UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE

THÈSE

présentée et soutenue publiquement le 19 Décembre 2013
pour obtenir le titre de
DOCTEUR SPÉCIALITÉ INFORMATIQUE
par

SANA CHERIF

Approche basée sur les modèles pour la conception des systèmes dynamiquement reconfigurables : de MARTE vers RecoMARTE

Composition du jury :

Examinateur :	ElBey BOURENNANE	Professeur Université Bourgogne
Rapporteurs :	Tahar KECHADI	Professeur University College Dublin
	Abderrazak JEMAI	Maître de Conférences INSAT de Tunis
Directeur :	Jean-Luc DEKEYSER	Professeur Université Lille 1 Sciences et Technologies
Co-Directeur :	Samy MEFTALI	Maître de Conférences Université Lille 1 Sciences et Technologies

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
LIFL - UMR 8022 - Cité Scientifique, Bât. M3 - 59655 Villeneuve d'Ascq Cedex

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

CHAPITRE 1

INTRODUCTION GÉNÉRALE

1.1 Contexte

Un système embarqué est un système intégré au sein d'un autre système plus large avec lequel il est interfacé et pour lequel il réalise des fonctions particulières (contrôle, surveillance, communication, etc.). Afin de réaliser ces fonctions, il est nécessaire de faire recours au logiciel autant que le matériel. Dans certaines applications, les systèmes embarqués doivent fonctionner en temps réel car ils doivent gérer des informations et en déduire des actions dans un délai maîtrisé.

Aujourd'hui, une course à la miniaturisation assez remarquable permet de concevoir ces systèmes toujours de plus en plus complexes. De nombreux domaines de la vie quotidienne sont couverts par l'utilisation des systèmes embarqués, on cite : la télécommunication, l'automobile, l'aéronautique, l'électroménager, etc. Ainsi, cette évolution permet aujourd'hui d'intégrer toujours plus de fonctionnalités au sein d'une même puce. On parle alors de systèmes sur puce ou (SoC - *System-on-Chip*). Ils constituent les composants de base pour de nombreux systèmes embarqués. Cette évolution est illustrée dans la figure 1.1.

Un SoC est un circuit intégré qui comporte un ensemble de composants matériels tels que les unités de calcul (hardcore, softcore, DSP, etc.), les mémoires (RAM, ROM, flash, etc.), connectés entre eux par des bus de communication suivant plusieurs topologies (crossbar, réseau sur puce, etc.), les périphériques d'entrée/sortie (USB, Ethernet, FireWire, etc.), les capteurs, etc. La figure 1.2 illustre un exemple d'architecture d'un système sur puce [1].

1. http ://www.xilinx.com/products/silicon-devices/soc/zynq-7000/use-cases-and-markets/index.htm

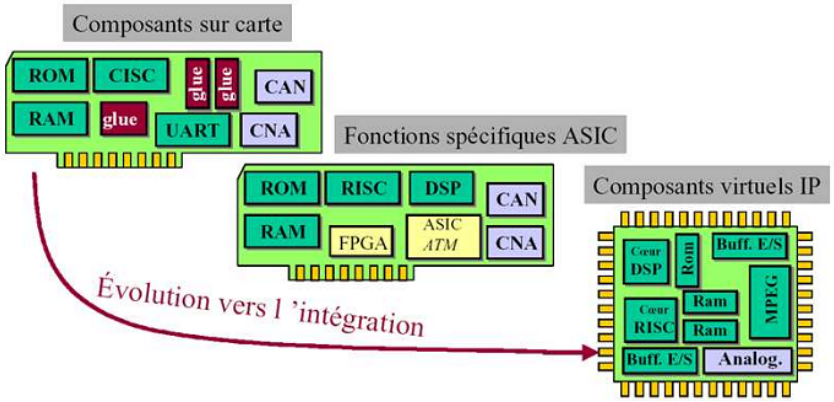


FIGURE 1.1: Evolution de la capacité d'intégration [50]

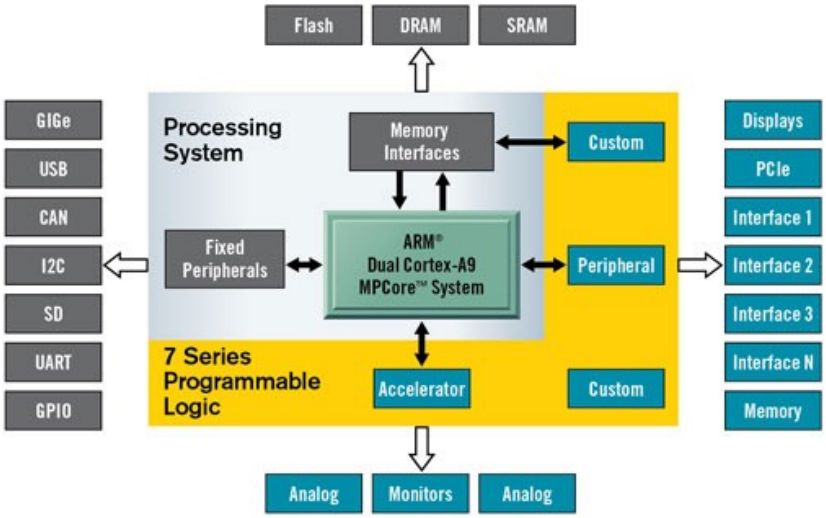


FIGURE 1.2: Exemple d'architecture d'un système sur puce

L'utilisation des SoCs va se répandre grâce aux systèmes re-programmables (*SoPC : System on Programmable Chip*) de type FPGA (*Field-Programmable Gate Array*). Ces systèmes reconfigurables supportent des composants implémentés sur du matériel reconfigurable. De ce fait, ils offrent une grande flexibilité par rapport aux SoCs classiques. La notion de la reconfiguration dans ces systèmes leur permet d'être reconfigurés un nombre illimité de fois et offre aux concepteurs la possibilité d'ajouter de nouvelles fonctionnalités en modifiant le système, même après sa fabrication. La reconfiguration dynamique, un type spécifique de la reconfiguration, permet la modification d'un système au cours de l'exécution. Ainsi, le système peut s'adapter aux changements dûs à certaines contraintes de qualité de service, aux exigences des utilisateurs, de consommation d'énergie, etc.

Les FPGAs se présentent comme une solution adéquate pour implémenter la reconfiguration dynamique surtout avec l'évolution de certains types d'FPGA qui supportent la reconfiguration dynamique partielle (RDP). La RDP offre la possibilité de reconfigurer seulement une partie de l'FPGA alors que le reste du circuit continue son exécution, sans interruption.

Cependant, la conception de ces systèmes ne peut être implémentée par des méthodes classiques où la conception du matériel précède celle du logiciel. Ces méthodes sont incompatibles avec les contraintes de temps de mise sur le marché actuelles.
L'évolution de la démarche de conception des systèmes électroniques embarqués passe par une élévation du niveau de représentation du système illustré dans la Figure 1.3. Cette démarche part d'une description schématique électrique du système (*full custom*), puis vers une conception de porte logique (cell based), suivie d'une description sous forme de langage matériel et désormais une représentation directement au niveau système. L'avantage d'une telle évolution est la réutilisation des fonctionnalités déjà développées sous forme d'IP (*Intellectual Property*).

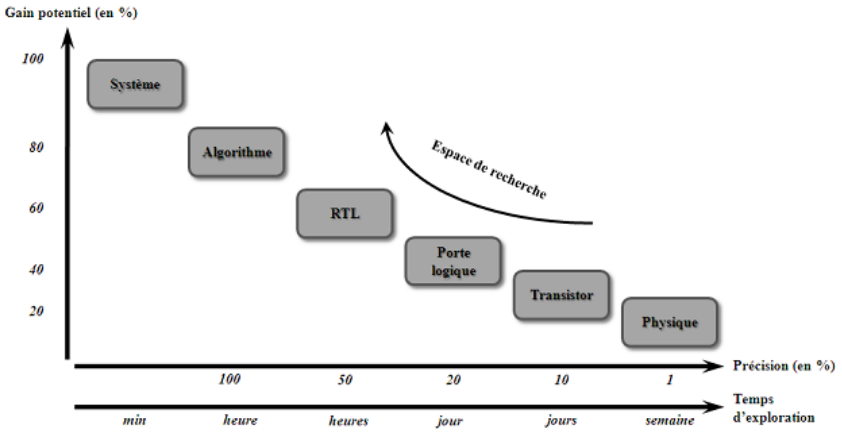


FIGURE 1.3: Impact de l'élévation du niveau de représentation d'un système électronique sur le temps d'exploration, la précision par rapport au gain potentiel de productivité [50]

Il s'agit d'associer dans un même composant des cœurs de processeurs exécutant des programmes et des blocs fonctionnels spécifiques appelés IPs. C'est une conception dite conjointe du matériel et du logiciel (*Co-Design*), elle requière l'introduction de nouvelles méthodologies traitant des spécifications logicielles et des blocs matériels synthétisables.

Les IPs utilisés par le concepteur sont souvent d'origines diverses et ayant des modèles hétérogènes (différents niveaux d'abstractions). Cette approche permet d'améliorer le délai de mise sur le marché (*time to market*), mais nécessite éventuellement de nouvelles méthodes de conception. Les méthodes de conception se sont toujours adaptées, avec un temps de retard, pour intégrer au mieux les nouvelles contraintes d'utilisation des nouveaux composants, et contenir le temp de conception.

La maîtrise de la complexité croissante des systèmes temps réel embarqués est l'un des enjeux majeurs de l'industrie européenne. Le monde industriel doit se donner les moyens de modéliser, de concevoir, de développer, d'intégrer et de qualifier comme sûrs des systèmes continuellement évolutifs et facilement maintenables. En conséquence, les moyens mis à disposition des concepteurs sont à un stade préliminaire et la conception ciblant les systèmes reconfigurables en général souffre d'un fossé (*gap*) de productivité. La reconfigurabilité de tels systèmes augmente encore la complexité de leur conception en nécessitant des tâches supplémentaires liées au contrôle de l'adaptation dynamique.

Plusieurs méthodologies sont proposées dans la littérature pour manipuler le Co-design

ciblant ces systèmes. Elles se distinguent essentiellement par : les concepts de modélisation utilisés, les modèles de l'architecture cible et la méthode d'allocation, etc.

Dans ce contexte, une méthodologie de conception haut niveau est requise afin de faciliter le travail des concepteurs. Cette méthodologie doit inclure également les moyens de modéliser la reconfiguration dynamique des plateformes ciblées (FPGA) et garantir la réutilisation, la portabilité et l'automatisation du processus de conception.

1.2 Le contexte du projet FAMOUS

Les travaux présentés dans ce manuscrit s'inscrivent dans le cadre d'un projet ANR (Agence Nationale de la Recherche), au nom de FAMOUS (FAst Modeling and Design FlOw for Dynamically ReconfigUrable Systems). Ce projet a pour objectif d'introduire une méthodologie complète de modélisation pour la spécification et la conception des systèmes embarqués dynamiquement reconfigurables. FAMOUS est un projet de recherche avec un impact industriel immédiat. En fait, il facilitera la conception des systèmes reconfigurables et la rendra plus rapide. L'outil obtenu dans ce projet est prévu pour être utilisé par les concepteurs dans les entreprises industriels ainsi que les chercheurs dans le milieu académique, en particulier pour la conception des applications sur des systèmes modernes telles que la caméra intelligente (*smart camera*), le traitement d'image et vidéo, etc. Les outils de FAMOUS sont basés sur des standards pour la conception et la modélisation. En effet, la modélisation commence d'un haut niveau d'abstraction en utilisant le standard MARTE et une version étendue de MARTE. La phase de simulation et de synthèse des modèles est obtenue par des transformations automatiques de modèles en utilisant l'approche d'IDM (Ingénierie Dirigée par les Modèles). Ces techniques contribuent à réduire considérablement le temps de mise sur le marché.

La thèse s'est déroulée principalement au sein de l'équipe DaRT affiliée au laboratoire LIFL de l'Université des Sciences et Technologies (Lille), en collaboration avec mobilité au sein du laboratoire LE2i à l'Université de bourgogne (Dijon). Les partenaires académiques de ce projet font partie du lab-STICC à l'Université de Bretagne Sud (Lorient) et à l'INRIA Rhône-Alpes (Grenoble). Le partenaire industriel est présent à Nantes en tant qu'éditeur du logiciel Sodius. Ces partenaires se répartissent les responsabilités dans la réalisation de ses segments.

Le but de cette thèse décrit le premier axe de recherche du projet FAMOUS à savoir établir une méthodologie de conception haut niveau pour la modélisation des systèmes dynamiquement reconfigurables, principalement les FPGAs. De plus, nos contributions définissent un nouveau profil qui étend le standard MARTE et qui prend en compte tous les aspects de la reconfiguration dynamique. Ainsi, les résultats de nos travaux sont considérés comme un point de départ pour le bon déroulement des autres thèses des différents acteurs.

A partir d'une modélisation à haut niveau, les autres acteurs interviendront afin de compléter le flot de FAMOUS illustré dans la figure 1.4 :

– Construction et Génération d'un contrôleur de reconfiguration : [35] propose une méthodologie de conception *par contrainte* du contrôle ciblant une représentation synchrone. Cette

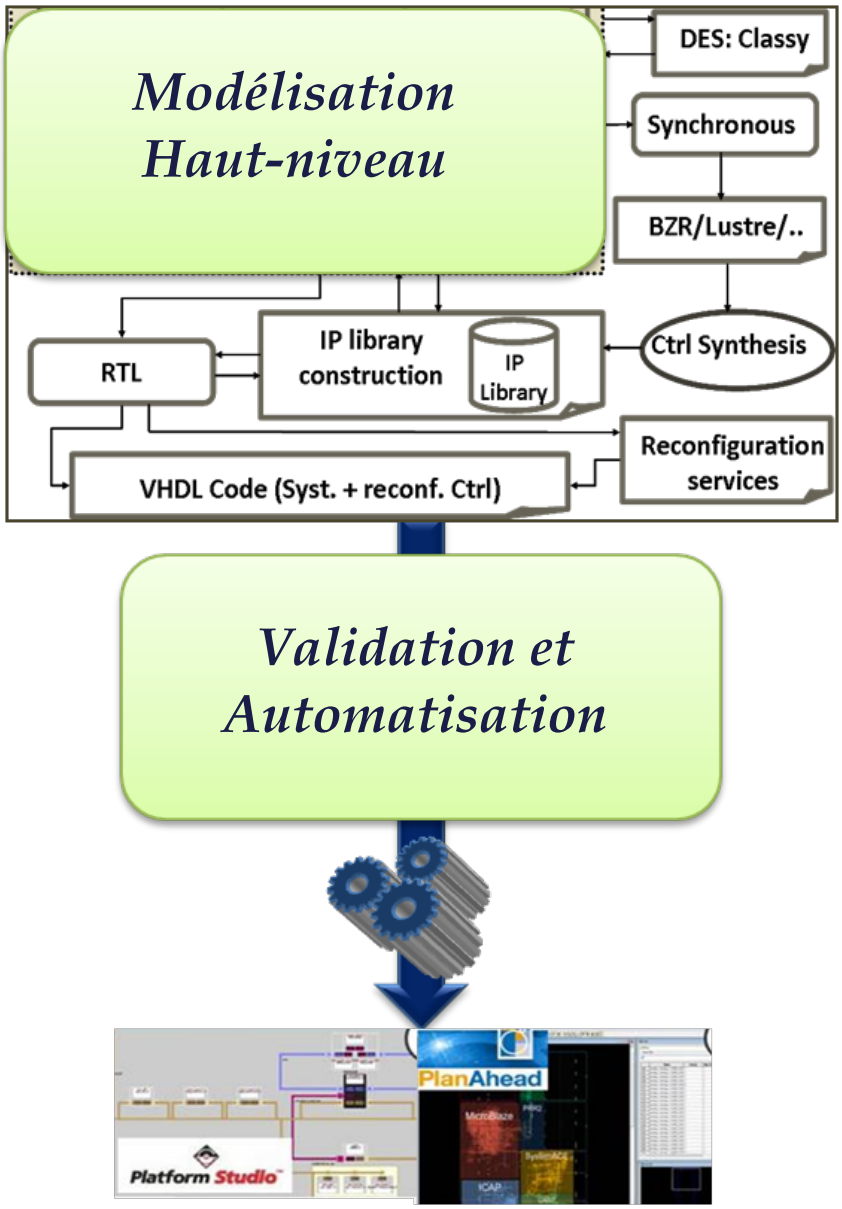


FIGURE 1.4: Le flot complet de FAMOUS

représentation est exploitable par une technique correcte par construction : la synthèse de contrôleur discret.

– Génération de la description de la plateforme en utilisant des méta-modèles associés aux outils et langages d'implémentation (EDK, VHDL, etc.). [65] propose une représentation intermédiaire IP-XACT entre les modèles MARTE et les outils de la RDP de Xilinx et définit les différentes règles de transformation jusqu'à la génération de code.
– Définition d'un ensemble de services de la RDP à implémenter par le système : [97] présente une approche permettant la sauvegarde de contexte des modules reconfigurables afin d'améliorer la dynamicité de la reconfiguration partielle des systèmes.
– Développement et exploration de l'analyse et la vérification des configurations et du contrôle de la reconfiguration : dans leurs travaux, les auteurs [8][9] présentent une

spécification des techniques permettant de vérifier l'absence de cycle de causalité, le déterminisme ou la réactivité des comportements.

1.3 Approche envisagée

Avec la croissante complexité des systèmes sur puce (en particulier ceux implantés sur les FPGAs), il est devenu impossible de les concevoir dans un bas niveau (RTL : *Register Transfert Level*) où il faut préciser chaque détail du comportement des composants. De ce fait, les ingénieurs se sont mis face à un grand défi pour réussir à maîtriser cette complexité lors de la phase de conception de ces systèmes et d'arriver à une conception rapide sous de fortes contraintes de qualité et de temps de développement. Pour dépasser ce défi, ils ont fait appel à de nouvelles méthodes de conception basées sur des concepts d'abstraction de haut niveau ainsi que le raffinement jusqu'à atteindre le niveau bas, plus proche du système à réaliser.

Nos travaux décrits dans ce manuscrit proposent une approche de conception dirigée par les modèles pour la conception des systèmes dynamiquement reconfigurables, en particulier les FPGAs. Cette approche est basée sur des standards OMG et a pour objectifs de garantir la flexibilité, la réutilisabilité et l'automatisation.

- *L'élévation du niveau d'abstraction* : permet de cacher les détails techniques aux concepteurs souvent source d'erreurs et réduire la complexité de conception. Les utilisateurs peuvent modéliser leurs systèmes à haut niveau d'abstraction sans qu'ils soient experts des FPGAs modernes.
- *Modélisation basée sur le standard UML MARTE* : Les modèles décrits par UML sont de nature graphique et rendent la compréhensibilité d'un système plus facile. La caractéristique de *générécité* dans le profil, nous laisse complètement indépendant de toute technologie.
- *La Réutilisation* : les méta-modèles ainsi que les transformations de modèles proposés, peuvent être réutilisés pour les besoins d'une nouvelle implémentation de l'application sur une nouvelle plateforme. Ceci permet également de faciliter le travail du concepteur et d'améliorer sa productivité.
- *l'Automatisation* : est offerte par l'IDM ; elle permet à notre approche d'automatiser le flot proposé ce qui facilite et accélère la conception. De plus, elle permet la génération de code à partir d'une modélisation de haut niveau.

Les premiers résultats de nos travaux ont été publiés dans [21]. Notre méthodologie permet la modélisation des systèmes dynamiquement reconfigurables basés sur les FPGAs et ce, à partir d'un haut niveau d'abstraction. Cette modélisation, basée principalement sur le standard UML MARTE, prend en compte la notion de la reconfiguration dynamique des FPGAs. Cependant, bien que riche en concepts, le profil MARTE manque de certains concepts qui permettent la modélisation de la reconfiguration dynamique. Ainsi, nous identifions ces concepts manquants et proposons des extensions afin de compléter ce profil. Les concepts rajoutés définissent un nouveau profil, qui étend le profil UML MARTE et qui prend en compte la notion de la reconfiguration dynamique.

Ce profil, baptisé RecoMARTE (*Reconfigurable MARTE*) vient répondre également aux besoins des différents acteurs du projet FAMOUS, en intégrant les concepts permettant :

- La modélisation d'une application à caractère reconfigurable en prenant en compte son comportement ;
- La modélisation du contrôle de la reconfiguration dynamique qui permet de gérer ces tâches ;
- La modélisation d'un niveau de déploiement, intermédiaire entre une description haut niveau MARTE et les outils de la RDP de Xilinx ;
- La modélisation de membrane qui permet la sauvegarde de contexte des modules reconfigurables ;
- La modélisation de la plateforme physique de l'FPGA qui permet de se rapprocher le plus des détails d'implémentation.

Dans le but de générer une description complète du système, notre flot de conception proposé nous permet de passer d'une modélisation de haut niveau vers une description complète du système conçu et la génération de fichiers utiles pour l'environnement EDK de Xilinx.

Afin d'assurer l'automatisation de notre chaîne de conception, notre approche s'appuie sur l'Ingénierie dirigée par les modèles. Cette approche nous permet donc de rendre transparents les détails d'implémentation pour les concepteurs. Elle accélère également la phase de conception en évitant les erreurs dues à la manipulation directe de ces détails. De ce fait, nous présentons une spécification détaillée des règles de transformations de modèles qui permettront l'automatisation de notre flot et l'intégration complète dans le flot du projet FAMOUS.

1.4 Organisation du document

La suite de ce manuscrit comporte sept chapitres qui s'articulent en trois parties. La première partie, composée de deux chapitres, présente les concepts de base et un état de l'art du domaine d'étude. La seconde partie présente nos contributions et notre méthodologie. Enfin, la dernière partie décrit l'automatisation de notre flot et valide nos travaux en présentant une étude de cas particulière, avant de conclure le manuscrit.

Partie 1 : Problématique et analyse de l'état de l'art

Le deuxième chapitre présente les architectures reconfigurables ainsi que le besoin d'une nouvelle méthodologie de conception de ces systèmes. Les concepts fondamentaux utilisés dans nos travaux sont également présentés : l'IDM et le profil standard MARTE. Le troisième chapitre, différents travaux qui traitent la conception des systèmes reconfigurables sont présentés et discutés. Les principaux aspects traités dans ce chapitre sont : la co-conception des SoCs en général, les profils UML pour la modélisation des SoCs et une étude des différents travaux qui proposent la modélisation de la reconfiguration dynamique. Tous ces aspects sont discutés en détails.

Partie 2 : modélisation à haut niveau

Cette partie est consacrée à la contribution de cette thèse. Le chapitre 4 introduit le flot de

conception proposé basée en partie sur l'IDM et qui prend en compte la modélisation physique des FPGAs et le contrôle de la reconfiguration dynamique.

Le chapitre 5 présente une méthodologie de conception de FPGAs et présente les différentes extensions du profil MARTE en introduisant le nouveau profil RecoMARTE. Ce profil prend en compte les aspects de la reconfiguration dynamique. Des exemples justifiant ces concepts seront présentés à chaque étape de la méthodologie.

