit associer à chaque service un mode de déclenchement correspondant à un type d'événement SEP et relatif à une des quatre familles de composants. Enfin, il faut encapsuler ces services dans un objet *MaterialComponent* qui représente un composant, permet l'interconnexion avec les autres composants et la réutilisation. Le mécanisme qui permet de construire un composant à partir de services est semi-automatique, il est assisté par une interface graphique. Cette manipulation est rendue possible par la définition d'**un méta-modèle des composants SEP** présenté par le chapitre III.

Afin d'améliorer la réutilisation des modules, le contrôle associé aux modules peut être défini comme un ensemble de services. Ces services sont définis comme une composition de services élémentaires exécutés en séquence ou en concurrence. Un module peut alors fournir des services pour la description de nouveaux composants. On définit ainsi l'**héritage d'une description structurelle** dont la sémantique était jusqu'alors souvent restreinte à un héritage de l'interface matérielle.

La **simulation** se fait en jouant le comportement des différents composants excités par les messages qui s'échangent. Des composants sont disposés aux endroits critiques de l'architecture afin d'observer la réaction en fonction des différentes configurations. Ces composants appelés observateurs permettent l'**évaluation de performances.**

Le jeu d'instructions des architectures programmables est modélisé grâce à la définition d'un langage appelé SEP-ISDL.

La première partie présente les aspects sur la modélisation, la simulation et l'évaluation des performances dans SEP.

Après validation de ce procédé, l'objectif est alors d'utiliser les modèles construits dans SEP comme des **modèles de référence** pour la conception de nouvelles architectures. Les résultats obtenus dans SEP, utilisés comme référence, permettent la mise au point de modèles qui peuvent être synthétisés (VHDL ou Verilog). Toutefois, les techniques de simulation permettent principalement de valider un système en simulant le plus grand nombre de jeux de test. Nous montrons alors comment notre modèle objet rend possible la mise en place des techniques de validation partielle de propriétés statiques ou dynamiques dans cet environnement de simulation. Ceci nous permet de réduire le nombre de jeux de test nécessaires, d'augmenter notre confiance dans les modèles construits et de nous convaincre que de mauvaises propriétés ne devraient pas survenir dans certains cas non simulés.

Pour cela, nous avons identifié quatre étapes de validation :

- **La validation de la composition :** Ce mécanisme, détaillé au chapitre VI, utilise l'information contenue dans les services pour inférer le type des données échangées. En effet, les connecteurs ne sont pas explicitement typés, il faut s'assurer que les connexions effectuées ne sont pas aberrantes. Dans un premier temps, on s'assure que le type des

données émises est compatible avec le type des données attendues. Dans un deuxième temps, il faut déterminer dynamiquement les opérateurs les mieux adaptés aux types des données échangées.

- **La validation du contrôle** lié à l'exécution des services de module et du jeu d'instructions : La concurrence introduite dans la définition des services de module peut être partiellement validée en s'assurant que les ressources physiques nécessaires à l'exécution concurrente des services de module seront suffisantes. On établit ainsi que le contrôle associé à l'exécution de ces services est correct. Il s'agit principalement d'allouer des ressources et d'ordonnancer leur utilisation concurrente. Cette étape est détaillée par le chapitre VII.
- **La validation fonctionnelle** des services et du jeu d'instructions : Une fois la correction du contrôle associé aux services de module établie par l'étape précédente, il s'agit de vérifier leur fonctionnalité. Pour cela, il faut permettre une description algébrique de la fonction que l'on souhaite réaliser, puis sa comparaison avec l'expression algébrique de la fonction effectivement réalisée calculée en fonction de l'architecture par l'environnement SEP. On détermine ainsi la correction fonctionnelle du comportement construit. Cette étape est décrite par le chapitre VIII.
- **La validation du comportement** des composants orientés contrôle : Nous avons intégré une machine d'exécution du langage Esterel pour permettre la description de composants dont le comportement est réactif et à contrôle dominant. Cette intégration présentée au chapitre IX permet de valider localement certaines propriétés grâce à l'environnement synchrone.

Les aspects concernant la validation de propriétés sont exposés dans la deuxième partie.

Enfin, une troisième partie présente la mise en œuvre et les résultats obtenus sur des exemples choisis par la société VLSI pour valider la méthode.

Ce document décrit en détail la méthode proposée, il vise à mettre en évidence l'intérêt de l'utilisation d'une technique objet pour la modélisation d'architectures matérielles. En particulier, il montre comment notre modèle objet peut être étendu et enrichi. De nombreuses extensions supplémentaires sont possibles, un aperçu rapide des possibilités d'extension non développées actuellement est donné en perspectives.