Partie 3 : Validation et Automatisation du flot

Les règles de transformations permettant le passage des modèles de haut niveau vers une description technique complète du système, sont présentées dans le chapitre 6. Le dernier chapitre valide l'approche par l'expérience. Ce chapitre présente une étude de cas d'une application afin d'illustrer les différents étapes de conception présentées précédemment.

Enfin, la conclusion générale du manuscrit résume un bilan du travail réalisé et donne quelques perspectives au travail présenté et de son intégration au sein du projet FAMOUS.

Première partie

Problématique et analyse de l'état de l'art

CHAPITRE 2

MODÉLISATION DES ARCHITECTURES RECONFIGURABLES

2.1 Introduction

Aujourd'hui, la complexité des systèmes électroniques est telle qu'elle nécessite l'utilisation d'architectures hétérogènes, composées d'unités de nature différente, ce qui rend la conception de ces systèmes difficile. Cette croissance de complexité place les SOCs face à de multiples contraintes. En effet, ils doivent assurer une certaine fiabilité, minimiser leur temps d'arrivée au marché (Time to Market) et minimiser le coût de fabrication ainsi que la taille du système.

Ce chapitre s'attache à présenter les systèmes sur puce reconfigurables, en particulier les FPGAs puis définir la notion de la reconfiguration dynamique partielle. Certains détails sur la conception des FPGAs sont par la suite présentés avant d'entamer la partie introduisant les méthodologies de conception. Les standards OMG qui seront utilisés le long de ce manuscrit sont alors présentés : l'Ingénierie dirigée par les modèles (IDM) et le profil UML MARTE.

2.2 Systèmes sur puce reconfigurables

2.2.1 Présentation

La principale différence entre une architecture d'un SOC classique et d'un SOC reconfigurable est la présence de régions reconfigurables ; des FPGAs ou des CPLDs (Complex Programmable Logic Devices) font partie des composants du SOC. D'autres systèmes peuvent être implémentés en tant que ASIC (Application Specific Integrated Circuits). Le circuit ASIC ne peut exécuter qu'une seule application avec des contraintes de performance très élevées (temps de latence, la superficie, la consommation d'énergie, le débit). Quant aux systèmes sur puce reconfigurables, ils sont conus pour exécuter plusieurs applications différentes en s'appuyant sur les mêmes capacités matérielles. Ainsi, ils introduisent la notion de *matériel virtuel*.

La notion de virtualisation signifie qu'une application s'exécute grâce à un matériel d'exécution virtualisé, par opposition au matériel physique. Dans [69], les auteurs emploient ce mot pour désigner plusieurs notions, notamment l'exécution virtuelle, les machines virtuelles et le partitionnement temporel.

En termes de fabrication, les solutions basées sur les ASICs produisent un SOC extrêmement coûteux avec un temps de mise sur le marché assez long. De plus, ceci nécessite l'intervention de différentes équipes et plusieurs concepteurs, ce qui peut engendrer des erreurs dans le cycle de conception. La solution alternative est l'utilisation des FPGAs pour la construction des SOCs reconfigurables.

2.2.2 Les FPGAs

Les circuits programmables tels que les FPGAs, sont des circuits qui peuvent être programmés après leur fabrication. C'est une solution qui vient répondre au manque de flexibilité dont souffrent les SOCs basés sur les ASICs. L'avantage de l'utilisation de ces circuits est la rapidité de leur conception et leur re-programmation. En effet, la conception de ces circuits est réduite par rapport à celles des ASICs grâce aux changements immédiats sur les FPGAs avant leur fabrication. Ce qui fait des FPGAs une solution pour le prototypage des systèmes à base d'ASICs.

Un autre avantage est que ces circuits peuvent être reconfigurés pour s'adapter à une application donnée. Leur reconfiguration peut être effectuée un nombre illimité de fois.

De manière générale, un FPGA est composé de deux couches. La première couche contient des blocs logiques reconfigurables dit CLBs : *Configurable Logic Blocks* pour le vendeur Xilinx ou LEs : *Logical Elements* pour le vendeur Altera). Ces blocs illustrés dans la figure 2.1 sont

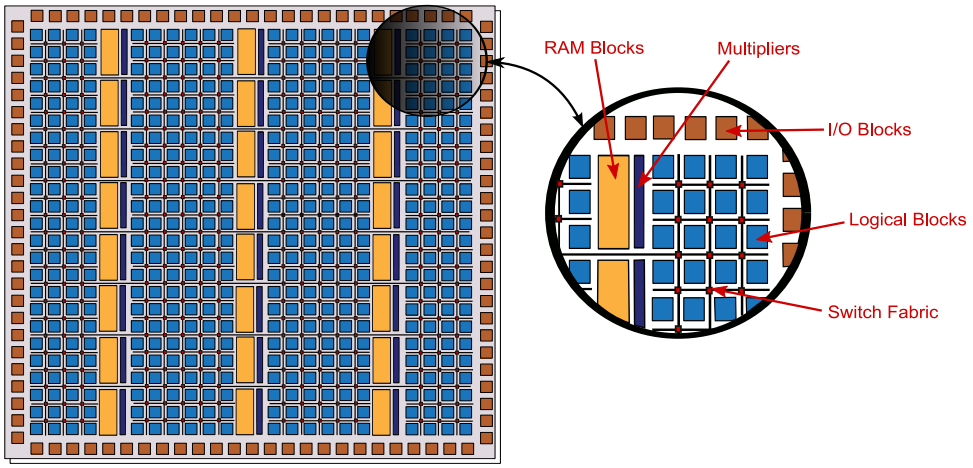


FIGURE 2.1: La couche matérielle configurable d'un FPGA Xilinx

interconnectés par une hiérarchie d'interconnexions reconfigurables. Ils sont généralement composés de LUTs (Look Up Tables) pour réaliser des fonctions élémentaires et des bascules flip-flop pour la mémorisation. Les fils de connexion sont des fils horizontaux et verticaux disposés entre les lignes et les colonnes des blocs logiques.

Les blocs d'E/S (IOBs) sont eux aussi connectés aux lignes de routage adjacentes. L'FPGA intègre également dans cette couche un nombre variable de ressources supplémentaires, cette fois-ci non reconfigurables mais optimisées pour un besoin transversal particulier : blocs de mémoire (BRAMs), processeurs de signal numérique ou DSP (Digital Signal Processor), multiplicateurs, etc.

La seconde couche constitue la mémoire de configuration. Pour écrire dans cette mémoire, on utilise des fichiers binaires appelés *bitstreams*. Ces fichiers contiennent les informations de contrôle pour la configuration ainsi que des données de configuration. Ces données permettent de décrire l'état des éléments reconfigurables sur la première couche.

Traditionnellement lors de la conception d'un SOC reconfigurable à base de FPGA, la partie logicielle, composée de pilotes pour le matériel et/ou de programme(s) compilable(s) pour un ou plusieurs processeur(s), est généralement écrite en C, C++, voire en assembleur. Quant à la partie matérielle à virtualiser, la spécification s'effectue via un langage de description d'architectures, tel que VHDL ou Verilog. Cette synergie de conception matérielle/logicielle se nomme *co-design*(Co-conception) et est couramment menée via un outil de CAO (*Conception Assistée par Ordinateur*) pour l'électronique. Les constructeurs des FPGAs conçoivent généralement ce type d'outils ciblant uniquement leurs produits (EDK pour Xilinx, SoPC Builder pour Altera, etc.).

Suite à la phase du co-design, vient la phase qui permet de traduire, pour le matériel, les modules d'architecture spécifiés en descriptions de circuits appelées *netlists*. Ensuite, la phase du «*Placement et routage*» (Place-and-route) permet de trouver leurs positionnements et interconnexions adéquats sur l'FPGA ciblé.

Enfin, la combinaison de la description du circuit et du placement et de routage, est inscrite dans un fichier bitstream qui peut ensuite être chargé sur l'FPGA afin de virtualiser le matériel

correspondant.

Dans le cas où la place mémoire est suffisante, le bitstream peut également contenir la partie logicielle sous forme de binaire compilé. Dans le cas contraire, il est nécessaire d'embarquer, sur la plateforme du SoC, une mémoire telle que la DDR ou la compact Flash, sinon il faut garantir l'accès à une mémoire adéquate, afin de stocker la partie logicielle.

2.3 La reconfiguration dynamique partielle

La reconfiguration dynamique et partielle (RDP) est connue dans le cadre des FPGAs. La RDP, illustrée dans la figure 2.2, désigne la modification de certaines régions de l'FPGA durant l'exécution, sans impacter les autres régions qui peuvent continuer leur exécution. Ceci permet le partage temporel des ressources physiques de cette région reconfigurable afin de supporter plusieurs états mutuellement exclusifs, chacun permettant d'exécuter une ou plusieurs tâches logicielles. On fait appel à la RDP dans plusieurs cas [66][67] : lorsque le système doit s'adapter au changement de l'environnement, ou bien pour répondre aux exigences de qualité de service, les ressources physiques sont limitées, ou encore pour respecter certaines contraintes de consommation d'énergie, etc.

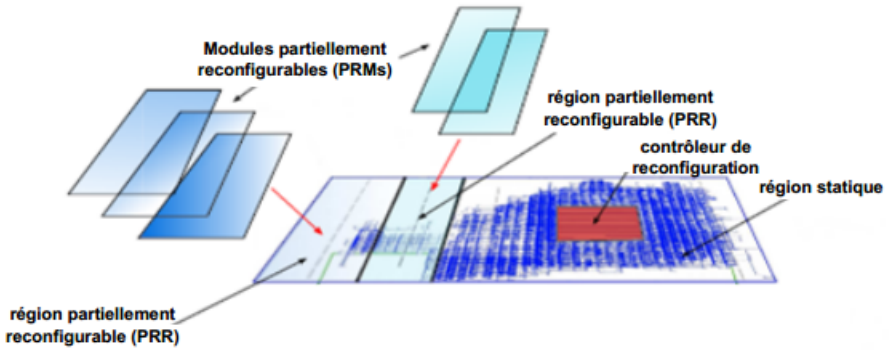


FIGURE 2.2: Vue globale d'un système partiellement reconfigurable

La mise en œuvre des FPGAs a eu lieu à la fin des années 90 avec l'apparition de la failles Virtex du constructeur Xilinx [54]. Ensuite, le fabriquant Altera a mis sur le marché son premier FPGA dynamiquement reconfigurable en 2010, nommé Startix-V [7].

Xilinx est le premier à avoir mis en œuvre cette technologie et a commencé par présenter deux méthodologies de reconfiguration partielle (basée-différence et basée module) [98, 99], suivies de la méthodologie EAPR(Early Access Partial Reconfiguration) en 2006 [101]et de Partition-based Partial Reconfiguration en 2010 [108] .

2.3.1 Le processus de la RDP

Selon le flot modulaire de Xilinx, l'FPGA est divisé en régions ; chaque région ciblant une zone spécifique peut être constituée de CLBs, BRAMs, DSPs, etc. Si une région est dite reconfigurable, alors elle est considérée comme boite noire (*black box*), où seule son interface est visible afin de communiquer via ses ports, avec le reste de la plateforme.

Afin d'effectuer une RDP dans un FPGA, il est nécessaire d'isoler la zone à reconfigurer et charger les bits de configuration (bitstream) correspondants à cette zone. L'outil PARBIT [40] développé par Xilinx permet de transformer les bitstreams afin d'implémenter la reconfiguration partielle.

Pour charger un bitstream sur FPGA, un contrôleur externe ou interne peut être utilisé. Lorsqu'il s'agit d'une reconfiguration partielle externe, on utilise des interfaces telles que JTAG et SelectMap [109]. Dans le cas d'une reconfiguration partielle interne, les FPGAs de Xilinx utilisent l'ICAP *(Internal Configuration Access Port)* [15], qui est un sous-ensemble de l'interface parallèle à 8-bits SelectMAP. Ainsi, l'ICAP permet d'accélérer le processus de reconfiguration dynamique partielle par rapport à l'interface SelectMAP et l'interface série JTAG [89]. L'ICAP est présent presque dans tous les FPGAs Xilinx [14].

2.4 Flot de conception des systèmes sur FPGA

2.4.1 Les systèmes statiques

Le flot de conception d'un système FPGA est décrit principalement selon quatre phases : description logicielle, matérielle, la phase d'association et enfin la simulation du flot.

2.4.1.1 Vue logicielle

Cette partie est déclarée par un fichier nommé MSS (*Microprocessor Software Specification*) qui contient des librairies, des pilotes et parfois des OS utilisés pour les composants matériels. Les bibliothèques logicielles sont générées à l'aide de l'outil LibGen. Généralement, le microprocesseur exécute des applications qui sont écrites en langage C/C++ ou assembleur. Le fichier ELF est le résultat de la compilation.

2.4.1.2 Vue matérielle

Pour cette partie, les outils Xilinx utilisent le fichier MHS (*Microprocessor Hardware Specification*) afin de déclarer les composants à instancier dans le système. La description matérielle est écrite soit en langage matériel (HDL) tels que VHDL ou Verilog, ou bien selon une description schématique. Les étapes qui suivent sont énumérées comme suit :

1. La partie HDL du système ainsi que la description des BRAMs utilisées pour les microprocesseurs (*system.BMM*) sont générées à l'aide de l'outil PlatGen.
2. Les fichiers NGC ou *netlists* sont obtenus suite à la synthèse pour décrire le design au niveau portes logiques
3. L'étape de traduction vient dans la suite du flot afin de fusionner les fichiers NGC, BMM et le fichier de contraintes UCF (*User Constraint File*) pour obtenir à la fin le ficher design NGD.
4. Le mapping vient superposer la logique décrite dans les netlists sur les composants (CLBs, IOBs, etc.) de l'FPGA.

5. Le placement et le routage (PAR) permet de positionner les différents éléments du système et assurer leur interconnexion.
6. à la fin le fichier de configuration (bitstream) est généré.

2.4.1.3 Phase d'association

Lors de cette phase, le bitstream correspondant à la partie matérielle du système est fusionné avec le fichier exécutable du logiciel à l'aide de l'outil BitInit. Cette fusion permettra de décrire le code et les données qui seront stockés dans les BRAMs pour exécuter l'application. Le fichier bitstream résultant *download.bit* peut, par la suite, être chargé sur l'FPGA à partir du PC en utilisant l'outil iMPACT. Le stockage de ce bitstream est possible dans une mémoire externe de l'FPGA ou un compact flash pour qu'il soit chargé automatiquement dès la mise sous tension de l'FPGA.

2.4.1.4 Simulation

La phase de simulation peut être effectuée à deux niveaux différents lors de la conception : avant la synthèse (simulation comportementale) ou après la synthèse (simulation structurelle). La première permet de vérifier le bon fonctionnement du système, le modèle utilisé fait une abstraction des composants du système et contient des opérations à haut niveau. La simulation structurelle intègre plus de détails et consomme donc plus de temps.

2.4.2 Les systèmes reconfigurables

2.4.2.1 Le flot Early Access Partial Reconfiguration

Ce flot de conception a été introduit en 2006 par Xilinx afin de permettre l'utilisation des modules reconfigurables à 2 dimensions. Par définition, un module reconfigurable à 2 dimensions (appelé PRM :Partial Reconfigurable Module) est un module qui peut prendre une forme rectangulaire quelconque. Ce qui fait la différence avec les modules à 1 dimension, qui ne pouvaient occuper que la hauteur d'un FPGA, selon la méthodologie modulaire de la reconfiguration partielle. La méthodologie EAPR permet aussi le passage directe des fils statiques à travers les PRMs sans pour autant utiliser les bus macros [98]. Ceci permet ainsi d'améliorer la performance et simplifier la conception. Parmi les terminologies utilisées dans le jargon de Xilinx, une PRR (Partial Reconfigurable Region) est une région reconfigurable qui peut accueillir un ou plusieurs PRMs. Tous les PRMs d'une PRR ont la même interface externe afin de faciliter la compatibilité et doivent également être prédéterminés. La première étape consiste à charger un bitstream initial sur l'FPGA. Ce bitstream contient la partie statique et les PRMs initiaux des PRRs. Ensuite, au cours d'une reconfiguration partielle, le contrôleur charge un bitstream partiel correspondant à un autre PRM de la même PRR. A l'aide du mécanisme *Read-Modify-Write*, les trames différentes entre les deux PRMs sont modifiées est réécrites dans la mémoire de configuration.

2.4.2.2 *La reconfiguration partielle basée-partition*

L'amélioration du flot de conception de la RDP de Xilinx s'intègre au niveau de la phase de placement du système. Dans le nouveau flot, on ne parle plus de bus macros, mais de broches de partition (*Partition Pins*). La création de ces broches n'estplus manuelle comme c'était le cas des bus macros, mais plutôt de manière automatique pour les ports des partitions reconfigurables. De plus, on ne parle plus de PRR et PRM du flot EAPR, mais de PR et RM. L'implémentation d'un système partiellement reconfigurable dans FPGA est similaire à l'implémentation de plusieurs systèmes statiques qui partagent des parties logiques. Les partitions sont utilisées pour s'assurer que cette partie commune est identique pour tous ces systèmes [106].

2.5 Le besoin d'une nouvelle méthodologie de conception

Les méthodologies de co-conception actuellement utilisées pour concevoir les systèmes électroniques embarqués ne répondent plus aux exigences et aux challenges qu'imposent le développement de systèmes toujours plus complexes. De ce fait, les processus de conception doivent aujourd'hui évoluer pour répondre aux attentes des équipes de conception et donc en améliorer significativement la productivité. Dans [36], les auteurs présentent en quoi consiste la conception des systèmes embarqués et mettent en évidence les challenges à relever.

2.5.1 Les méthodologies de co-conception

Depuis une décennie, les systèmes électroniques embarqués associent de plus en plus des ressources matérielles et logicielles. De ce fait, des méthodologies de conceptions conjointes sont apparues pour prendre en compte les contraintes que cela implique. La figure 2.3 illustre la co-conception selon le modèle en Y : l'application et l'architecture sont conçues d'une manière parallèle et c'est durant la phase l'allocation que les deux peuvent s'entrelacer [25]. Ainsi, l'application peut être conçue indépendamment de l'architecture ce qui permet de réduire notamment le temps de conception.

La phase d'allocation permet de placer les unités de l'application sur les différents types de processeurs. Nous passerons également par cette étape dans notre flot de conception afin de placer des tâches applicatives (IPs) sur des processeurs ou des accélérateurs matériels. L'étape qui suit l'allocation dans le modèle Y est la description électronique du système avant de passer à la vérification et la validation pour étudier les performances du système.

Arrivés à maturité, les environnements de co-conception " traditionnels " sont loin d'avoir résolus tous les challenges imposés par le développement de systèmes complexes. En effet, aujourd'hui les limites des méthodologies de co-conception sont de plus en plus mises-à-jour et il est de plus en plus difficile pour les équipes de conception de relever les défis.

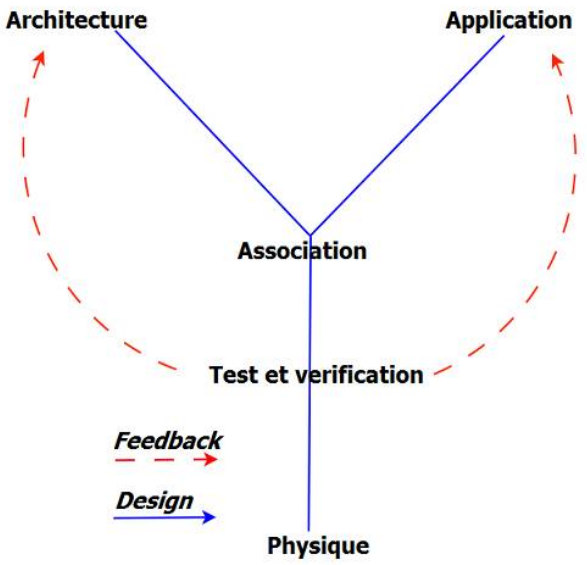


FIGURE 2.3: Flot de co-conception des SoCs selon le modèle Y

2.5.2 Les challenges à relever

Aujourd'hui, les défis que doivent relever les équipes de conception de systèmes embarqués, sont de plus en plus nombreux.

2.5.2.1 L'évolution technologique

Les technologies disponibles pour concevoir des systèmes embarqués ne cessent d'évoluer à toute vitesse. Les équipes de conception doivent sans cesse appréhender ces évolutions. Cependant, ces équipes ne disposent pas de beaucoup de temps pour maîtriser une technologie avant l'arrivée de sa remplaçante.

2.5.2.2 La complexité

Plusieurs facteurs exigent l'intégration de plus de fonctionnalités dans les systèmes embarqués, à savoir l'évolution technologique, la demande et l'exigence du marché, etc. Afin de réaliser ces systèmes, de nombreuses ressource sont nécessaires. Cependant, les technologies utilisées sont assez hétérogènes et leur utilisation nécessite une bonne maîtrise afin de les faire cohabiter dans un même système. Désormais, ce défi s'avère de plus en plus complexe à relever.

2.5.2.3 Le temps de mise sur le marché

Face aux nouveaux besoins de plus en plus exigeants et afin d'être réactifs aux publications de nouveaux standards, les concepteurs des SOCs disposent de très peu de temps pour concevoir, programmer leur système et le publier dans le marché [16].

Cette forte contrainte de temps fait que les ingénieurs n'ont pas la possibilité d'exploiter réellement tous les transistors disponibles aujourd'hui sur la puce. L'exploitation de la plus

grande partie des transistors disponibles sur une puce passe par un outillage qui permet d'augmenter la productivité des concepteurs.

2.5.2.4 Le coût de conception

Le coût de conception des SOCs menace désormais l'avenir du semi-conducteur vu l'augmentation de son coût qui s'élève à des dizaines de millions d'Euros. Quatre-vingt pour-cent (80%) de ces dizaines de millions sont liés à la conception du logiciel du SoC [18]. Il est donc indispensable de proposer de nouvelles techniques et méthodologies de conception pour réduire ce coût.

Une solution à ce problème est d'unifier les représentations d'un SOC pour faciliter la collaboration entre les différents spécialistes intervenants dans sa conception.

2.6 La modélisation à haut niveau

La conception des systèmes embarqués sur FPGA est devenue une tâche coûteuse en temps et nécessite une maîtrise des détails techniques des plateformes ciblées. Ceci est dû à la complexité et la taille continuellement croissantes de ces systèmes, il faut ainsi trouver la solution adéquate pour améliorer la productivité de conception.

L'élévation du niveau d'abstraction est une solution optimale afin d'améliorer la productivité de conception. Cette abstraction permet de cacher les détails de bas-niveau et accélérer la phase de conception en utilisant un mécanisme automatisant le passage d'une modélisation à haut-niveau des systèmes au code permettant l'implémentation physique.

Notre méthodologie sera donc basée sur l'Ingénierie dirigée par les modèles (IDM) qui permettra d'assurer la génération automatique du code à partir de modèle de haut niveau et améliorer ainsi la productivité des concepteurs. La modélisation haut niveau utilise le standard OMG MARTE.

Dans ce qui suit, nous présentons les notions de base de l'IDM et les concepts utilisés dans le profil MARTE.

2.6.1 L'Ingénierie dirigée par les modèles

Modéliser un système revient à le représenter de manière abstraite et simplifiée. Un modèle met en évidence certaines caractéristiques du système en faisant abstraction notamment de ses détails d'implémentation. Parmi les nombreux méthodes et langages qui ont permis par le passé de manipuler les modèles nous citons : Merise [90] (années 1970), SSADM (Structured Systems Analysis and Design Methodology) [32] (1980), UML (Unified Modeling Language) [57] (1995), etc.

L'Ingénierie Dirigée par les Modèles (IDM ou encore MDE pour *Model Driven Engineering*) vient pallier la déficience des méthodes traditionnelles de modélisation.

Le principe de l'IDM consiste à utiliser intensivement et systématiquement les modèles tout au long du processus de développement logiciel. Les modèles devront désormais non seulement

être au cœur même du processus, mais également devenir des entités interprétables par les machines. L'IDM fait sortir ces modèles d'une phase de passivité à une phase de productivité en faisant d'eux l'élément de base du processus de développement.

J. Bézivin [19] a présenté l'analogie avec « tout est objet » des années 80 et vient confirmer que le principe de base de l'IDM consiste à dire que « tout est modèle». Les modèles permettent non seulement de se dégager de certains détails, mais également de réaliser un système complexe par petits blocs plus simples et facilement maîtrisables.

Ainsi, on pourrait utiliser des modèles pour exprimer les concepts spécifiques à un domaine d'application, des modèles pour décrire des aspects technologiques, etc. , chacun de ces modèles étant exprimé dans la notation et/ou le formalisme les plus appropriés.

L'intérêt d'utiliser l'IDM a pris de l'amplification lorsque l'OMG a créé son initiative MDA (Model Driven Architecture). L'MDA a visé l'automatisation de la génération des applications suite à la modification des plateformes cibles, grâce à une séparation entre les parties métier et technique de l'application.

Les trois concepts de base qui permettent de caractériser la méthodologie liée à l'IDM sont les *modèles*, les *méta-modèles* et les *transformations* que nous présentons dans ce qui suit.

2.6.1.1 Modèle

Un modèle correspond à une abstraction de la réalité qui permet de la représenter à l'aide de concepts et de relations entre ces concepts. L'IDM utilise les modèles tout au long du développement logiciel. La notion de modèle n'est pas nouvelle mais elle reflète néanmoins une manière de penser que l'homme utilise ; l'IDM reprend cette manière de penser et l'applique à l'informatique [47].

2.6.1.2 Méta-modèle

Un modèle ; abstraction de la réalité, *est conforme* à un *méta-modèle* qui définit les concepts et leurs relation présents à ce niveau d'abstraction, de manière précise. Un méta-modèle permet donc de représenter des mécanismes complexes faisant intervenir plusieurs concepts. En IDM, un méta-modèle spécifie en quelque sorte la syntaxe des modèles, comme par analogie, on peut dire qu'un langage spécifie sa grammaire.

La figure 2.4 illustre les quatre niveaux traités par l'IDM [19] illustrés par un exemple :

- Le niveau M0 : représente la réalité, il contient les entités à modéliser. Les variables *Numéro* et *Solde* de la figure sont affectées par des valeurs.
- Le niveau M1 : contient les modèles utilisés pour modéliser les entités. Il s'agit du modèle manipulé par les développeurs. Dans l'exemple, le modèle déclare les variables de M0 et identifie la notion *Compte*. Un modèle du niveau M1 est conforme à un méta-modèle du niveau M2.
- Le niveau M2 : les concepts manipulés dans le modèle du niveau M1 sont définis dans ce niveau. *Compte* est une *classe* et les déclarations de variables sont des *Attributs* de cette

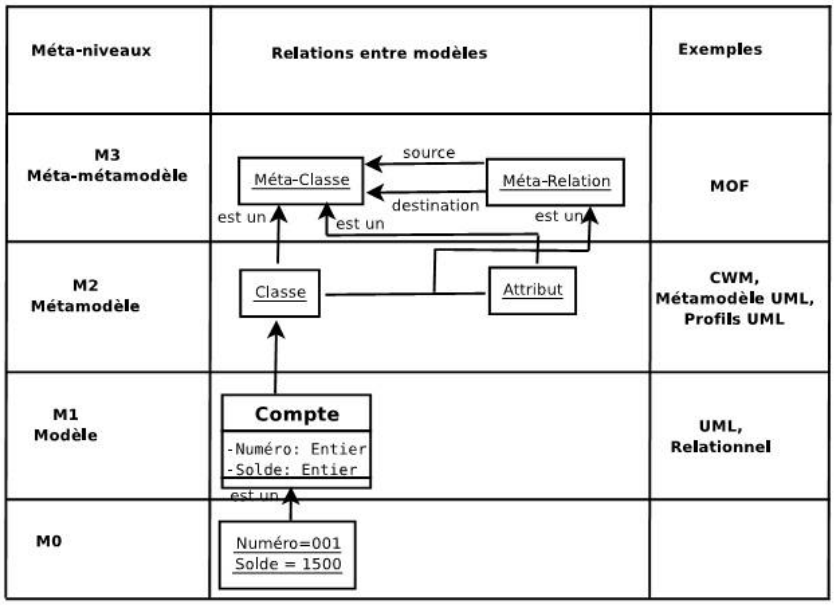


FIGURE 2.4: Les différents niveaux de le modélisation dans l'IDM

classe. Le méta-modèle du niveau M2, qui contient ces concepts (classe et attributs) est conforme au méta-modèle du niveau M3.

– Le niveau M3 : représente le plus haut niveau d'abstraction ; il contient le méta-méta-modèle qui s'auto-définit. Il n'y a pas de niveau supérieur au niveau M3. Le méta-méta-modèle définit par l'OMG est le MOF (*Meta Object Facility*).

2.6.1.3 Transformation de modèles

Le processus de développement basé sur l'IDM commence à un haut niveau d'abstraction et se termine par le modèle de bas niveau ciblé (un modèle exécutable ou un code), en suivant des niveaux d'abstraction intermédiaires à travers les transformations de modèles [82].

La transformation de modèles est un autre point clé de l'IDM car elle permet de passer d'un modèle source décrit à un certain niveau d'abstraction à un modèle destination décrit éventuellement à un autre niveau d'abstraction. Ces modèles source et destination sont conformes à leur méta-modèle respectif et le passage de l'un à l'autre est décrit par des règles de transformation. La transformation de modèles illustrée par la figure 2.5, permet de passer d'un modèle source à un modèle destination par l'exécution de règles de transformation.

Selon T.Mens et al.[55], deux types de transformations de modèles peuvent être réalisés : les transformations endogènes et les transformations exogènes. Une transformation endogène est une transformation de modèles utilisant le même méta-modèle en source et en cible. Ce type de transformation est principalement utilisé pour restructurer le modèle transformé afin de l'optimiser ou de l'adapter à une utilisation particulière (appelé aussi le *refactoring*). Par ailleurs, une transformation exogène est effectuée entre des méta-modèles différents.

Règle de transformation

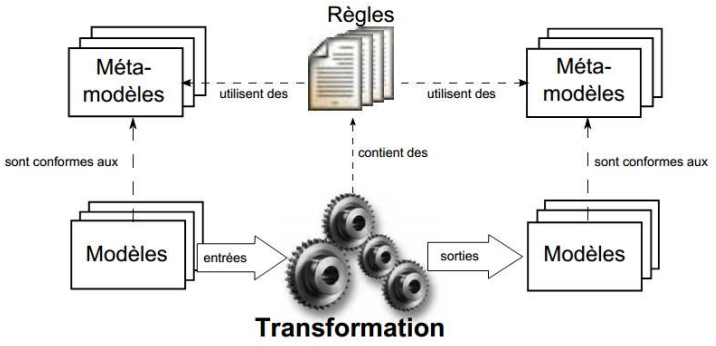


FIGURE 2.5: Transformation de modèles

Une règle de transformation [47] définit la manière dont un ensemble de concepts du méta-modèle source est transformé en un ensemble de concepts du méta-modèle destination. Du point de vue du fonctionnement, une règle peut se décomposer en trois étapes 1) La vérification de la condition correspond à l'analyse du modèle source de manière à détecter la présence d'un ensemble de concepts correspondant à la règle, 2)l'exécution de la règle n'est validée que lorsque ces concepts respectent la condition, 3) la création génère un ensemble de concepts dans le modèle de sortie dont les champs sont remplis par les variables affectées durant l'exécution.

Nous utiliserons les notions de règles de transformations dans le chapitre 6 afin de montrer le passage des modèles de haut niveau d'abstraction vers un code ou un modèle exécutable.

2.6.2 UML pour la conception des SoCs

L'UML (*Unified Modeling Language*) [57] est un langage de modélisation unifié généraliste par opposition aux langages dédiés DSL (Domain Specific Language) [1] tels le VHDL, le System Verilog pour modéliser l'implémentation matérielle ou encore l'AADL [80], standardisé par le SAE, utilisé pour modéliser des systèmes avioniques notamment.

Aujourd'hui, l'UML est largement utilisé dans le monde du génie logiciel car il permet de modéliser une partie du système et de générer la documentation. Cependant, UML n'a été utilisé pour la conception des systèmes embarqués qu'au début des années 2000, et ce grâce aux caractéristiques de ce langage. Les auteurs de [53] et [95], présentent une vue d'ensemble de l'utilisation de l'UML dans le contexte de la co-conception, ainsi que les différentes motivations à cette utilisation.

2.6.2.1 Extensions vers un profil

Bien que développé dans un souci de généricité, l'UML ne permet pas de représenter correctement certains éléments de système dans des domaines métiers spécifiques. Pour cela, un mécanisme d'extension d'UML a été mis en place, sous forme de profil, afin de définir un nouveau méta-modèle le plus souvent adapté à un domaine en particulier comme le sont les DSL

1. DSL :langage de modélisation dédié à un domaine métier. On retrouve notamment le DSL XML IP-XACT développé pour répondre aux besoins de l'électronique

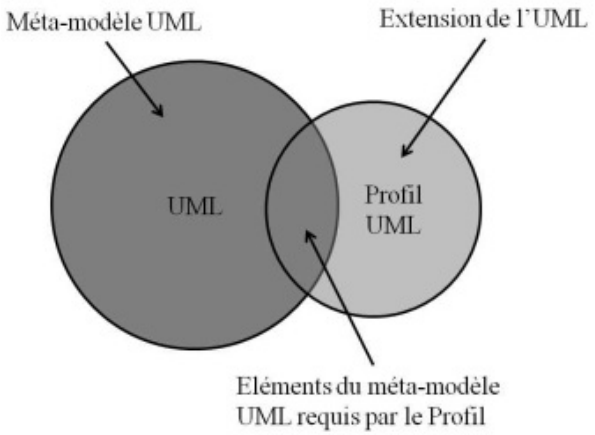


FIGURE 2.6: Extension d'UML à l'aide du mécanisme des profils

. Toutefois, le profil UML, contrairement aux DSL, se base sur tout ou partie du méta-modèle UML avec une extension qui définit de nouveaux éléments comme le montre la figure 2.6.

2.6.2.2 *Les profils UML*

L'OMG a standardisé un certain nombre de profils [50] tels que :

- SysML(System Modeling Language)[59] : profil dédié à la modélisation système qui prend en compte la notion d'exigence.
- UML Profile for QoS, Fault and Tolerance Fault Tolerance Characteristics and Mechanisms [61] :profil qui permet d'ajouter des informations non fonctionnelles sur le modèle afin de caractériser celui-ci en termes de qualité de service. Ce profil ne se suffit pas à lui-même pour modéliser un système ;
- UML Profile for SoC (System on a Chip)[60] :profil dédié à la modélisation de SoC. Il ajoute à l'UML la sémantique nécessaire qui permet notamment de modéliser des modules et des canaux hiérarchiques. Ce profil est très proche des concepts du langage SystemC ;
- MARTE (*Modeling Architecture Real Time Embedded*) [56] : profil qui reprend des éléments du profil SPT (Schedulability, Performance and Time) [58] et ajoute des concepts pour la modélisation de système soumis à des contraintes temps réel. Étant donné que nous utilisons le profil MARTE dans nos travaux, nous les détaillerons dans la section suivante.

2.7 Le profil MARTE

2.7.1 Présentation

Le standard MARTE [56] (Modeling and Analysis of Real-Time and Embedded systems, nommé en français Modélisation et Analyse des Systèmes Temps Réel Embarqués) dont la version préliminaire a été approuvée en juin 2007 devrait apporter une unification « syntaxique » des différentes initiatives.

Le standard OMG MARTE est un profil UML qui définit les fondations de la modélisation des SETRs (Systèmes Embarqués Temps Réels) et est conçu pour succéder à son prédécesseur

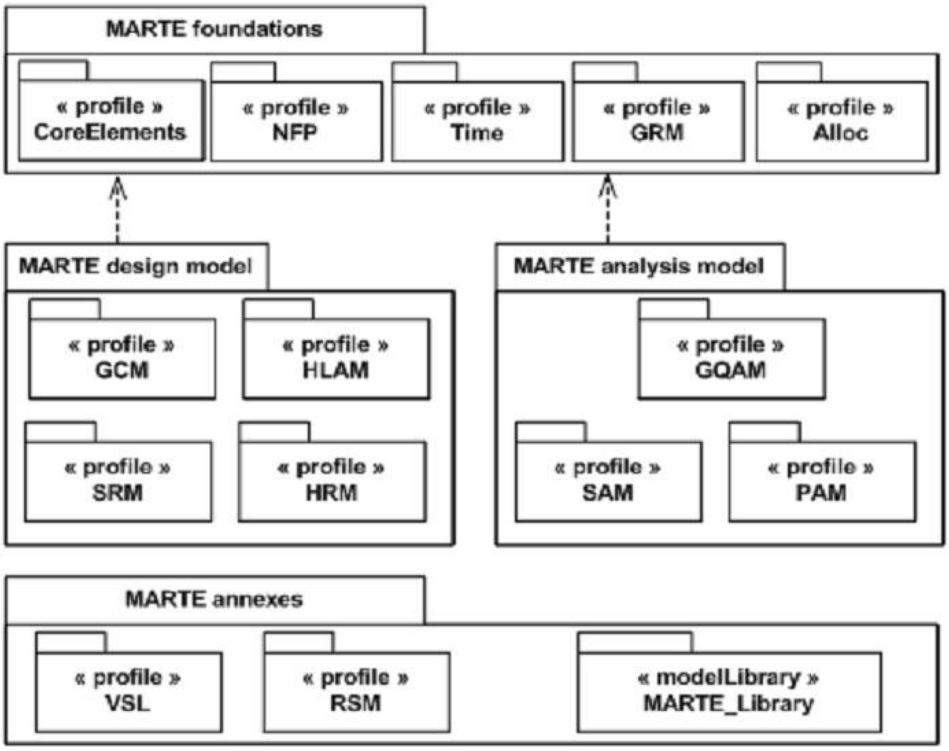


FIGURE 2.7: Vue globale de la structure du profil MARTE

appelé SPT (profile for Scheduling, Performance and Time Specification) [58]. En effet, il présente de nombreux concepts permettant d'annoter un système logiciel ou matériel en vue de l'analyser. L'objectif n'est pas de définir de nouvelles méthodologies de conception ou de nouvelles techniques d'analyse des SETRs, mais il s'organise en différents paquetages afin de les soutenir par une riche base d'annotations. La dernière version de la specification de MARTE 1.1 date de Juin 2011.

2.7.2 Structure du profil

Le profil traite deux axes principaux : l'un pour modéliser les caractéristiques des systèmes temps réel et embarqués et l'autre pour annoter des modèles d'application afin de faciliter l'analyse des propriétés système.

L'architecture du profil est composée de quatre paquetages comme le montre la figure 2.7 : *foundations*, *design model*, *analysis model* et *annexes*. Étant donné que MARTE s'intéresse à la modélisation et l'analyse des systèmes temps réel et embarqués, il utilise pour ce faire, les paquetages *design model* et *analysis model*. Les éléments partagés par ces deux paquetages sont regroupés dans le paquetage *foundations*. Ces paquetages sont détaillés brièvement dans la suite.

2.7.2.1 Foundations

C'est le cœur du profil, sur lequel les autres paquetages s'appuient. Il contient entre autres une définition précises des différentes notions de temps que l'on retrouve dans les systèmes

temps réel. Le sous-paquetage *CoreElements* contient les éléments de base de tout modèle MARTE tels que ModelElement, Classifier, Instance, etc. Il contient aussi les éléments liés au comportement (Behavior, Action, Event, etc). Quant au sous-paquetage *Non Functional Properties*(NFP) , il fournit un ensemble de notions pour exprimer différentes grandeurs comme par exemple le temps, la fréquence, la puissance, incluant les unités de mesure correspondantes. Le sous-paquetage *Time* contient les notions liées au temps. Le sous-paquetage *Generic Resource Modeling* (GRM) permet de modéliser des ressources d'une façon générique à travers des concepts tels que les ressources de calculs, de stockage, de synchronisation, etc. Le sous-paquetage *Allocation* (Alloc) introduit les concepts pouvant être utilisés pour la spécification d'une projection d'un modèle fonctionnel sur une architecture matérielle.

Dans ce manuscrit, nous détaillerons les sous-paquetages *CoreElements* et *Allocation* dans le chapitre 5 pour la modélisation respective du comportement et de la distribution spatiale et temporelle de l'application sur l'architecture.

2.7.2.2 Design Model

Ce paquetage contient les sous-paquetages GCM, HLAM, SRM et HRM. Le premier sous-paquetage *Generic Component Modeling* (GCM) contient les concepts liés à la modélisation générique des composants structurés, les ports de flots, les ports de messages, etc. Le sous-paquetage *High-Level Application Modeling* (HLAM) contient les notions de temps-réel tels que les services temps-réel, les stratégies d'ordonnancement, etc. Ensuite, le sous-paquetage *Software Resource Modeling*(SRM) permet de modéliser les ressources logicielles et de décrire des API pour un système d'exploitation temps-réel . Le sous-paquetage *Hardware Resource Modeling* (HRM) permet de modéliser les ressources matérielles telles que le processeur, la mémoire, etc.

Dans nos travaux, nous ferons appel à ces sous-paquetages pour la modélisation des FPGAs et des applications traitées dans notre étude de cas.

2.7.2.3 Analysis Model

Ce dernier paquetage se focalise sur les activités d'analyse des systèmes embarqués. Il identifie deux sous-activités particulières à savoir l'ordonnancement et la performance. Le sous-paquetage principal *Generic Quantitative Analysis Modeling* (GQAM) permet l'analyse quantitative liée à des domaines tels que la performance, l'ordonnancement, la puissance, la mémoire, la sécurité, etc. Deux autres sous-paquetages héritent du GQAM : *Schedulability Analysis Modeling* (SAM) qui contient les notions spécifiques à l'analyse des scénarios d'ordonnancement ; ensuite le *Performance Analysis Modeling* (PAM) qui contient les concepts spécifiques à l'analyse des performances et des propriétés temporelles.

2.7.2.4 Annexes

Le paquetage *annexes* regroupe les profils et libraires prédéfinies permettant au concepteur d'ajouter des informations supplémentaires à ses modèles. On distingue le langage *Value Specifi-*

cation Language (VSL) dédié à la spécification de propriétés non fonctionnelles (NFP), permettant la description de types, paramètres, constantes, énumérations et expressions. Ensuite, le sous-paquetage *Repetitive Structure Modeling* (RSM) représente une sémantique opérationnelle pour la conception d'applications à traitements de données en parallèle. Enfin, la librairie *MARTE Library* qui contient les types primitifs employés dans MARTE, des types de données génériques et une librairie pour la notion de temps.

2.8 Conclusion

Dans ce chapitre, nous avons présenté trois grands axes sur lesquels se basent nos travaux de thèse à savoir :

- Les systèmes dynamiquement reconfigurables : nous avons introduit la notion de la reconfiguration dynamique et partielle en particulier dans les FPGAs et avons défini le flot de conception de ces systèmes selon les fabricants industriels.
- Le besoin des méthodologies de conception en particulier avec la complexité des systèmes et des technologies et nous avons traité en particulier les méthodologies de conception haut niveau ; l'IDM et les profil UML pour la conception des SoCs.
- Le standard OMG MARTE sur lequel se basent nos contributions a été présenté et sera également détaillé au fur et à mesure dans le chapitre 5.

Dans le prochain chapitre, nous présenterons les différents travaux qui traitent la conception et la modélisation des systèmes dynamiquement reconfigurables. Une discussion sera également présentée afin de positionner notre contribution par rapport à ces travaux.

CHAPITRE 3

ÉTAT DE L'ART

3.1 Introduction

Face à l'évolution technologique continue des systèmes embarqués ainsi que la complexité croissante des applications qu'ils ciblent, la conception de tels systèmes est devenue une tâche

complexe et coûteuse en temps. Cette complexité est due au fait que les outils de conception n'évoluent pas au même rythme que la technologie matérielle.

En outre, la reconfigurabilité de certains systèmes augmente encore plus cette complexité d'où le besoin d'adopter une méthodologie efficace de conception afin d'assurer les facteurs de flexibilité, réutilisabilité et automatisation. Ceci permettra de faciliter le travail des concepteurs et d'améliorer leur productivité.

Afin de répondre à ces objectifs, nous étudions dans ce chapitre, les différentes approches reliées à nos travaux.

- Nous présentons tout d'abord, les différents profils UML utilisés qui permettent la conception des SoCs et nous discutons le positionnement de MARTE par rapport à eux.
- Nous soulignons ensuite, les méthodologies les plus intéressantes qui ont fait usage de l'IDM pour la conception conjointe des SoCs.
- L'approche que nous proposons définit une méthodologie pour la conception des systèmes dynamiquement reconfigurables, en particulier les FPGAs. Ainsi, nous étudions dans les sections qui suivent, les différents projets de la littérature qui ont traité 1) la modélisation haut-niveau de la reconfiguration dynamique sur les FPGAs, des aspects particuliers de la RDP concernant 2) le mécanisme du contrôle de la reconfiguration 3) la notion du déploiement des IPs moyennant des standards tels que IP-XACT 4) Modélisation de l'architecture physique des FPGAs.

3.2 Les profils UML pour la modélisation des SoCs

3.2.1 UML for SoC

Ce profil est développé par *Fujitsu Limited* et *Fujitsu Laboratories*. Il vise à décrire les informations spécifiques des SoCs en utilisant UML. Il intègre les concepts du SoC et permet la génération automatique de code pour le matériel (par exemple, SystemC), tout en couvrant les niveaux d'abstraction de modélisation de niveau transactionnel (TLM) jusqu'au Register Transfer Level (RTL). UML-SoC se focalise sur le diagramme de structure UML 2.0. Il propose des stéréotypes qui permettent la modélisation structurelle, la modélisation de communication, la modélisation des opérations et des propriétés. Le tableau 3.1 résume la correspondance entre certains stéréotypes SoC et les concepts UML. La motivation pour le profil, c'est que UML définit plusieurs types de diagrammes, mais ne décrit pas comment les utiliser.

Ainsi, les décisions concernant la partie de la spécification à modéliser, les diagrammes à utiliser ainsi que la manière de modéliser la spécification avec des diagrammes différents, doivent être effectuées. Dans cette approche, UML est utilisé comme un modèle formel pour la spécification de la conception des SoCs afin de permettre la validation de la cohérence de la spécification (Cf. figure 3.1).

Par conséquent, l'implémentation du SoC est validée par un calcul systématique des scénarios de test à partir du modèle UML. En outre, UML est intégré dans le processus de vérification sans modifier le style de conception en cours. Seuls les diagrammes de cas d'utilisation, les

TABLE 3.1: Ensemble de stéréotypes utilisés dans le profil UML for SoC [60]

Élément du modèle SoC	Stéréotype	Méta-classe UML
Module	SocModule	class
Process	SoCProcess	Operation
Data	Data	Class
Controller	Controller	class
Protocol Interface	SoCInterface	Interface
Channel	SoCChannel	Class
Protocol	SoCProtocol	Collaboration
Port	SoCPort	Port/Class
Module Part	SoCModuleProperty	Property
Channel Part	SoCChannelProperty	Property

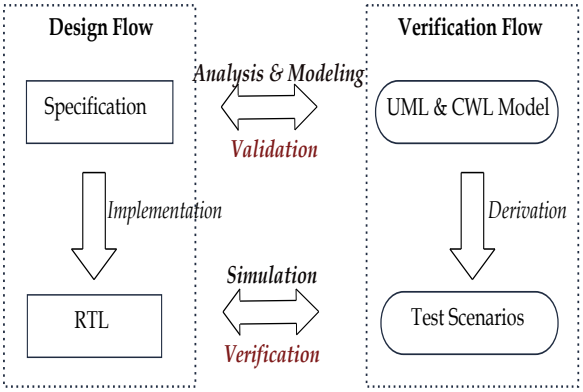


FIGURE 3.1: Flot de conception du profil UML for SoC [110]

diagrammes de séquence et les diagrammes de classes UML sont utilisés dans la modélisation des fonctions, des types de données (Data Type) et des comportements de la spécification. Les interfaces des SoCs ne peuvent pas être modélisées simplement par des opérations et des méthodes. Pour ce faire, un langage spécifique dit *Component Wrapper Language* (CWL) est utilisé comme langage de spécification formelle des interfaces afin de modéliser la spécification des changements des signaux dans les ports d'entrée/sortie [110].

3.2.2 UML for SystemC

Ce profil est développé par l'Université de Catane et STMicroelectronics [74]. Il prend à la fois les avantages du profil UML 2.0 et du langage SystemC tout en suivant les principes de MDA. Le langage SystemC est bien adapté pour l'implémentation des modèles UML, car il supporte le paradigme orienté-objet et peut représenter uniformément le matériel et le logiciel dans un seul langage. En outre, comme UML, SystemC devient le langage standard pour la conception des SoCs au niveau système. Selon [74], UML peut améliorer le flot de conception du SoC en trois façons :

– UML à la manière plateforme indépendante : peut être adopté au niveau du modèle

fonctionnel exécutable du système afin de décrire la spécification.

- UML pour SystemC : peut être utilisé pour la description du matériel dans les couches d'abstraction supérieures à la couche RTL.
- Les profils UML conçus pour les langages de programmation tels que C/C + +, Java, etc. : peuvent être utilisés putôt pour les parties logicielles.

Le profil UML for SystemC reflète à la fois la structure et les caractéristiques comportementales du langage SystemC et permet la modélisation de haut niveau des SoCs avec une traduction directe au code SystemC. Il est basé sur deux diagrammes : le diagramme de classes pour décrire la structure du modèle et les diagrammes d'états pour décrire le comportement du modèle. Les stéréotypes les plus importants utilisés dans les différents diagrammes de structure UML représentent les blocs de construction structurel de SystemC tels que le *module*, le *port*, l'*interface*, le processus *thread*, l'événement, etc. La figure 3.2 résume la correspondance entre SystemC et les concepts UML. Le profil proposé est censé bénéficier fortement de la portabilité, l'échange et la réutilisation des IPs.

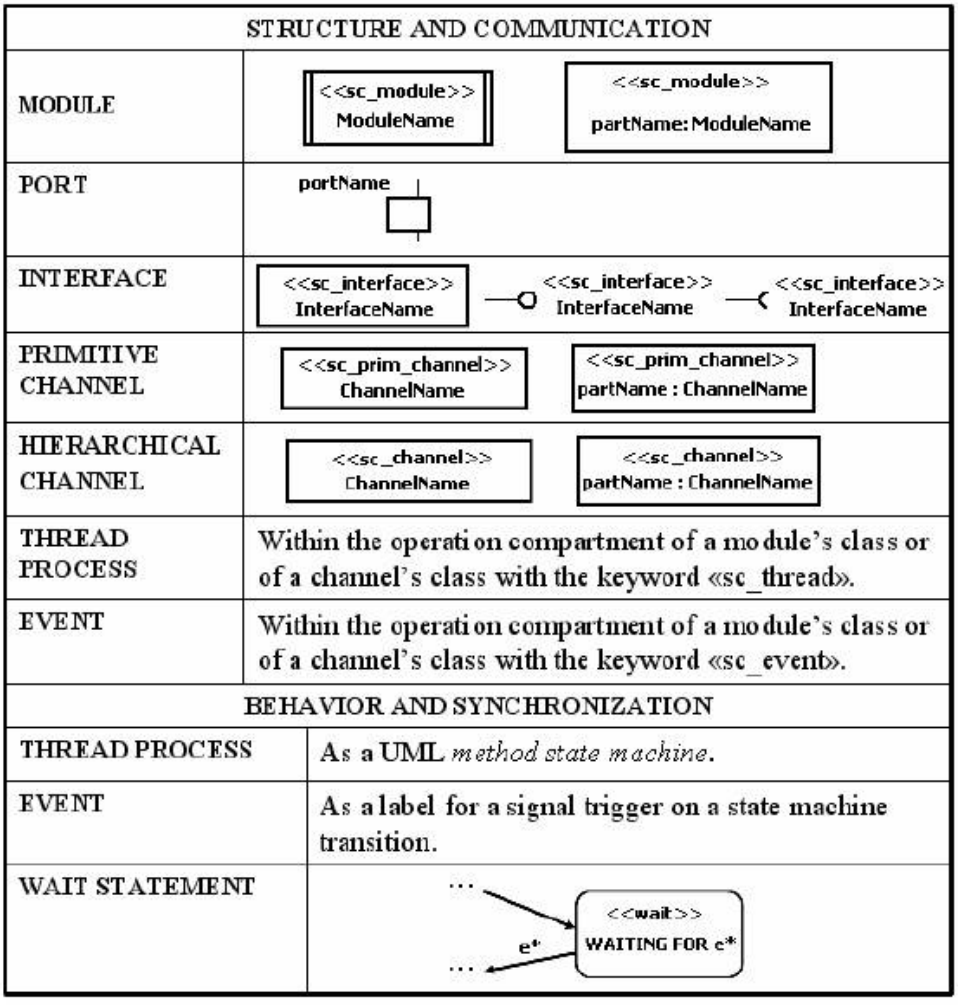


FIGURE 3.2: Notation UML des concepts SystemC

3.2.3 SysML

Le langage SysML (System Modeling Language) [59] est le premier standard pour l'ingénierie système proposé par l'OMG, qui vise à décrire des systèmes complexes. SysML est une extension d'un sous-ensemble UML, où les classes, les composants, les structures composites, les activités, les machines d'état et les cas d'utilisation sont considérés comme des bases. Mais ils ont améliorés avec certains paramètres, des exigences et des concepts d'allocation. La figure 3.3 résume les différents diagrammes utilisés dans SysML.

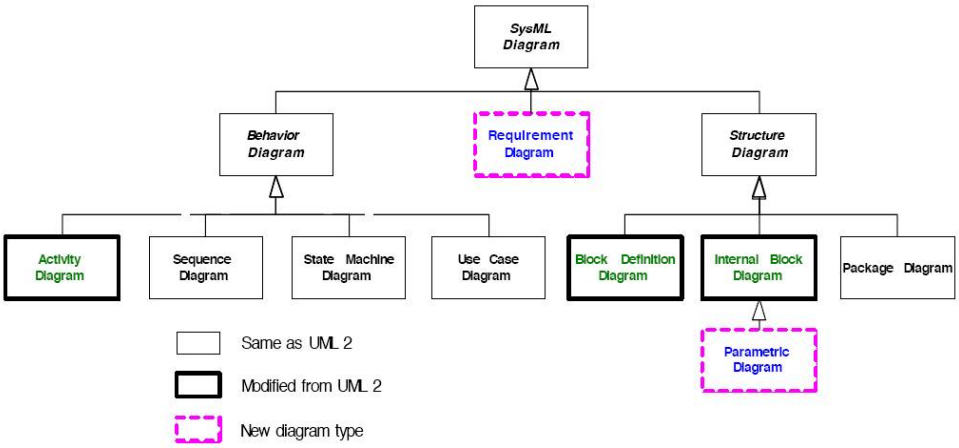


FIGURE 3.3: Les diagrammes utilisés dans le profil SysML

De plus, SysML permet la description des exigences de traçabilité et fournit les moyens pour exprimer le comportement et la composition des blocs de système. De plus, ce profil fournit également au concepteur des formalismes paramétrés utilisés pour exprimer des modèles analytiques basés sur des équations. Les contributions clés de ce standard sont les suivantes :

- *Organisation d'architecture* : les concepts de modélisation liés à exprimer les aspects architecturaux, les concepts de vue (*view*) et point de vue (*Viewpoint*) sont les plus importants
- *Blocs et Flots* : les blocs permettent de représenter des systèmes complexes de façon composée. Les flots de SysML permettent la modélisation de données/flots de contrôle ainsi que les flots physiques, tels que les flots électriques.
- *Comportement* : SysML affine les concepts de comportement communs de UML (comme les machines d'état, activités entre autres) pour la modélisation de systèmes continus.
- *Exigences* : les exigences du système peuvent également être modélisées via SysML. Ces exigences peuvent être présentées soit sous forme de graphiques ou de tableaux moyennant le modèle de traçabilité.
- *Paramètres* : SysML permet aux concepteurs de décrire, sous forme de graphiques, les relations analytiques ainsi que les contraintes.

Cependant, alors que SysML est utilisé par la communauté des RTES (Real-Time Embedded System) pour la conception du SoC, il n'était pas créé principalement, pour la modélisation de la conception des systèmes embarqués. Certes, les propriétés non fonctionnelles telles que les contraintes de temps, la latence et le débit, qui sont des notions cruciales pour la conception de

RTES sont absentes dans ce profil. Ce qui n'est pas le cas du profil UML MARTE.

3.2.4 Discussion

Les divers travaux présentés dans la section précédente montrent la diversité des besoins des concepteurs en matière de sémantique afin d'exprimer toute la complexité d'un système embarqué temps réel et de son fonctionnement. Notons que les systèmes à concevoir sont très variés de point de vue les types d'applications à développer ou encore le choix des architectures matérielles à utiliser.

La solution d'utiliser le langage UML a permis de converger vers un unique langage de représentation graphique de ces systèmes. En d'autres termes, la représentation visuelle proposée par ce langage permet de limitée les erreurs d'interprétation et facilite, en général, la compréhension. Cependant, il faudrait se mettre d'accord sur ce qu'il est utile de représenter et sur la manière de le représenter. L'évolution des besoins, induite par l'évolution technologique, explique la variété des profils UML utilisés dans la conception des systèmes électroniques. Ainsi, se présente le profil MARTE pour remplacer le profil SPT ainsi que le profil QoS. De plus, il pourrait intégrer le profil SoC, afin de ne pas avoir une redondance dans les concepts offerts par ces nombreux profils. Cependant, il est très difficile de tout pouvoir exprimer à partir d'un seul modèle car il peut y avoir des exigences et des configurations particulières qui nécessitent le besoin d'enrichir la sémantique existante.

Les profils UML for Soc [60] et UML for SystemC [74] permettent la modélisation des systèmes embarqués en utilisant des concepts de bas niveau tels que les concepts Clock, Module, Channel qui sont principalement des concepts du SystemC. Le fait d'être étroitement liés aux détails d'implémentation de bas niveau est un inconvénient majeur pour ces profils d'autant plus qu'ils manquent de niveaux d'abstraction nécessaire pour la conception des systèmes complexes.

Par ailleurs, il est à noter la complémentarité entre le profil MARTE et SysML. En effet, SysML permet la modélisation des exigences lors des premières phases de conception tandis que MARTE définit les concepts pour la modélisation du temps ainsi que les aspects non-fonctionnels et s'avère plus approprié pour les phases de conception ultérieures. De plus, SysML permet la description des exigences de traçabilité et fournit les moyens d'exprimer le comportement et la composition des blocs. Plus de détails sur cette comparaison sont présentés dans [31].

En comparant à ces profils, le profil MARTE se présente comme l'un des profils qui offre des moyens de modélisation à un haut niveau d'abstraction et plus de généricité grâce à l'empaquetage de ses concepts.

3.3 L'IDM pour la co-conception des SoCs

Depuis le début des années 2000, les systèmes sur puce ont émergé comme un nouveau paradigme pour les systèmes embarqués. Un SoC peut contenir plusieurs éléments tels que les processeurs, les mémoires, les périphériques d'entrée/sortie, etc. La conception des SoCs peut

être vue de plusieurs côtés, y compris la modélisation par l'agrégation des éléments fonctionnels, la vérification du système et l'implémentation sur un ASIC ou un FPGA . Suite à ces aspects, la complexité de conception des SoCs devient difficile à gérer.

L'unification d'un langage de modélisation de haut niveau s'avère la meilleure solution facilitant la construction d'un système plus cohérent. Les méthodologies de co-conception tirent profit de cette unification afin de construire des modèles mixtes sûrs et facilement compréhensibles.

3.3.1 Conception conjointe logicielle/matérielle

Cette méthodologie consiste à choisir une implémentation pour chaque tâche de l'application, qui peut être logicielle ou matérielle. Une tâche logicielle est écrite en un langage C/C++ et est implémentée sur des processeurs. Ainsi, un système conçu totalement en logiciel est dédié à des exécutions sur des processeurs, ce qui diminue sa vitesse. Tandis qu'une tâche matérielle est décrite dans un langage HDL afin d'être implémentée sur un accélérateur matériel ou un FPGA ce qui conduit à des coûts de fabrication très élevés. Ce compromis est difficile à gérer, d'autant plus qu'il peut faire intervenir plusieurs facteurs tels que les exigences des utilisateurs, le temps de mise sur le marché, etc. D'où la naissance de la vision basée sur une conception mixte logicielle/matérielle définie dans plusieurs travaux de recherche.

Selon certains travaux, le processus de conception conjointe s'appuie sur une certain nombre de phases, illustrées par la figure 3.4 [20].

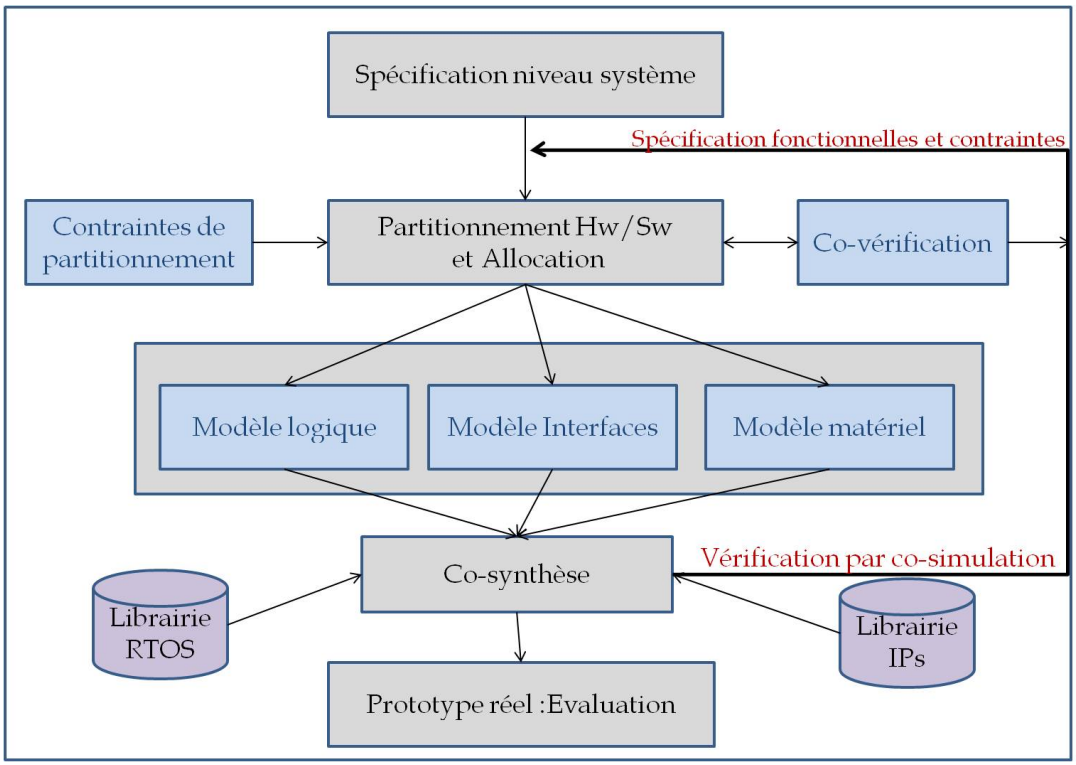


FIGURE 3.4: Démarche de Co-Design

A partir d'une spécification mixte décrivant le système à concevoir dans sa totalité, l'étape du partitionnement vient chercher une répartition adéquate entre le hardware et le software pour

chaque tâche du système. Le but est de trouver un compromis qui respectent certaines contraintes du système pouvant être de type fonctionnel ou non fonctionnel telles que le temps d'exécution, l'espace mémoire consommé, le coût de développement, le degré du parallélisme, etc. Avant d'entamer la phase suivante, le système partitionné doit être vérifié et validé. Tant qu'une solution judicieuse n'ait pas été obtenue, il est nécessaire de renvoyer des lignes de retours. La phase de simulation permet d'effectuer un choix des composants pour l'implémentation du matériel et la synthèse du logiciel. Par ailleurs, l'aspect communication et la synchronisation entre les différents éléments qui forment le système est un aspect non négligé et doit être pris en considération. Autrement, il y a risque de goulot d'étranglement, d'où l'importance de la phase de la synthèse de communication. Après la phase de simulation et de synthèse, le système présente ainsi le fonctionnement souhaité et répond aux exigences et aux contraintes spécifiques de temps réel. Enfin, la phase de prototypage physique est la dernière phase qui permet d'évaluer le système sur une architecture cible choisie.

3.3.2 Gaspard2 : environnement pour la Co-modélisation

GASPARD2 (Graphical Array Specification for Parallel and Distributed Computing) [33][26] est un framework orienté IDM pour la co-conception des SoCs. Il utilise un ensemble de concepts du profil MARTE actuellement supporté par l'industrie du SoC. Ici, un framework signifie un environnement qui fournit aux concepteurs : 1) un formalisme pour la description des systèmes embarqués à un haut niveau d'abstraction, 2) une méthodologie couvrant toutes les étapes de la conception d'un système et 3) un ensemble d'outils qui prend en charge toute l'activité complète de la conception.

GASPARD2 fournit un environnement de développement (*Integrated Development Environment* (IDE)) dédié à la co-modélisation visuelle des systèmes embarqués et a été développé au sein de l'équipe de projet DaRT à INRIA Lille-Nord Europe. Le framework permet la conception rapide et la génération de code moyennant les outils graphiques de UML (e.g., MagicDraw UML [1] et Papyrus [2]) ainsi que l'environnement EMF d'Eclipse.

La méthodologie de GASPARD2 vise à générer, à partir d'une modélisation en MARTE, différents codes afin d'entamer la phase d'exploration du SoC en simulant, de vérifier et de synthétiser. GASPARD2 cible différents langages, tels que le VHDL qui permet la synthèse d'accélérateur matériel, le SystemC permettant la simulation, etc.

La figure 3.5 illustre le flot de conception des systèmes embarqués dans GASPARD2 qui suit les étapes suivantes :

– La modélisation du système : les trois premiers modèles modélisés dans GASPARD2 sont basés sur le profil MARTE. Le modèle d'application décrit les tâches logicielles ou matérielles. Le modèle de l'architecture décrit l'architecture sur laquelle l'application va être exécutée. Le modèle d'allocation est utilisé afin de faire le lien entre les deux modèles. Ensuite, le modèle de déploiement, basé sur le profil GASPARD2, vient relier chaque

1. www.magicdraw.com/
2. www.papyrusuml.org/

composant élémentaire (de l'application ou de l'architecture) à un code qui existe déjà dans une bibliothèque d'IP.

- Les transformations de modèles : organisées en chaînes de transformations qui peuvent partager les transformations liées à des notions communes entre les cibles
- La génération de code : A la fin d'une chaîne de transformations, on obtient un modèle PSM (Platform Specific Model) avec des détails techniques permettant la génération du code lié à la technologie cible. GASPARD2 permet la génération de code ciblant différents langages tels que Fortran, Lustre, SystemC, OpenCL, C et VHDL.

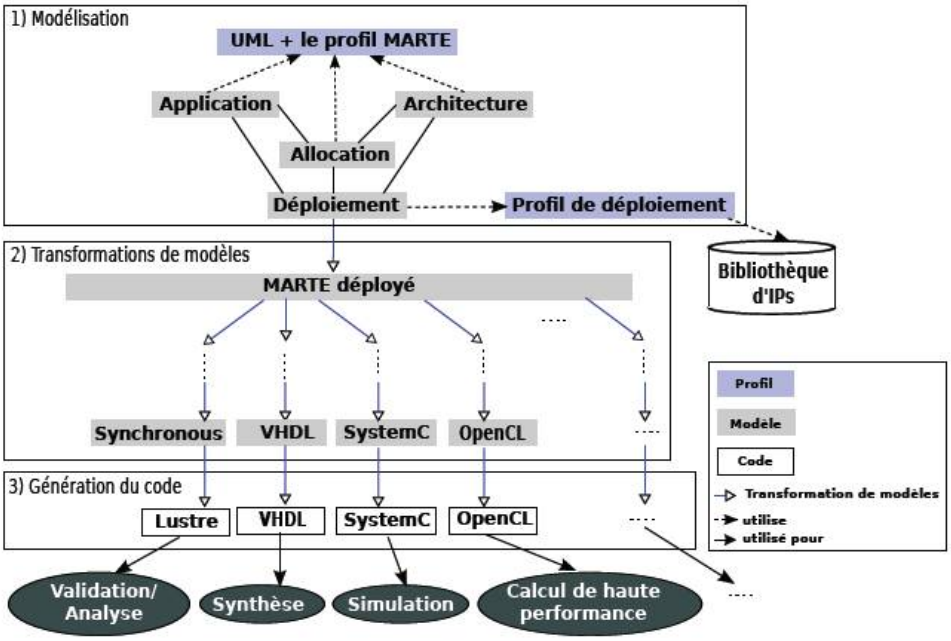


FIGURE 3.5: Le flot de conception des systèmes embarqués dans Gaspard2

3.3.3 Méthodologie de Wang

Les auteurs dans [96] définissent un flot de développement qui vise à générer du code en SystemC à partir des modèles UML. La figure 3.6 montre le flot de conception de cette approche. Le modèle PIM (*Platform Independent Model*) en UML est construit à partir des exigences du système. Le modèle PSM (*Platform-specific model* est obtenu grâce à la transformation de modèles, aux raffinements et les annotations. Le code SystemC est généré à partir du modèle PSM, où la simulation est effectuée afin de vérifier l'exactitude du système. Ensuite, l'implémentation est effectuée à partir du code SystemC généré.

Afin de pouvoir générer correctement le code SystemC, les auteurs définissent également un profil UML pour capturer des concepts spécifiques du SystemC.

3.3.4 Méthodologie UPES

La méthodologie UPES (*Unified Process for Embedded Systems*) définit un ensemble d'éléments afin de guider la conception vers un processus de développement de système embarqué. Tout

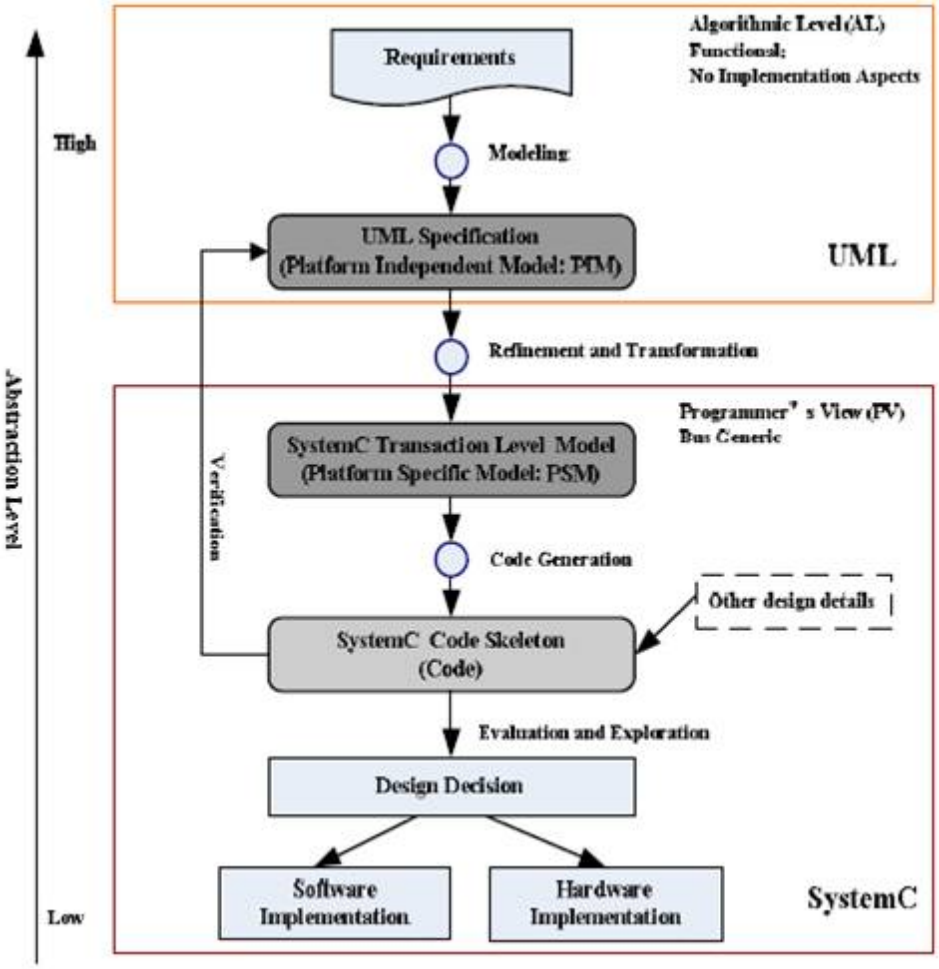


FIGURE 3.6: Le flot de conception des SoCs basé sur la MDA dans l'approche Wang

d'abord, les auteurs dans [77] ont défini le profil UML for SystemC à partir duquel le code SystemC pour le niveau TLM (*Transaction Level Modeling*) peut être généré. Dans [76], les auteurs améliorent la méthodologie précédente en définissant un processus unifié (*Unified Process* : UP) qui utilise le profil SystemC pour la modélisation des SoCs et la génération du code SystemC. En effet, l'utilisation d'un processus pour guider le concepteur dans le développement d'un système accélère le temps de conception.

La figure 3.7 illustre le flot de conception du processus précité. Le système peut être spécifié à l'aide du profil UML (ou tout autre profil UML tel que SysML [59] ou MARTE [56]), il est ensuite mappé (alloué) à une plateforme matérielle spécifiée à un haut niveau. Suite à la phase du mapping, deux flots indépendants sont développés au moyen de modèles et de transformations : le flot logiciel et le flot matériel. Le flot matériel utilise le profil SystemC et atteint le niveau RTL à partir d'une spécification haut niveau de la plateforme, moyennant des transformations de modèles, définies dans [77].

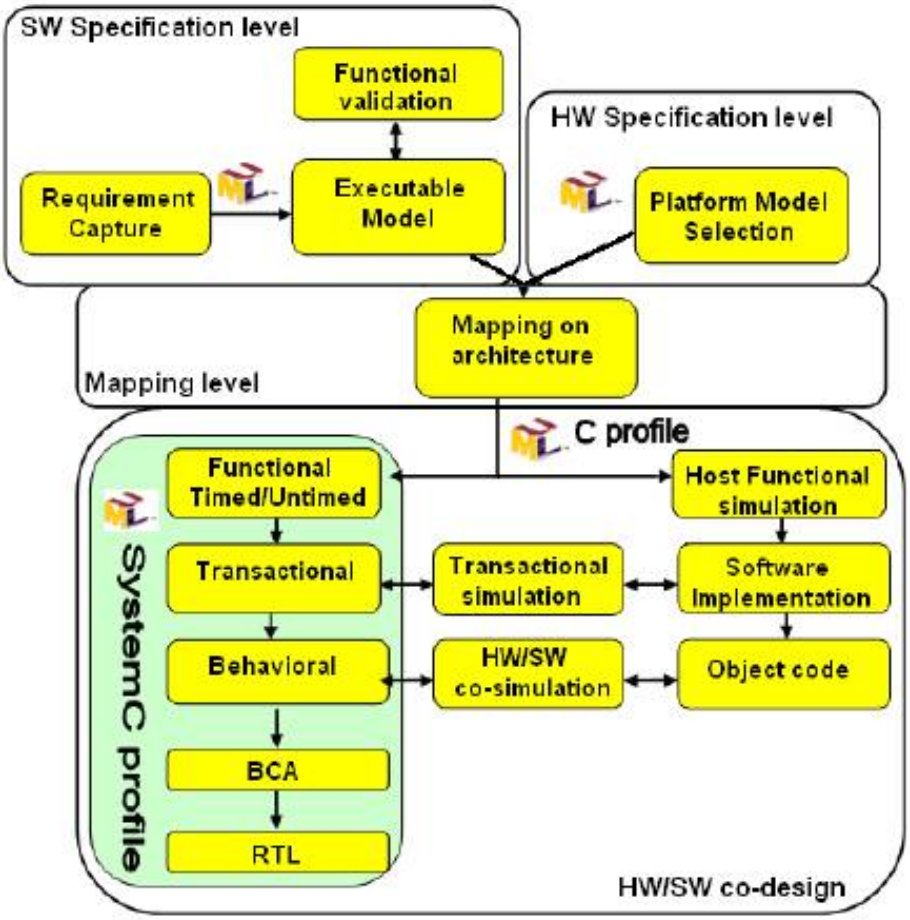


FIGURE 3.7: Le flot de conception de la méthodologie UPES

3.3.5 Méthodologie du projet A3S

La méthodologie A3S [79] est basée sur le langage UML et suit une approche MDA. Elle propose aux concepteurs d'applications Radio Logicielle en particulier, un environnement UML qui permet de tester différents scénarios d'implantation de composants de l'application sur des modèles de plates-formes matérielles.

Cette méthodologie suppose que le concepteur de SoC réutilise l'existant avec des IPs matériels et logiciels afin de réaliser ses systèmes. Dès lors, la fonctionnalité même d'un composant IP n'est plus à démontrer, car elle est assurée par le vendeur de l'IP. De même, le concepteur dispose de toutes les caractéristiques de performances, de surface et de consommation propre à chacune des IP qu'il détient. La figure 3.8 représente le flot de conception A3S, basé sur ces considérations et propose une modélisation et une spécification au niveau système en quatre étapes :

– *Modélisation de l'application* :la modélisation réalisée est indépendante de la plateforme matérielle (PIM). Le concepteur peut donc modéliser son (ses) application(s) de manière indépendante de toute cible architecturale par l'assemblage de composants logiciels "génériques" sous la forme d'un graphe de tâches.

- *Modélisation de l'architecture* : Le concepteur a accès à une bibliothèque de composants dit "matériels" qui lui propose un ensemble de composants matériels qu'il peut utiliser pour créer son architecture. Ces différents composants sont paramétrables en fonction des configurations d'utilisation requises par l'architecte (exemple : paramétrer la fréquence de fonctionnement de certains composants (DSP, FPGA) ou dimensionner la taille des mémoires en fonction de ses contraintes, etc.).
- *Déploiement* : à ce niveau, l'application logicielle et l'architecture matérielle sont modélisées, ce qui permet au concepteur alors de décider des choix d'implantation à mettre en oeuvre et à tester pour son système. Chacune des fonctions du graphe de tâches peut être réalisée en matériel ou logiciel selon ses choix, ce qui engendre une dépendance avec la modélisation de l'architecture (PSM). Par conséquent, certaines contraintes doivent être spécifiées quant aux choix architecturaux pour chaque composant logiciel instancié.
- *Vérifications et Analyse* : les vérifications sont effectuées lors de chacune des étapes précédentes afin de valider les modèles réalisés. Elles permettent d'identifier les éventuelles erreurs et de vérifier le respect des contraintes d'utilisation des composants matériels afin de garantir la cohérence des choix. Dès lors, les analyses de faisabilité et d'estimation de performances peuvent être effectuées. Elles déterminent la faisabilité de l'implantation et renvoient les résultats d'ordonnançabilité et de performance.

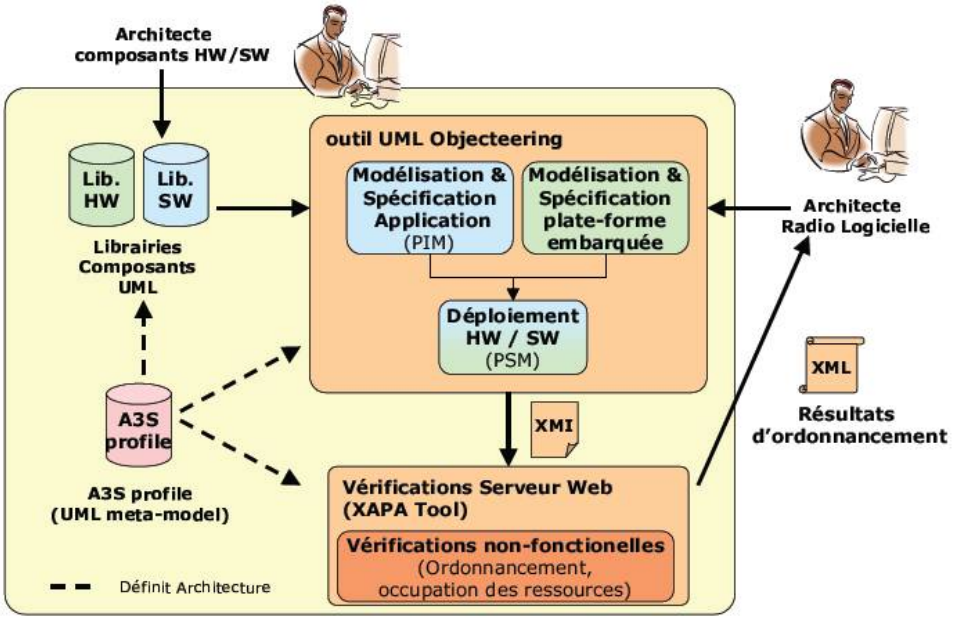


FIGURE 3.8: Description des étapes suivies dans le flot de conception A3S [79]

3.3.6 Discussion

Les efforts actuels fournis par la communauté de la Co-conception s'intéressent principalement au développement de méthodologies claires pour la conception, permettant la réduction du gap entre l'évolution de la technologie des semi-conducteurs et des outils de conception des

systèmes numériques. Les différentes représentations fournies par ces méthodologies prouvent que la Co-conception n'est autre qu'une partie de la conception de la totalité du système.

Les principaux aspects traités par les méthodologies présentés dans cette section se résument en trois points : 1) Les concepts utilisés durant toute la phase de co-conception (partant de la modélisation, à la spécification du système jusqu'au produit final) 2) La méthodologie suivie et son outil support et finalement 3) La manière adoptée pour la modélisation de l'application, de l'architecture et le modèle d'exécution.

En outre, l'utilisation des approches basées sur l'IDM pour la co-conception des SoCs a été discutée dans [28], qui souligne certains avantages : la réduction des coûts, la gestion de la complexité du silicium, l'augmenttation de la productivité. D'autres travaux [111][96][77][76][78] ont proposé et également justifié les avantages de l'utilisation de UML pour la modélisation de systèmes embarqués.

3.4 Modélisation de la reconfiguration dynamique

Le concepteur de systèmes électroniques d'aujourd'hui dispose d'un éventail de solutions matérielles pour réaliser un système conforme à son cahier des charges, en terme de consommation d'énergie, de performances temporelles, d'occupation d'espace, d'opportunité de reconfiguration ou d'évolutivité. Pour ce faire, il a besoin d'une architecture matérielle dédiée à son application avec tous les choix architecturaux que cela implique. Les FPGAs se présentent ici comme solution grâce à leur flexibilité et la possibilité d'être reconfigurés. Ils possèdent désormais des ressources de calcul importantes avec les processeurs qu'ils intègrent. Avec les contraintes de temps de mise sur le marché, il est donc primordial de proposer au concepteur un outil lui permettant d'estimer l'adéquation entre des architectures et des applications, au plus tôt dans le flot de conception (niveau système), afin de valider des solutions d'implantation conformes à son cahier des charges.

La reconfiguration dynamique (RD) est traitée selon plusieurs approches qui ont proposé plusieurs manières de la modéliser. Nous nous intéressons, dans cette section aux travaux qui ont traité la modélisation de la RD, la modélisation du contrôle de la reconfiguration et la modélisation du déploiement, d'autres approches ont proposé la modélisation de l'architecture physique des FPGAs sont également introduits. Les travaux qui seront présentés sont principalement basés sur un des profils UML et le standard IP-XACT. Nous ferons à la fin un bilan de synthèse afin de se positionner par rapport à ces travaux.

3.4.1 La modélisation haut-niveau de la RD

3.4.1.1 Le projet ANDRES

L'une des toutes premières propositions pour la modélisation de systèmes embarqués reconfigurables émane du projet européen ANDRES [1] [39]. L'objectif de cette méthodologie est de modéliser à la fois une application reconfigurable et une plate-forme hétérogène reconfigurable.

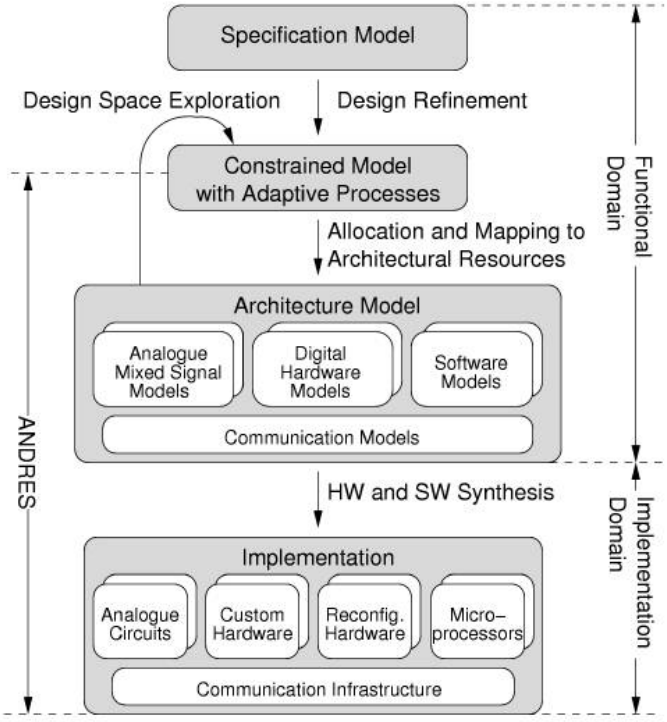


FIGURE 3.9: Flot de conception du projet ANDRES [39]

Pour ce faire, cette approche se base sur une extension de la librairie SystemC intitulée OSSS+R (Oldenburg System Synthesis Subset+Reconfigurable). L'idée de base ce la librairie OSSS+R est de modéliser une région reconfigurable d'un système matériel comme étant un objet adaptatif (polymorphique) avec une interface fixe. Cette interface est définie par une classe de base. Ainsi, cette approche permet d'adapter le concept de polymorphisme avec une abstraction de la reconfiguration matérielle.

Le flot de conception du projet ANDRES est illustré par la figure 3.9. Le flot part d'un modèle de contraintes du système conçu qui comprend les propriétés fonctionnelles et non fonctionnelles. Ce modèle de contraintes est un modèle décrit en SystemC dont les règles de conception ainsi que les directives sont basées sur une approche formelle. Ensuite, ce modèle subit un raffinement vers différentes cibles en utilisant la librairie OSSS+R qui fournit des concepts d'un langage haut niveau permettant la modélisation des systèmes reconfigurables. Ces modèles peuvent être automatiquement mappés sur des plateformes supportant la RDP.

Notons que l'inconvénient majeur de cette approche est le manque de portabilité des modèles car la librairie OSSS+R est propriétaire.

3.4.1.2 Le projet OverSoC

Les travaux de ce projet [68] proposent une méthodologie de modélisation à haut niveau pour l'implémentation des architectures dynamiquement reconfigurables. Les auteurs intègrent un système d'exploitation (Operating system OS) afin de fournir et traiter le mécanisme de la reconfiguration. La plateforme globale est divisée en deux composants *actif* et *réactif* repré-

sentant respectivement l'architecture reconfigurable (FPGA) et l'OS. L'OS est exécuté sur un processeur GPP (General Purpose Processor) via une interface spécifique avec l'FPGA. En outre, le composant actif est composé de plusieurs sous-composants qui représentent les composants de calcul et de reconfiguration, les ressources physiques de l'FPGA (CLBs, LUTs, etc.). Tandis que le composant réactif correspond au contrôleur ICAP. Enfin, le SystemC a été utilisé pour la phase de simulation et la vérification de l'OS afin de bien gérer les aspects de la RD.

3.4.2 Les approches GASPARD2 et MOPCOM pour la modélisation du contrôle de la reconfiguration

Les deux projets les plus proches de notre étude sont le projet GASPARD2 et MOPCOM. Tous les deux ont proposé des flots de conception des systèmes dynamiquement reconfigurables.

3.4.2.1 L'approche GASPARD2

Le projet GASPARD2 dont le flot de co-conception a été précédemment présenté, propose ici une manière particulière pour la modélisation du comportement des tâches dynamiquement reconfigurables.

La première notion spécifique aux travaux de GASPARD2 a été introduite par [46]. Ces travaux ont défini la notion d'automate de contrôle inspirée d'automates de modes. L'objectif de ces travaux est de modéliser une application contrôlée en vue d'automatiser la génération de son code en langages synchrones qui est une des plateforme ciblée par GASPARD2. Cette approche se base sur la séparation des flux de contrôle et de données. La figure 3.10 montre que la sortie de l'automate est une valeur de mode qui sera utilisée par la partie contrôlée de l'application. Le but est de choisir à la fin un mode de fonctionnement à activer parmi plusieurs modes exclusifs. Pour ce faire, un automate de contrôle est modélisé par une fonction de transition et une dépendance inter-répétition.

La valeur du mode en sortie de l'automate de contrôle est utilisée pour déterminer le mode à activer pour la partie contrôlée de l'application. Pour cette raison, chaque tâche contrôlée est modélisée par un composant composé qui regroupe les différents modes d'exécution de cette tâche comme le montre la figure 3.11.

En outre, les travaux de [46] ont également proposé plusieurs améliorations de la première version du profil GASPARD2 en lui introduisant de nouveaux concepts et des contraintes OCL.

Le stéréotype «*ControlledComponent*» est utilisé pour modéliser la tâche contrôlée alors que «*AlternativeComponentPart*» est appliqué sur les différents modes d'exécution de la tâche.

Une autre notion est introduite dans la méthode GASPARD2 afin de définir une tâche [70] : deux composants spécifiques `Mode Switch component` (MSC), Cf. figure 3.12(a) et `Gaspard State Graph` (GSG), Cf. figure 3.12(b).

Les implémentations liées aux modes sont rattachées au MSC via des collaborations UML. Alors que le GSG tient le rôle d'une machine à état et est équivalent au concept de *ModeBehavior* du package *CoreElement* du profil MARTE (chapitre 5). Ces deux composants sont encapsulés

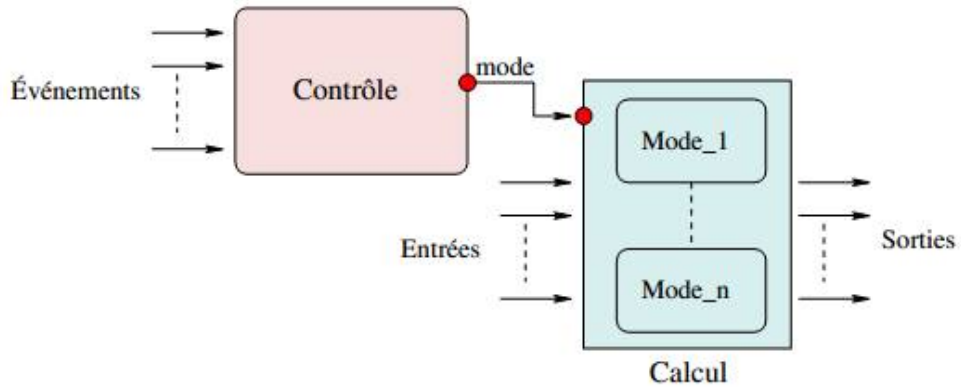


FIGURE 3.10: Vue globale du modèle de séparation contrôle/données [46]

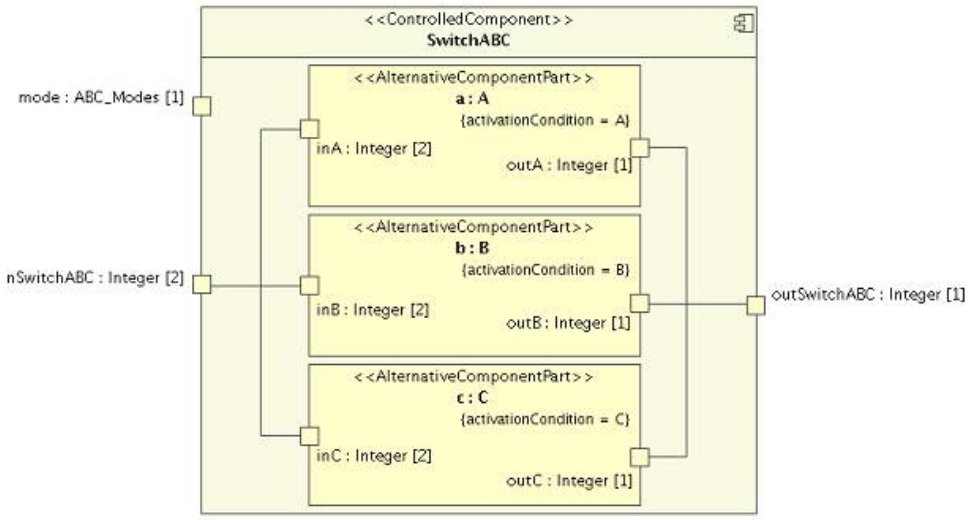
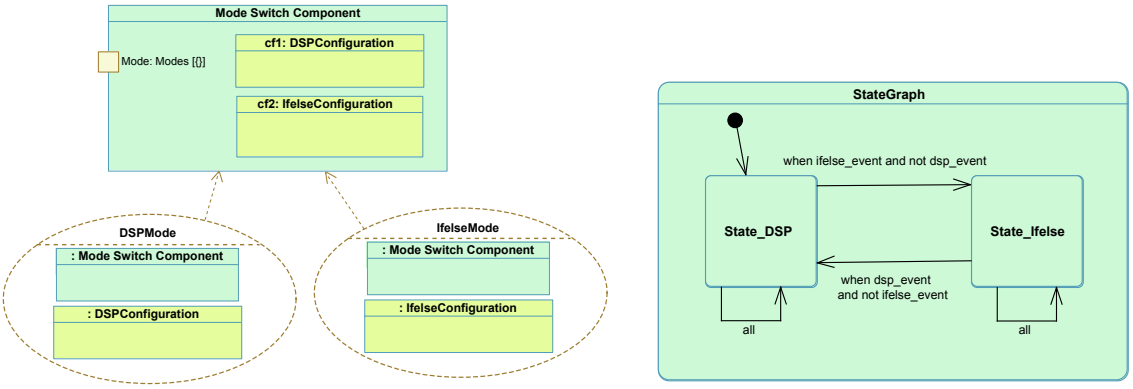


FIGURE 3.11: Exemple de modélisation de la partie contrôlée [46]



(a) Définition de `Mode Switch Component` (MSC)

(b) Définition d'un `Gaspard State Graph` (GSG)

FIGURE 3.12: Modélisation d'un composant reconfigurable dans GASPARD2

dans une tâche reconfigurable, dite `Macro Component`, Cf. figure 3.13.

Suivant un ensemble d'événements, le GSG adapte son comportement et transmet l'identifiant de son mode actif typé «*Mode*». Une fois l'identifiant reçu par le MSC, ce dernier sélectionne

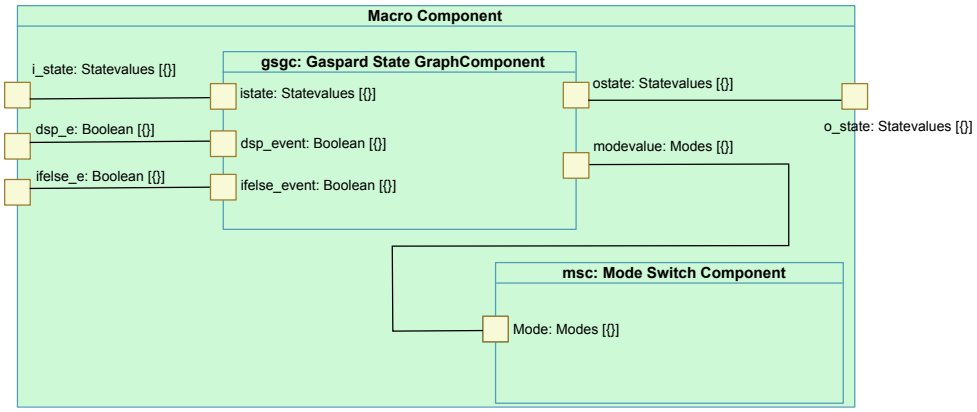


FIGURE 3.13: Modélisation du `Macro Component` dans GASPARD2

l'implémentation liée à ce mode.

D'autres approches ont traité la modélisation du contrôle de la reconfiguration et on été intégrés dans GASPARD2. Les premiers résultats [22] ont proposé une modélisation de haut-niveau d'un contrôleur modulaire et distribué pour la gestion de la reconfiguration dynamique sur les FPGAs. Cette approche, basée sur le profil MARTE, a été par la suite améliorée dans [91] pour modéliser le contrôle semi-distribué composé d'un ensemble de contrôleurs distribués modulaires assurant chacun les tâches d'observation, de prise de décision et de reconfiguration pour une région reconfigurable du système, et d'un coordinateur entre les décisions des contrôleurs distribués afin de respecter les contraintes et objectifs globaux du système.

3.4.2.2 L'approche MOPCOM

La reconfiguration selon la méthode MOPCOM [94] adresse deux cas de figures. Le premier définit la reconfiguration d'une tâche comme étant un changement d'implémentation de la tâche. Alors que le deuxième cas traite les reconfigurations pour l'exploitation temporelle des ressources physiques dans une séquence d'exécution de tâches.

Étant donné que dans cette approche, un modèle de calcul n'est pas imposé, il est donc nécessaire de traiter le second cas. En effet, afin d'obtenir une exécution correcte d'une séquence de tâches, il est important de définir l'ordre de modification de chaque ressource physique. Pour ce faire, la méthodologie MOPCOM définit un `Contrôleur` comme étant une tâche particulière stéréotypée «*RtUnit*», qui va gérer l'ordonnancement des tâches. La modélisation du contrôleur est illustrée par la figure 3.14 [41].

Dès réception d'événements provenant du système (événements reçus par les capteurs, ou résultat obtenu par d'autres tâches, etc.), le contrôleur va communiquer avec les autres tâches afin de déclencher leur reconfiguration, quand cela est nécessaire dans la séquence d'ordonnancement. L'état *add* de la machine à états du contrôleur correspond à un contrôleur qui communique avec le composant *adder* pour exécuter la tâche d'addition tandis que l'état *mult* correspond à un contrôleur qui communique avec le composant *multiplier* pour exécuter la tâche de multiplication. Cependant, c'est à l'utilisateur de définir le comportement du contrôleur

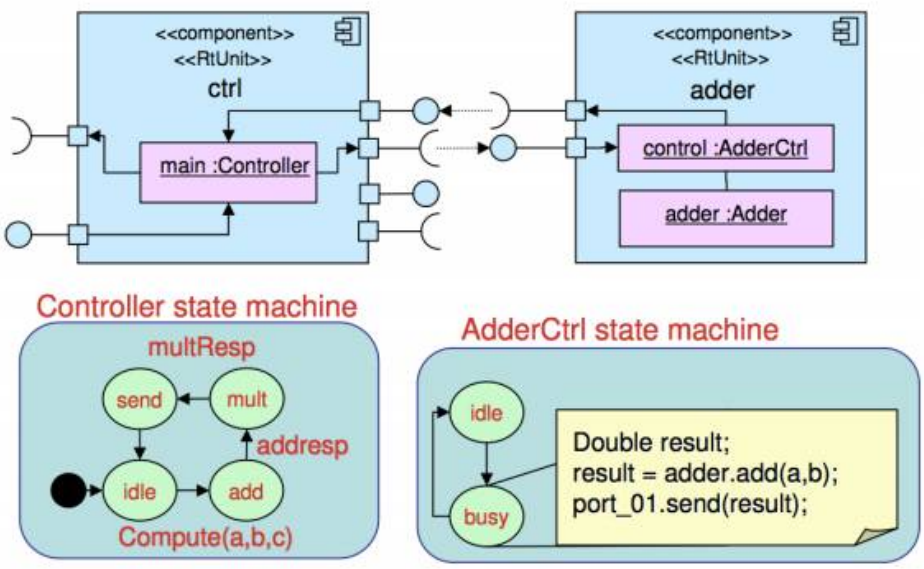


FIGURE 3.14: Modélisation du contrôleur (contrôle et ordonnancement) des tâches dans MOPCOM [41]

grâce à une classe encapsulée disposant d'une machine à état UML.

Par conséquent, cette approche considère la reconfiguration seulement d'un point de vue ordonnancement. Elle ne prend pas en compte d'autres éléments tels que les préférences des utilisateurs et ne définit pas explicitement les contraintes exigées.

3.4.3 La Modélisation du déploiement selon UML et IP-XACT

Le modèle en Y est généralement adapté pour représenter les approches de co-conception des SoCs. Le point central de ses branches représente l'association de l'application sur les ressources de l'architecture. Une autre étape est primordiale : le déploiement dans cette méthodologie car il permet de réduire le fossé entre une description à un très haut niveau d'abstraction et la génération d'un système synthétisable. Il est également important de relier chaque composant élémentaire à son IP (Intellectual Property) qui définit son (ses) implémentation(s). Plus précisément, certaines informations nécessaires et suffisantes devraient être fournies à ce stade afin de permettre le paramétrage des IPs et faciliter l'intégration du code de manière automatisé.

Certaines approches ont relié, de manière manuelle, la description d'un haut niveau d'abstraction à la phase de génération de codes. Tandis que d'autres approches ont proposé un méta-modèle intermédiaire à cet effet.

3.4.3.1 Modélisation avec les profils UML

Plusieurs travaux ont abordé l'utilisation de méthodologies UML et MARTE pour la conception des SoCs, en particulier au niveau de la phase de déploiement.

Le projet MOPCOM :

L'approche MOPCOM [43] est intéressante car elle cible des systèmes embarqués basés sur des

FPGAs en utilisant le profil UML MARTE [88]. Dans cette approche, trois modèles sont définis pour le système conçu : le modèle fonctionnel, le modèle d'architecture et l'allocation. Ce dernier associe les tâches aux différents composants du modèle d'exécution, ensuite un partitionnement logiciel/matériel est effectué. Cependant, les auteurs présentent seulement les résultats dans lesquels ils sont capables de générer le fichier MHS (*Microprocessor Hardware Specification*) qui sert d'entrée pour l'outil EDK de Xilinx. Ensuite, les auteurs rajoutent des fonctionnalités de réutilisation des IPs (*IP reuse*) dans [93] qui manquaient à leur approche initiale. Toutefois, l'information qui représente la réutilisation des IPs est annotée directement dans le modèle, ce qui rend le processus difficile à automatiser. En outre, ils ne tiennent pas compte des concepts de la reconfiguration dynamique partielle.

D'autres approches ont utilisé UML pour la modélisation des systèmes embarqués, mais très peu ont abordé les fonctionnalités de la reconfiguration dynamique et partielle (RDP).

Le projet GASPARD2 :

L'approche proposée par [72] présente un système dynamiquement reconfigurable en intégrant des extensions dans le profil MARTE avec des stéréotypes spécifiques. Cette approche a été développée au sein de l'environnement de conception GASPARD2 [26]. Toutefois, cette approche nécessite un grand niveau d'expertise et de connaissance tant tous les éléments du flot de conception de la RDP doivent être modélisés.

Par ailleurs, malgré sa complexité, les auteurs dans [71] ont montré la manière d'exploiter leur méthodologie afin de passer de modèles MARTE haut niveau vers une génération de code automatique. Pour ce faire, ils font usage d'un niveau intermédiaire appelé méta-modèle de Déploiement [70][12] qui fournit différents mécanismes permettant de relier le bas niveau d'implémentation (interfaces des composants, les paramètres de configuration, fichiers d'implémentations logiciels et matériels) avec les modèles haut niveau.

Cependant, le méta-modèle de déploiement proposé dans [12] n'est autre qu'une extension du profil Gaspard utilisé pour la modélisation. Il ne s'agit donc pas d'une représentation standard. Notons aussi que cette version proposée traite plusieurs détails qui concernent l'implémentation bas niveau des IPs. De plus, le méta-modèle proposé dans [70] reprend pratiquement les mêmes concepts du méta-modèle déploiement dans GASPARD2 et il a gardé tous les détails du bas niveau d'implémentation. Dans ses travaux, il a proposé des extensions dans le méta-modèle MARTE mais a tout de même utilisé le profil Gaspard dans ses modèles. Ses propositions étaient limitées au méta-modèle MARTE car il n'a pas explicité les relations entre ses concepts et les stéréotypes des paquetages de MARTE. Ceci limite la réutilisation de son approche. Ainsi, il reste étroitement dépendant de la méthodologie intégrée dans l'environnement GASPARD2. Il n'offre pas la possibilité de généraliser les concepts utilisés ce qui rend son approche difficile à appliquer sur les différents flots de conception et ne facilite donc pas l'interopérabilité des outils.

Malgré les limites de ces deux approches, nous avons fait usage de certaines idées présentées dans ces travaux concernant le déploiement et la modélisation des IPs. Cependant, les blocs d'IPs dans notre approche ont une représentation selon IP-XACT [64][63], ce qui simplifie leur

liaison aux modèles de composants MARTE (et donc, leur déploiement). Par conséquent, avoir cette représentation facilite le paramétrage et la personnalisation des composants de manière automatique, ce qui n'est pas le cas avec les approches précédentes.

3.4.3.2 Modélisation avec IP-XACT

Dans ce qui suit, nous présentons certains projets qui ont intégré le standard IP-XACT à la modélisation UML.

L'approche HELP

Cette approche a été proposée par l'équipe de recherche AOSTE à l'Inria dans le cadre du projet ANR HELP [2]. L'objectif de ce projet est de fournir un profil UML spécifique à IP-XACT pour l'intégration des blocs d'IPs dans des modèles de haut niveau. Les travaux dans [49] permettent de relier les conceptions en SystemC à leurs représentations structurelles dans IP-XACT moyennant une translation. Ensuite, ils exportent le modèle IP-XACT résultant dans une framework de modélisation en UML MARTE, pour permettre l'annotation des modèles existants avec des informations supplémentaires. En outre, les auteurs fournissent leur bibliothèque SocLib avec des exemples génériques en systemC. Le processus qui permet la création d'un système en IP-XACT à partir d'un code SystemC est illustré par la figure 3.15.

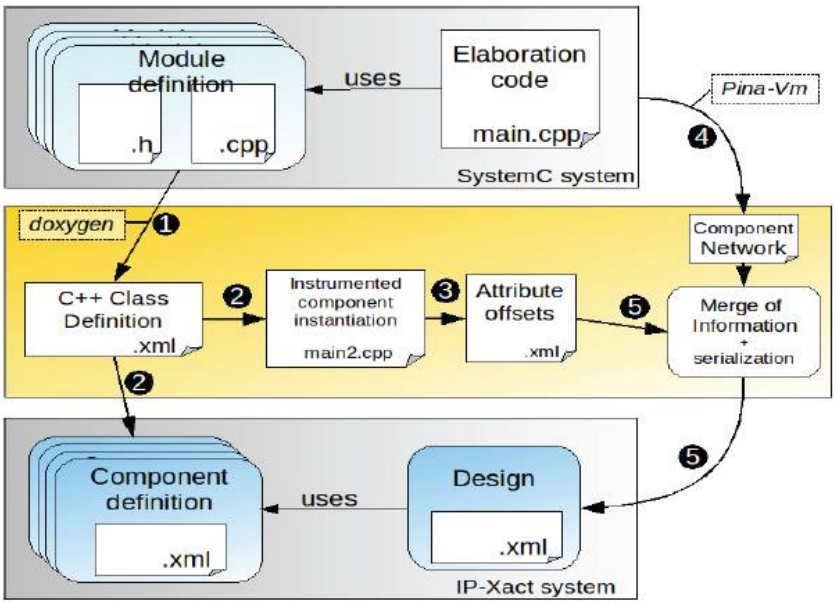


FIGURE 3.15: Processus de création d'un système en IP-XACT à partir d'un code en SystemC [49]

Dans [10], les auteurs ont créé un ensemble de méta-modèles IP-XACT pour les différents objets décrits dans le standard. La figure 3.16 décrit un exemple de méta-modèle d'un composant comme il est considéré en IP-XACT : selon son interface et ses mémoires hiérarchiques.

Par ailleurs, les auteurs présentent certaines règles de transformation nécessaires qui permettent de passer d'un modèle UML MARTE avec les extensions vers une description de la

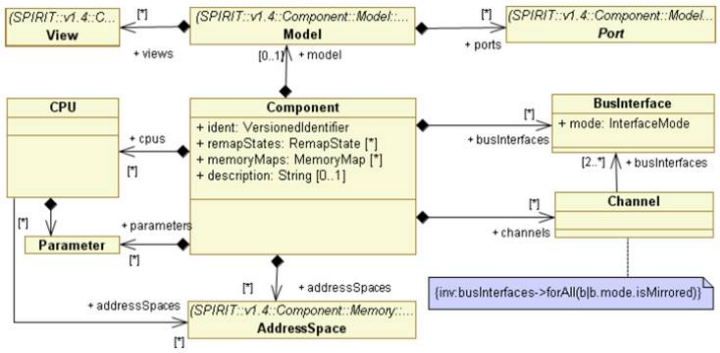


FIGURE 3.16: méta-modèle UML d'un composant IP-XACT [10]

conception IP-XACT.

Afin de faciliter l'exploration de la conception du système, ils font usage d'un outil dit SCIPX [48] qui permet l'automatisation de l'extraction de modèle structurel et sa conversion en une représentation IP-XACT. Ensuite, les auteurs présentent certaines règles de transformation nécessaires qui permettent de passer d'un modèle UML MARTE avec les extensions vers une description de la conception IP-XACT et vice versa. Le flot de transformation complet est décrit par la figure 3.17.

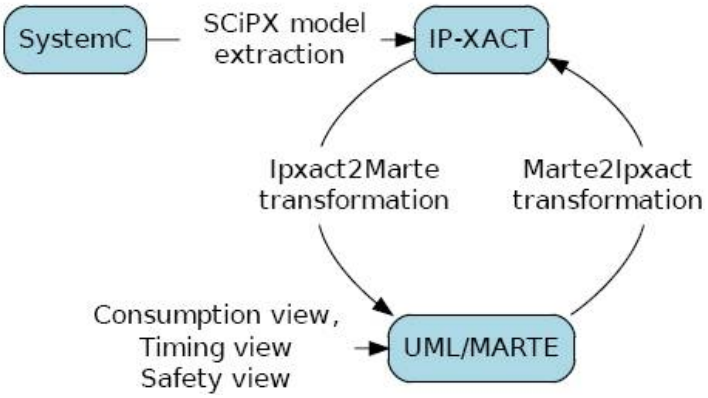


FIGURE 3.17: Le flot de transformation présenté par [48]

Cependant, il est à noter que les efforts fournis dans ces travaux ne font pas usage de tout le méta-modèle complet UML du standard IP-XACT, mais seulement un sous-ensemble de concepts qui est suffisant pour la génération d'une description IP-XACT d'une plateforme.

L'approche SPRINT

Dans [81], les auteurs ont étudié l'utilisation de UML pour la modélisation des composants et la description système en IP-XACT, en faisant correspondre différents concepts IP-XACT à leurs équivalents en UML. Ils utilisent un profil UML spécifique pour IP-XACT qui prend en compte la même quantité d'information correspondant à la description IP-XACT. Selon les auteurs, cette fonctionnalité permet la génération automatique des composants IP-XACT ainsi que les descriptions de la conception à partir des modèles UML.

Par ailleurs, cette approche permet l'intégration automatique des IPs existants dans UML. Tout comme le projet HELP, cette approche cible également le langage SystemC et permet la génération de code à partir de UML, passant par une représentation intermédiaire de IP-XACT, comme le montre la figure 3.18.

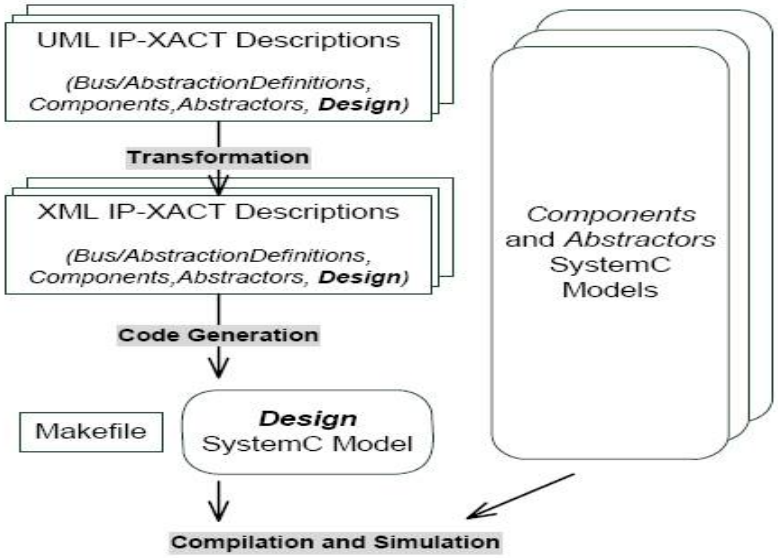


FIGURE 3.18: Le flot de conception SystemC dirigé par les descriptions UML IP-XACT [81]

De plus, plusieurs méta-modèles ont été présentés dans [81]. Ces méta-modèles correspondent aux différents concepts du standard IP-XACT tel que la description de composant, d'interface de bus, de port, etc. Le profil UML IP-XACT est représenté par la figure 3.19.

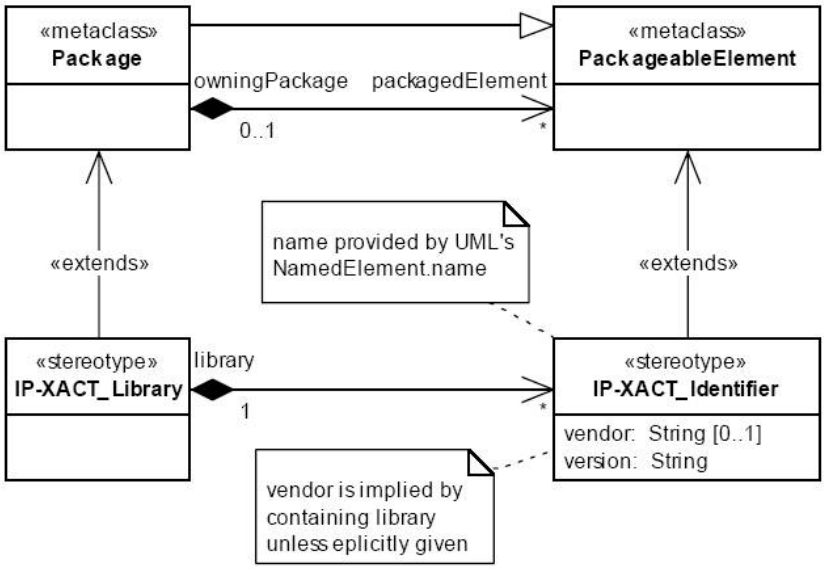


FIGURE 3.19: Le profil IP-XACT (Diagrammes de libraries et des identifiants) selon le projet SPRINT [81]

La méthodologie SPRINT est considérée comme l'une des toutes premières approches ayant pour objectif de fusionner UML et IP-XACT. Bien qu'elle soit très intéressante, nous pensons que cet effort est peu adapté pour la conception des SoCs en utilisant UML. En effet, cette approche

tente de mettre à la disposition du concepteur la totalité des méta-données contenues dans les descriptions IP-XACT telles que les interfaces de bus, les composants et les modèles, parmi d'autres objets définis dans la norme. Ainsi, l'inconvénient de cette approche est qu'elle peut rendre les diagrammes UML illisibles à cause de l'encombrement de l'information d'une part, et d'autre part, elle ne permet pas d'abstraire les informations appartenant au bas niveau. Ceci est particulièrement vrai avec les représentations IP-XACT, qui ont été conçues pour stocker autant d'informations que possible et d'être traitées par des machines.

Ces handicaps ont été reconnus par d'autres efforts en cours, dans lequel seulement un sous-ensemble de la description des composants IP-XACT est mis à disposition. Cette approche semble être plus logique à l'intégration de IP-XACT dans UML, puisque le concepteur de haut-niveau n'a pas besoin de connaître tous les détails de la description du composant. Un simple méta-modèle présenté lors de la phase de déploiement devrait permettre de capturer l'information la plus importante des IPs (à savoir, les paramètres et les interfaces de bus). Ainsi, le concepteur UML pourrait construire une plateforme contenant suffisamment d'informations pour la phase de transformations de modèles, dans laquelle une introspection est possible pour l'ajout de plus de détails pour la création d'un modèle exécutable.

Le profil TUT et le flot de conception Koski

Le profil TUT a été adapté pour l'exploration de l'espace de conception basée sur les modèles, la génération de code de l'application ainsi que l'intégration des IPs [11].

Ce profil défini dans [44] est une extension du profil UML pour la modélisation des systèmes embarqués. Les auteurs définissent un ensemble de stéréotypes pour modéliser les tâches d'application et les composants de la plateforme. Le tableau 3.2 résume l'ensemble de stéréotypes utilisés dans le profil TUT. Dans [42], le flot de conception de toute la méthodologie est défini. Tous les deux ont été utilisés dans [45] pour l'implémentation d'un terminal vidéo sans fil.

Selon la figure 3.20, les éléments du profil TUT sont définis comme suit : une application (*Application*) est composée de plusieurs composants applicatifs (*Application Component*) qui sont instanciés en tant que processus (*Application processes*). Ensuite, ces processus sont regroupés dans un groupe de processus (*Process Group*). Le principe est le même pour la partie matérielle de la plateforme. Enfin, le *Process Group* est mappé (alloué) sur les instances *platform component instances* en utilisant le *Platform mapping*.

La figure 3.21 illustre le flot de conception qui est composé de cinq phases principales où chaque phase affine la précédente. Le but étant de concevoir rapidement un système embarqué à partir d'une bibliothèque d'IPs permettant la DSE (Design Space Exploration) et la génération de code logiciel. La plateforme utilise une bibliothèque qui permet l'analyse de performance. De plus, le flot de conception permet la génération de code logiciel en langage C. Il dispose d'un outil d'exploration d'architecture qui permet de renvoyer des retours afin d'annoter le modèle UML. Il vise ainsi la modélisation des systèmes embarqués temps-réel.

La première proposition d'extension du profil TUT a été abordée dans [11] qui permet de

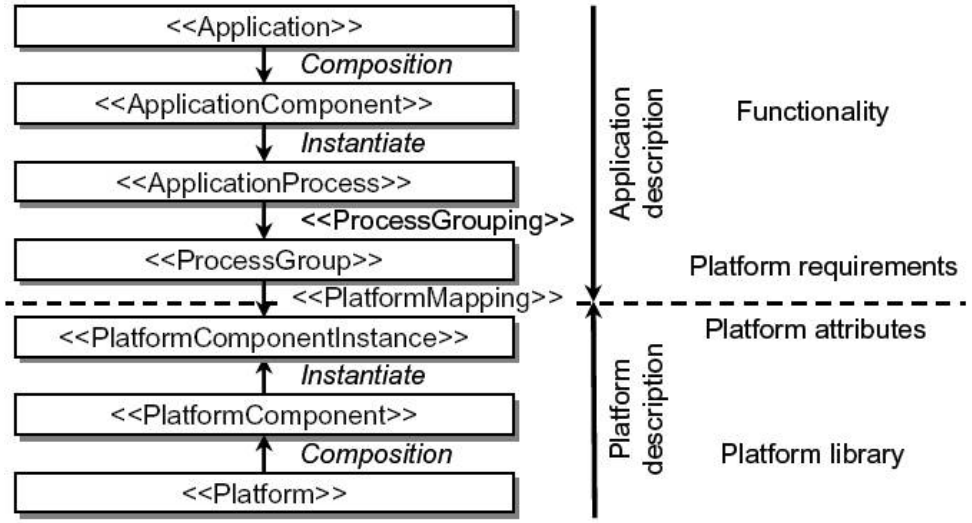


FIGURE 3.20: Le profil TUT

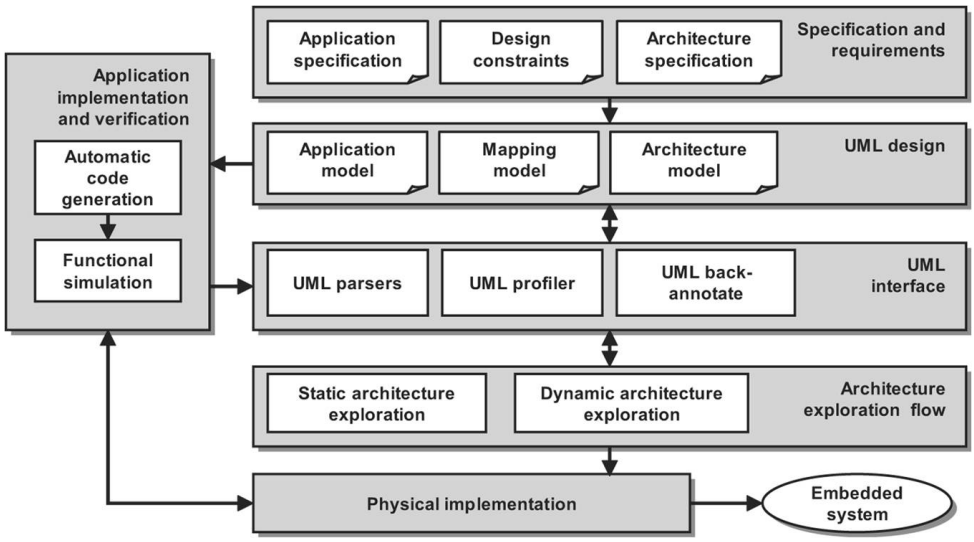


FIGURE 3.21: Le flot de conception de Koski

relier les modèles en UML TUT à une description de méta-donnée en IP-XACT. Les composants matériels et leurs descriptions IP-XACT correspondantes sont contenues dans une bibliothèque de composants IP. En effet, cette approche favorise la réutilisation des IPs et la description de la conception <top-level> en IP-XACT est représentée par un modèle de description matérielle en UML. Ainsi, ce framework permet au concepteur de réaliser une conception structurelle de la plateforme matérielle en utilisant les annotations UML, et de générer automatiquement les descriptions de la conception IP-XACT en utilisant un ensemble de règles de transformations, similaires à celles présentées dans le projet SPRINT et HELP. Les auteurs on montré, à l'aide d'un outil basé sur Eclipse créé, comment transformer les modèles TUT d'un système en un fichier XML personnalisé, appelé *XML System model* (XSM). Ce fichier contient les informations partagées par différents outils de conception dans le framework TUT (tels que les outils de DSE). Ce fichier est ensuite transformé en un fichier de conception top-level en IP-XACT qui est enfin

TABLE 3.2: Ensemble de stéréotypes utilisés dans le profil TUT [44]

Nom du Stéréotype *Méta-Classe étendue*	**Description**
Application (*Class*)	Classe d'application Top-level
ApplicationComponent (*Class*)	Composant fonctionnel de l'application (classe active)
ApplicationProcess (*Structural feature*)	Instance d'un composant fonctionnel de l'application
ProcessGroup (*Structural feature*)	Groupe de processus d'application
ProcessGrouping (*Dependency*)	Dépendance entre le processus d'application et le groupe de processus
Platform (*Class*)	Classe Top-level de la plateforme
PlatformComponent (*Class*)	Définit les caractéristiques du composant de la plateform
PlatformComponentInstance (*Structural feature*)	Instance d'un composant de la plateforme
CommunicationWrapper (*Dependency*)	Définit les paramètres du wrapper d'un agent de communication
CommunicationSegment (*Structural feature*)	Structure d'Interconnexion d'agents communicants
PlatformMapping (*Dependency*)	Dépendance entre un groupes de processus et une instance de composant de la plateforme

envoyé à un outil de configuration de plate-forme matérielle (*Hardware Platform Configuration Tool* (HPCT)). Cet outil intègre les composants des IPs basés sur les descriptions de méta-données IP-XACT. Par ailleurs, tous leurs travaux ont ciblé des FPGAs Altera qui ne supporte pas complètement la reconfiguration dynamique partielle comme les dispositifs de Xilinx.

Cependant, nous nous basons sur certaines de ces idées dans notre méthodologie pour l'intégration des IPs, en particulier en ce qui concerne la création des bibliothèques d'IPs. La seule différence entre l'approche TUT et la nôtre est que nous utilisons le profil MARTE qui est plus répandu et mieux adapté pour la modélisation des SoCs. Nous avons défini des extensions qui nous permettent d'intégrer les fonctionnalités de la réutilisation des IPs mais tout en exploitant des IPs qui doivent être intégrés dans un flot de conception RDP, ce qui qui requière des capacités spécifiques.

L'approche COMPLEX

Cette approche est l'une des plus intéressantes approches qui ont fusionné le profil UML MARTE et le standard IP-XACT, dans le cadre d'un projet européen COMPLEX [3]. Ce projet vise principalement la génération automatique de modèles exécutables de performance rapides. Il est basé sur un framework de UML MARTE supporté par un outillage entièrement intégré dans Eclipse, qui permet la génération automatique des différentes représentations intermédiaires. La figure 3.22 montre que le modèle de performance est généré au moyen d'un ensemble de générateurs de code, en s'appuyant sur des formats et des langages de normes intermédiaires

tels que IP-XACT, XML, SystemC et C/C++.

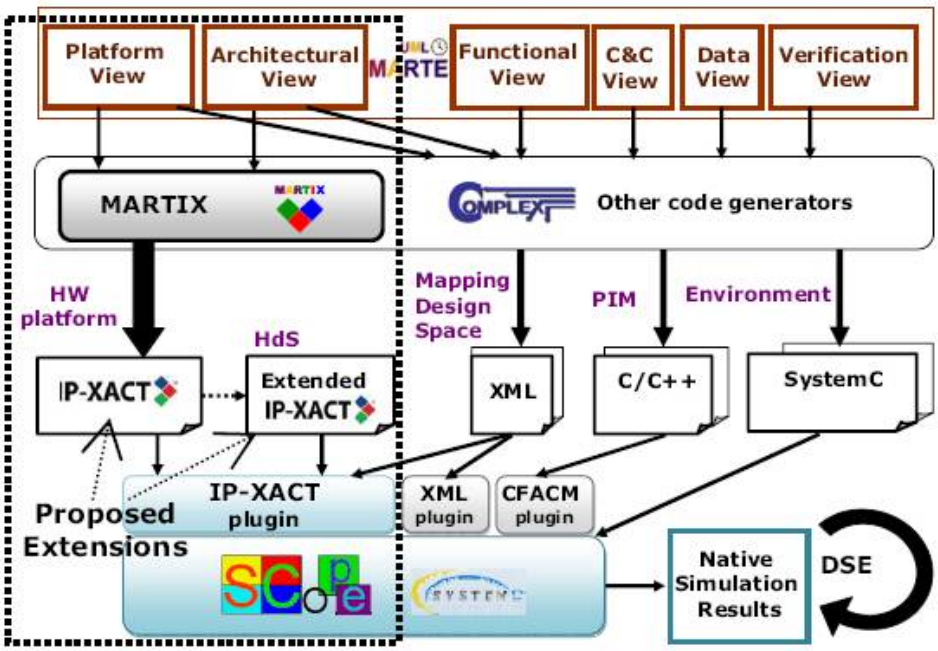


FIGURE 3.22: Le flot IP-XACT dans le flot de de génération de modèle de performance [37]

Dans cette approche, la méthodologie de spécification basée sur UML MARTE prend en compte la capture des architectures matérielles et logicielles, ainsi que des principaux paramètres du système. En outre, l'utilisateur peut spécifier un espace de conception (en gros, l'ensemble des paramètres accordables) où un outil d'exploration de conception va chercher les solutions d'implémentation optimales. Dans l'approche COMPLEX, l'outil d'exploration s'appuie sur des modèles dépendants de la plateforme (*Platform Dependent Models : PDMs*) exécutables en SystemC, selon différents niveaux d'abstraction. Ceci permet d'effectuer des boucles pour la DSE où l'estimation de la performance est faite selon un compromis entre différents précisions et la vitesse de simulation.

Concernant le standard IP-XACT, il y a un plug-in qui a été développé dans ce projet afin de permettre à un autre outil appelé *SCoPE* de lire la description de plateforme IP-XACT. L'objectif étant de construire un modèle rapide exécutable pour la validation fonctionnelle et l'estimation de performance. Dans [37] la génération de la description de plateforme IP-XACT et son usage ont été présentés, tandis que les auteurs dans [38] ont introduit les règles de transformations implémentées pour l'obtention d'une description IP-XACT à partir des modèles MARTE. Dans ces travaux, un outil de génération utilisé dans le projet COMPLEX appelé MATRIX a été également présenté.

3.4.3.3 Discussion

Malgré le nombre relativement important de propositions dans la modélisation de systèmes sur puce utilisant UML MARTE d'une part, et une combinaison de UML et IP-XACT d'autre

part, à ce jour il n'y a pas méthodologies qui utilisent une approche basée sur un méta-modèle pour les systèmes RDP. Par ailleurs, certaines approches précitées ou d'autres non citées, ciblent des implémentations sur FPGA, et en outre, elles ne fournissent pas des mécanismes clairs ou efficace pour la réutilisation IP, due en grande partie aux représentations XML intermédiaires pour la phase de déploiement.

De manière générale, ces représentations intermédiaires en XML ne sont pas orientées vers la description des plateformes SoCs, par conséquent, n'ont pas réussi à attirer l'intérêt de l'industrie. D'autres approches ont ciblé la génération de code SystemC mais sont toutefois en adéquation avec les objectifs de nos travaux de thèse. D'ailleurs, nous nous sommes inspirés de certaines idées en particulier le projet COMPLEX. Nous pensons que l'unification et la combinaison du profil UML MARTE avec le standard IP-XACT peut améliorer l'utilisation des approches dirigées par les modèles pour le développement des plateformes SoCs basées sur FPGA ; en particulier, faciliter la conception des systèmes dynamiquement reconfigurables.

Dans notre méthodologie de FAMOUS, nous utilisons certaines idées présentées dans [38] afin de réaliser les transformations de UML MARTE vers IP-XACT qui seront présentées dans le chapitre 6. La différence entre cette phase de notre méthodologie et celle proposé par l'outil MATRIX est que les modèles UML présentés comme entrée font usage d'une extension dans MARTE pour la phase de déploiement. Ceci permet en effet, de faciliter la réutilisation des IPs, un de nos objectifs qui sera introduit dans le chapitre suivant.

De plus, nous utilisons la description de la conception en IP-XACT non pas pour générer des modèles en SystemC (comme la plupart des approches), mais dans le but d'obtenir des modèles utilisés par l'environnement industriel de Xilinx XPS (*Xilinx Platform Studio*) pour la construction des plateformes SoCs basés sur FPGA.

3.4.4 Représentation physique des FPGAs

Une modélisation a été proposée par [87] qui met en valeur les aspects du sous-profil HRM de MARTE. Il s'agit de modéliser une plateforme matérielle SMP (Symmetric MultiProcessing). Cette architecture est composée de plusieurs processeurs qui partagent tous la même mémoire principale. Chaque processeur embarque du cache et relié aux autres par un bus système commun. D'autres composants : un DMA et une batterie étendent cette plateforme. Dans cet exemple, l'auteur sépare la vue logique de la vue physique de la modélisation.

La figure 3.23 représente la vue détaillée de la plateforme SMP. Ce modèle représente l'organisation des composants de la plateforme sur une grille (*grid*). Chacun de ces composants occupe une position particulière sachant qu'une *position* est un ensemble de rectangles contigus de la grille.

Il ressort de ces travaux que la représentation proposée concerne une simple plateforme dont les composants sont modélisés et organisés sur une grille contigüe. Tandis que nous nous intéressons dans notre étude à modéliser la plateforme de l'FPGA qui est considérée comme hétérogène vue les différentes ressources physiques qu'elle contient.

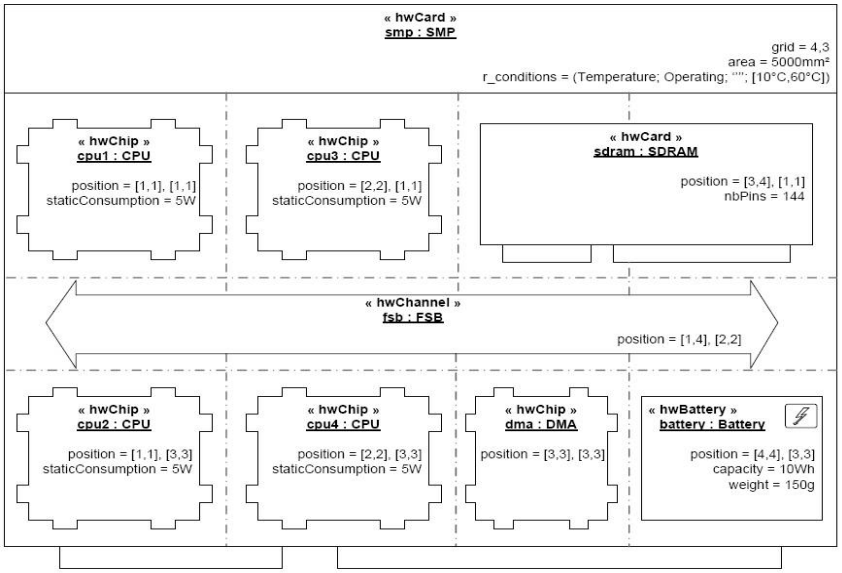


FIGURE 3.23: Vue physique détaillée de la plateforme SMP [87]

Le profil MARTE [56] propose toutefois, dans le même paquetage HRM, une représentation des placements d'architectures fonctionnelles (processeurs,mémoires, etc.) sur des supports physiques, tels que les FPGAs. Il permet aux concepteurs de spécifier le placement avec des concepts UML comme le montre la figure 3.24. En effet, un FPGA est représenté par le stéréotype «*HwPLD*», il peut contenir de la mémoire RAM (HwRAM) et est caractérisé par sa technologie (SRAM, Antifuse, Flash, Other). L'organisation des cellules de l'FPGA est caractérisée par le nombre de lignes et colonnes, mais aussi par le type de l'architecture (SymetricalArray, RawBased, SeaOfGates, Hierarchical PLD ou autre).

Afin de se rapprocher plus du niveau RTL, les travaux de [84] ont proposé une représentation des FPGAs sous formes de diverses couches superposées, comme le montre la figure 3.25. Chaque couche représente un type de ressources dans l'FPGA : une couche pour les blocs logiques et configurables (CLBs), une couche pour les mémoires et une couche pour les blocs multiplieurs (DSPs). Dans ces travaux, l'hétérogénéité dans les FPGAs est mis en valeur, en modélisant des zones qui contiennent différents types de ressources d'implémentation.

La décomposition en couches proposées permet d'identifier, pour un module donné, les types de ressources qu'il contient. En d'autres termes, il est possible de déterminer les différentes ressources réellement exploitées en fonction de leur type et de leur quantité. Dans le cas où un module n'exploite pas un type de ressources qu'elle englobe, il devient possible d'utiliser chacune de ces ressources par d'autres modules. Les modules peuvent ainsi se superposer afin d'exploiter au mieux chaque ressource de l'FPGA.

L'objectif final de cette approche [85] est de proposer un algorithme de placement qui gère individuellement les différents types de ressources hétérogènes d'un FPGA.

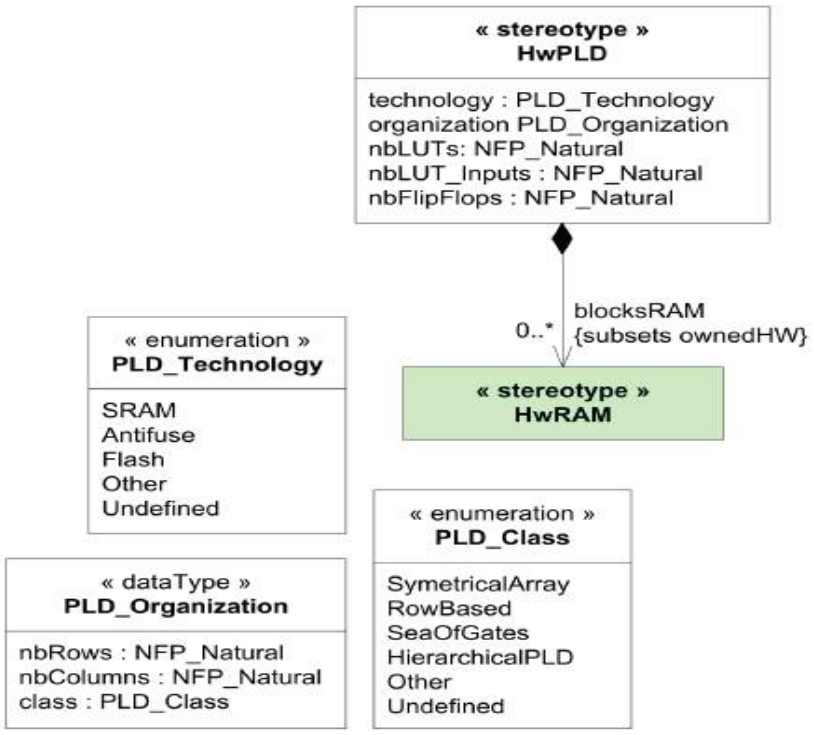


FIGURE 3.24: Modélisation des FPGAs selon le profil MARTE

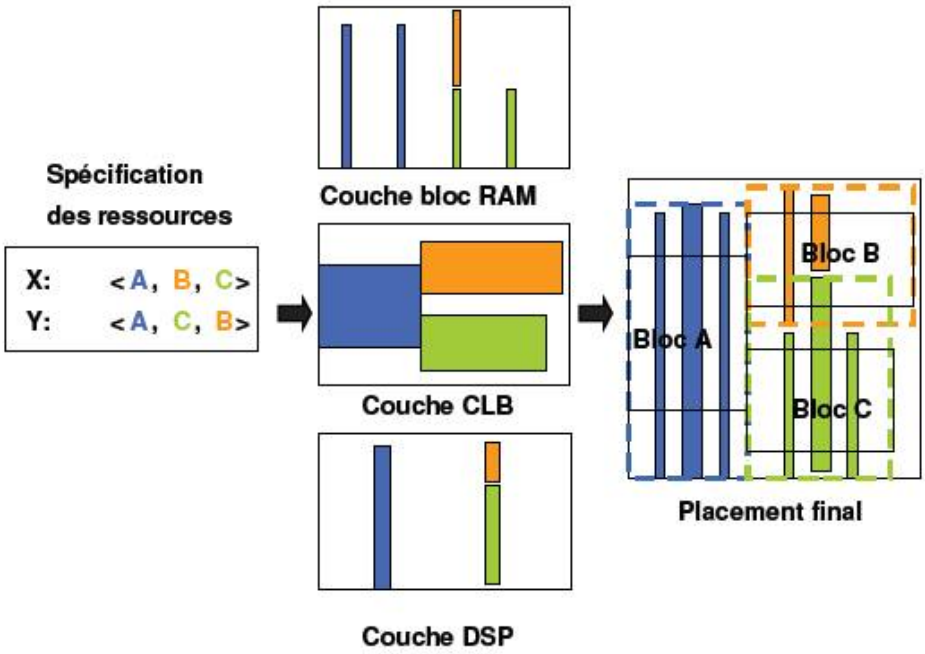


FIGURE 3.25: Représentation des FPGAs selon plusieurs couches [85]

3.4.5 Discussion

Les travaux de [87] concernent principalement des représentations de plateformes dont l'architecture est plus simple à modéliser que celle des FPGAs. La modélisation des composants

est basée sur les concepts du sous-profil HRM de MARTE. Les composants sont organisés sur un ensemble de rectangles contigus.

D'autres concepts dans HRM de MARTE permettent de classifier les FPGAs en fonction du type d'organisation des cellules, la taille de la grille de l'FPGA, ainsi que certaines informations technologiques. Toutefois, les concepts de MARTE ne permettent pas de modéliser une architecture hétérogène dont les ressources ne sont pas contiguës.

L'approche [84][85] réalise un placement sur Virtex 4 par le biais d'un fichier de contraintes utilisateur (*Relative Location (RLOC) Constraints*). En fonction de leur précision, les représentations permettent d'exprimer des placements voués ou non à être réalisés sur FPGA.

Nous partageons avec ces travaux la notion de séparation de vue (logique et physique). La modélisation à très haut niveau d'abstraction (où peu d'informations sont nécessaires pour la description de l'FPGA) s'avère intéressante car elle peut s'appliquer sur différentes technologies des FPGAs. En outre, une représentation plus abstraite est plus difficilement exploitable, elle doit être complétée par des fichiers de contraintes par exemple.

3.5 Synthèse

Nous partageons avec ces travaux des similarités et des différences. Le tableau 3.3 illustre l'évaluation des principaux travaux présentés dans ce chapitre. Cette évaluation se base sur un nombre de critères liés à l'élévation du niveau d'abstraction, la réutilisabilité et l'automatisation qui sont les objectifs visés par notre approche. Notre méthodologie de conception proposée dans ce manuscrit est illustrée par la dernière colonne. Elle vise à couvrir différents points permettant d'assurer ses objectifs à savoir :

- L'utilisation de l'IDM pour la conception conjointe des FPGAs, ceci afin de garantir l'abstraction et l'automatisation.
- La modélisation à haut niveau de la reconfiguration dynamique sur les FPGAs moyennant le standard UML ou tout autre profil UML parmis ceux présentés au début du chapitre.
- La modélisation du contrôle de la reconfiguration : différentes présentations du mécanisme de contrôle de la RD sont proposées.
- La modélisation du niveau intermédiaire de déploiement qui vise à supprimer le fossé (gap) entre la modélisation haut niveau et le niveau implémentation.
- La modélisation UML d'un module de diffusion nommé membrane qui permet la gestion et la sauvegarde du contexte. Plusieurs approches ont bien évidemment traité cet aspect mais sans pour autant utiliser UML.
- La modélisation physique de l'FPGA en particulier le placement des régions reconfigurables.

TABLE 3.3: Synthèse des principaux travaux sur la modélisation haut-niveau de la reconfiguration dynamique des FPGAs

Critères de comparaison	**Gaspard**[26]	**MoPCoM**[94]	**ANDRES**[39]/ **TUT**[11]	**COMPLEX**[37]/ **HELP**[49]	**UPES**[77]**Wang**[96]/ **A3S**[79]	**Notre approche**
Approche de co-conception	IDM	IDM	IDM	–	IDM	IDM
Langage/profil de modélisation	Gaspard/MARTE	UML2/MARTE	UML	MARTE	–	MARTE/RecoMARTE
Modélisation du contrôle de la RD	concepts Gaspard	MARTE	–	–	–	MARTE/RecoMARTE
Modélisation du déploiement	méta-modèle Gaspard	méta-modèle non défini	UML/IP-XACT	UML/IP-XACT	-/-/UML	RecoMARTE/IP-XACT
Cible :FPGA RDP	–	Oui	–	–	–	Oui
Modélisation UML de la sauvegarde du contexte	–	–	–	–	–	Oui
Modélisation physique des FPGAs	–	–	–	–	–	MARTE/RecoMARTE

Ce tableau nous a permis de démontrer que notre approche, bien qu'elle soit inspirée de ces travaux, elle permet toutefois de couvrir tous les points précités.

3.6 Conclusion

Les systèmes embarqués sont de plus en plus exigents quant à la modélisation et le développement du logiciel et du matériel qui doivent désormais avoir lieu en parallèle. Par conséquent, l'unification d'un langage de conception de haut-niveau 1) simplifie la construction d'un système plus cohérent 2) accélère le temps de conception et 3) facilite la communication entre les différentes personnes de l'équipe de conception. De plus, les méthodes de co-conception tirent les avantages de cette unification afin de construire des modèles mixtes sûrs et facilement exploitables.

Dans ce chapitre, nous avons présenté plusieurs travaux qui ont traité les points principaux de notre méthodologie. Tout d'abord, nous avons présenté les différents profils UML utilisés qui permettent la conception des SoCs. Nous avons également discuté du positionnement du profil MARTE, sur lequel nos travaux se basent, par rapport aux autres profils.

Ensuite, nous avons introduit les méthodologies les plus intéressantes qui se sont basées sur l'Ingénieurie Dirigée par les Modèles afin de concevoir des systèmes sur puce. Nous tirons profit de ce choix dans notre approche afin de réduire des coûts de conception, automatiser notre flot de conception et augmenter la productivité.

La caractéristique de la reconfiguration dynamique et partielle est une réalité dans les FPGAs, sauf que son utilisation est encore limitée aux experts du domaine de la RDP. En effet, les bas niveaux d'implémentation nécessitent une profonde connaissance pour le développement de ces systèmes. Nous avons donc étudié certaines approches qui ont traité la modélisation à haut niveau de la RD ainsi que certains aspects particuliers de la RD à savoir : le mécanisme du contrôle de la reconfiguration, les représentations intermédiaires reliant le haut niveau au niveau implémentation. De ce fait, certains travaux présentés ont proposé le niveau de déploiement des IPs en utilisant le profil UML d'une part et le standard IP-XACT, d'autre part. La dernière partie a concerné la modélisation de la plateforme physique des FPGAs.

Dans ce qui suit, nous présenterons nos contributions quant à la modélisation à haut niveau de la reconfiguration dynamique sur les FPGAs. La définition de notre méthodologie basée sur un flot de conception fera donc objet du chapitre suivant.

Deuxième partie

modélisation à haut niveau

CHAPITRE 4

MÉTHODOLOGIE DE MODÉLISATION À HAUT NIVEAU

4.1 Introduction

Dans ce chapitre, notre méthodologie de modélisation à haut niveau sera présentée. Elle est basée sur un flot de conception qui présente le premier axe de recherche du projet FAMOUS. D'une modélisation abstraite à haut niveau, vers une description détaillée d'un système qui se rapproche le plus de la réalité et qui vise principalement, les systèmes dynamiquement reconfigurables, basés sur FPGA. Dans ce qui suit, nos motivations pour la modélisation à haut niveau seront présentées. Par la suite, tous les niveaux de modélisation de notre flot seront détaillés.

4.2 Motivations : IDM et UML MARTE en particulier

Depuis ses débuts, la MDA a été employée avec succès par les développeurs du logiciel. A l'apparition du langage UML, l'utilisation de la MDA retrouve sa place au sein des concepteurs du matériel. L'intérêt qu'a porté la communauté électronique au langage UML a été assez immédiat. En particulier avec l'apparition du besoin d'avoir une description et une représentation complète d'un système électronique (*Electronic system Level* : ESL). L'usage des modèles haut niveau pour la spécification de systèmes complexes est une tradition dans l'ingénierie. La modélisation permet d'abstraire les aspects les plus importants afin de les analyser et les valider avant toute implémentation. Le recours à la modélisation est devenu, désormais, une pratique courante, autant pour le développement du logiciel que du matériel. C'est ainsi qu'intervient UML en mettant en place le paradigme objet et l'approche composant au service de la modélisation. Ces mécanismes offerts par ces deux approches sont parfaitement adaptés au développement aussi bien logiciel que matériel. Prenons l'exemple de la généralisation. C'est un mécanisme qui a servi efficacement à la classification des ressources matérielles. En effet, le matériel est largement varié mais facilement classifiable, on peut par exemple distinguer parmi les processeurs, plusieurs architectures, parmi ces architectures plusieurs générations et ainsi de suite.

Afin de couvrir l'intégralité du flot de conception des systèmes sur puce, il est impératif de définir et implémenter une méthodologie capable de couvrir tous les points du flot. Plusieurs travaux ont exploité la notion du flot en Y décrit dans la figure 4.1 [24]. Ce flot implique la définition d'un modèle UML fonctionnel, d'un modèle d'architecture et d'un modèle d'association des deux précédents pour l'analyse d'architecture et la génération de code. Parmi ces travaux, les projets MARTES [52] ou MOPCOM [51] qui ont spécifié des profils et des techniques associées pour la couverture du flot complet.

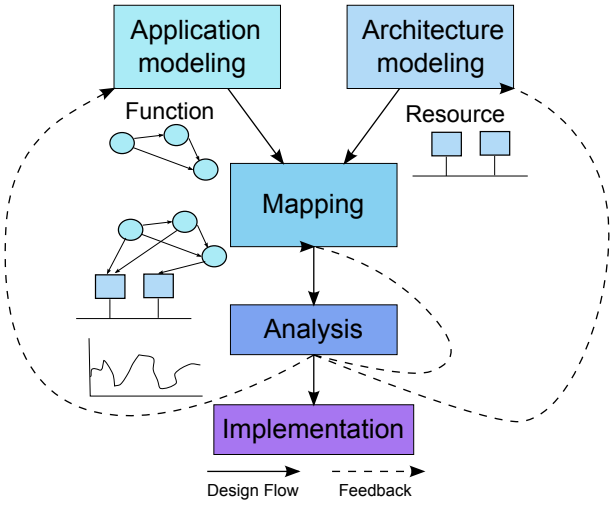


FIGURE 4.1: Flot de Co-conception des SoCs [24]

La standardisation de UML par l'OMG (Object Management Group), a encouragé le mécanisme d'extension au moyen des *profils*. En effet, les modèles définis en UML peuvent contenir

des `stéréotypes` et des `valeurs marquées` (ou *tagged values*). Un stéréotype représente une extension permettant de spécialiser un élément du méta-modèle (classe, association, attribut, etc.) alors que les tagged values marquent des attributs d'une classe pour préciser une contrainte ou un rôle particulier. La création de profil permet ainsi d'utiliser UML comme un DSL (Design Specification Language) voire comme un outil de Domain Specific Modeling. Dans le but de concevoir les SoCs, plus généralement, des systèmes temps-réel, plusieurs profils UML ont été développés. Deux catégories se distinguent. Une première catégorie regroupant des profils orientés implémentation, qui s'intéressent plus aux circuits et à la réalisation des composants. Leur but est de générer du code et utilisent de ce fait UML comme un HDL (Hardware Design Language) de haut niveau. Parmi ces profils, nous citons UML for SoC [60] et UML for SystemC [75].

La deuxième catégorie regroupe les profils modélisant le matériel de point de vue fonctionnel. A partir de modèles fonctionnels abstraits et compréhensibles du matériel, les intentions de conception sont communiquées entre les concepteurs afin de réaliser différentes analyses du système, aspect par aspect. Prenons l'exemple d'une analyse d'ordonnancement. Une telle phase requière une vue du matériel mettant en valeur l'architecture en termes de nombre de processeurs, de taille mémoire, etc. Une telle description est possible grâce à plusieurs profils UML comme le profil SPT (Schedulability, Performance and Time)[58], ou AADL (Architecture Analysis and Design Language)[5]. Ces profils permettent aussi d'annoter les fonctionnalités des composants modélisés selon leur nature (mémoire, ressource de calcul ou de communication, etc.).

Dans ce qui suit, nous résumons les principaux bénéfices qui nous ont motivé à utiliser les standards OMG : IDM, UML et MARTE. Ainsi, nous tirons avantages de ces techniques en particulier lors de la spécification de notre flot de conception.

4.2.1 Les bénéfices de UML

Dans notre approche, nous faisons appel au standard UML dans ses deux formes : le langage et le profil . L'avantage du langage UML est qu'il est adapté aux besoins de l'utilisateur grâce à ses nombreux diagrammes. Que ce soit pour la définition de spécifications, de modèle structurel ou comportemental, ces diagrammes permettent de représenter de différentes manières les besoins de l'utilisateur. En outre, UML favorise la dissociation des préoccupations : le nombre de diagrammes réalisables pour un même modèle est illimité. Ainsi, l'utilisateur peut laisser apparaître seulement les informations qu'il juge utiles pour une description particulière. La création d'un nouveau diagramme permet d'enrichir le même modèle qui centralise toutes les informations spécifiées.

Un second profit que nous tirons de UML est la réduction du nombre de formats de données au sein d'un flot de conception. En effet, l'utilisation d'un outil UML favorise la communication entre les différents corps de métiers impliqués qui peuvent désormais, s'échanger directement des modèles. Depuis la standardisation du format XMI, les acteurs ne sont plus obligés à utiliser

un même outil ce qui facilite encore plus la réutilisation des modèles.

4.2.2 MARTE en particulier

Parmi les profils UML dédiés à la conception des systèmes temps réel, le profil MARTE (Modeling and Analysis of Real-Time Systems)[56]. Le standard MARTE possède non seulement tous les avantages du profil UML mais fournit également les moyens de modéliser les systèmes temps réels.

Grâce à ses concepts précis, pour la modélisation d'architectures matérielles en relation avec un SoC, le profil UML MARTE offre clairement de nombreux avantages par rapport à d'autres approches. L'objectif de la modélisation avec MARTE est de pousser plus loin les niveaux d'abstraction vis-à-vis des détails d'implémentation des SoCs, qui deviennent de plus en plus complexes à concevoir.

Ainsi, le profil MARTE vient combler les manques des autres méthodes de modélisation. Il fournit un moyen commun pour la modélisation logicielle et matérielle d'un système, afin d'améliorer la communication entre les développeurs. Étant un standard de la MDA, il permet de conceptualiser un système grâce à des modèles qui, suivant le flot de développement, subiront des transformations appropriées afin d'obtenir une description qui se rapproche le plus du système réel.

4.2.3 Interopérabilité avec IDM

Certains outils spécifiques tels que la vérification formelle ou l'analyse d'architecture, requièrent un format de données spécifique en entrée. Ainsi, les transformations de modèles de l'IDM viennent faciliter l'extraction et la réécriture de ces données dans le format adéquat.

De plus, l'IDM fournit plusieurs possibilités d'automatisation telles que l'import/export qui peuvent intervenir dans certaines phases de conception. En effet, dans le cas d'une réalisation manuelle d'une activité spécifique, certaines opérations peuvent ainsi être codées dans une transformation de modèles ou un flot de génération de code. Ceci a pour effet non seulement d'accélérer de manière considérable le processus, mais aussi d'empêcher les erreurs qu'une opération manuelle aurait pu causer.

4.2.4 Synthèse

De ce qui précède, nous remarquons que, MARTE ne dispose pas encore de sémantique clairement définie pour la modélisation des SoCs reconfigurables. Dans ce contexte, le projet FAMOUS vient contribuer au profil UML MARTE en introduisant les concepts de la reconfiguration dynamique. L'objectif principal de ce travail est de mettre en place une méthodologie de modélisation pour la spécification et la conception de systèmes embarqués dynamiquement reconfigurables.

Cette méthodologie cible principalement les FPGAs et présente une modélisation à haut niveau des applications FAMOUS. Un méta-modèle contenant certains concepts de MARTE sera

donc défini accompagné de nouveaux concepts complémentaires. Ces extensions viennent remédier aux manques de MARTE quant à la modélisation de la reconfiguration dynamique. Le profil étendu contiendra ces extensions et sera renommé RECOMARTE *(Reconfigurable* MARTE*)*.

Par conséquent, le flot de conception présenté sera automatisé afin de combler le fossé qui existe entre les spécifications de haut niveau et les détails d'implémentation. Le but final étant de générer des fichiers utilisés par l'environnement de conception EDK de Xilinx et d'implémenter la description globale du système.

4.3 Flot de modélisation à haut niveau

La figure 4.2 décrit notre flot de conception haut niveau sur lequel sont basés les travaux de la thèse. En effet, ce flot concerne le premier axe du flot global de FAMOUS, à savoir la modélisation à haut niveau. En effet, la montée en abstraction des descriptions nécessite la création de niveaux de description intermédiaires. Il est donc nécessaire de définir l'objectif de chacun d'entre eux ainsi que le type d'informations à rajouter afin d'enrichir les modèles. Ces derniers sont indépendants des détails d'implémentation et éventuellement réutilisables. De plus, la description à haut niveau d'abstraction passe par une représentation graphique dont l'avantage principal est la vision immédiate de la structure, la hiérarchie et les dépendances des objets modélisés. D'où l'utilisation du profil UML MARTE.

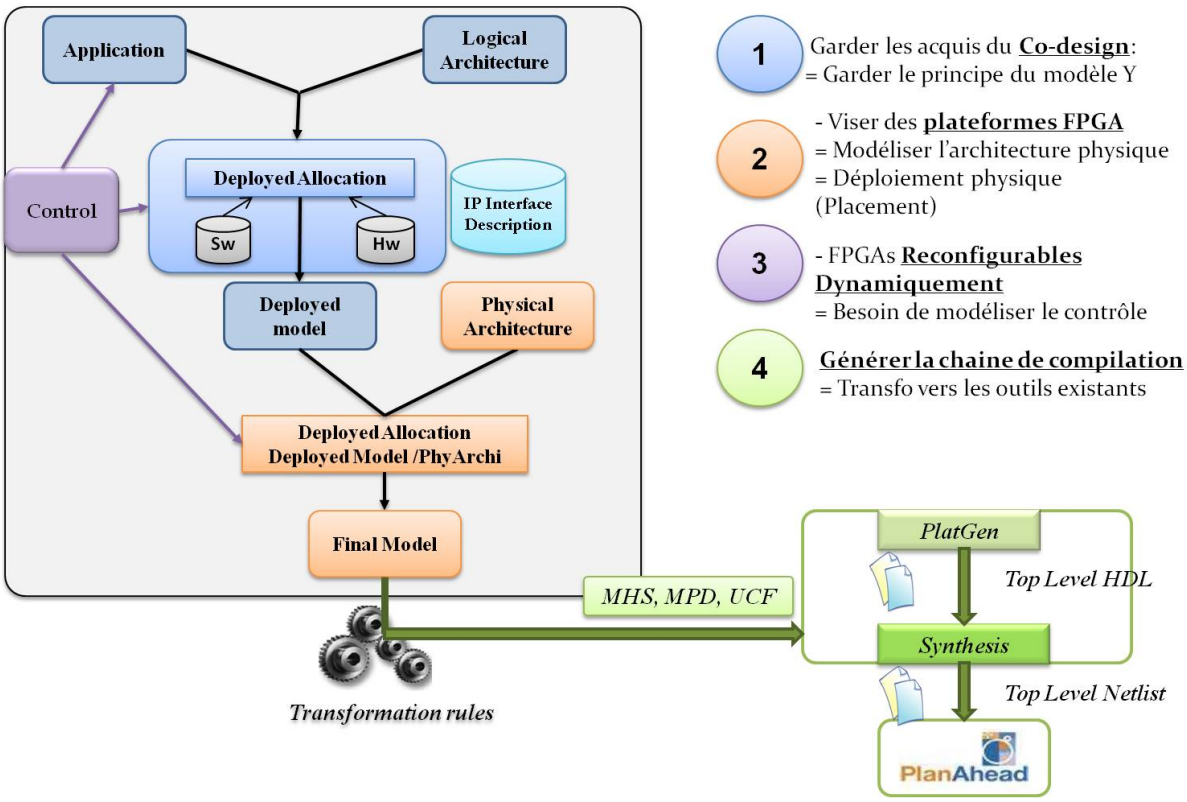


FIGURE 4.2: Flot de conception de la modélisation à haut niveau

Par la suite, le passage d'un niveau de description à un autre doit être automatisé de manière

à raffiner une description en la rapprochant du niveau de description de l'implémentation. Ce passage est possible grâce à des transformations de modèles. D'où l'intérêt de se baser sur le principe de l'IDM.

Par ailleurs, le processus de modélisation basé sur le profil MARTE est défini de manière à ce qu'il soit le plus indépendant possible des outils d'implémentation. Ainsi, il peut être facilement instancié dans n'importe quel outil (open-source modeller, java/EMF model transformer, etc.). Notre méthodologie permet d'introduire la notion de la reconfiguration dynamique et partielle dans le profil MARTE en vue de modéliser tout type d'FPGA supportant notre flot de conception.

Face à la multitude d'architectures reconfigurables hétérogènes potentielles, il s'avère ardu de développer une méthode permettant rapidement de converger vers la définition d'une architecture reconfigurable efficace (performante, flexible, facile à mettre en œuvre) pour une application ou un domaine d'applications donné et ceci très tôt dans le flot de conception. Pour définir l'architecture la plus en adéquation avec un ensemble d'applications, le concepteur d'architectures se trouve face à un large choix avec très peu d'outils pour l'accompagner. Ainsi, l'exploration de l'espace de conception (*Design Space Exploration : DSE*) vise l'adéquation de l'architecture avec l'application développée. Afin d'accélérer le processus de la DSE, il est impératif que le système soit modélisé et validé à différents niveaux d'abstraction dans le flot de conception.

Par conséquent, la vue globale de notre flot de conception peut être composée de quatre parties :

- *La modélisation à haut niveau* : il s'agit de la modélisation conjointe de l'application et l'architecture ainsi que la relation entre les deux. C'est une spécification basée sur le profil MARTE augmenté par certaines extensions définissant un nouveau profil nommé RECO-MARTE. Ce dernier contiendra les nouveaux concepts de la reconfiguration dynamique.
- *Modélisation ciblant les plateformes FPGAs* : Le modèle résultat de la conception conjointe sera mappé sur une plateforme physique de l'FPGA. Ainsi, la modélisation de l'architecture physique est nécessaire à ce stade afin de se rapprocher au mieux des technologies. Ce niveau définit ainsi la deuxième branche de notre modèle double Y (Cf. figure 4.3).
- *Contrôle de la RD* : étant donné que nos travaux dans le projet FAMOUS ciblent des FPGAs dynamiquement reconfigurables, alors il est inéluctable d'introduire le mécanisme du contrôle de la reconfiguration.
- *Transformation et génération de fichiers* : Les règles de transformations nécessaires sont spécifiées afin de pouvoir générer les fichiers utilisés par l'environnement de conception EDK de Xilinx. Ces fichiers permettront ainsi d'implémenter la description globale du système.

Un flot de conception en Double Y

La vue globale de notre flot de modélisation à haut niveau est partiellement inspiré du <Y chart>. Le flot ressemble à un *double Y* dont les branches correspondent aux différents niveaux de

modélisation comme illustré dans la figure 4.3. Ces modèles sont basés sur le profil UML MARTE ainsi que RECOMARTE pour la modélisation des concepts de la reconfiguration dynamique.

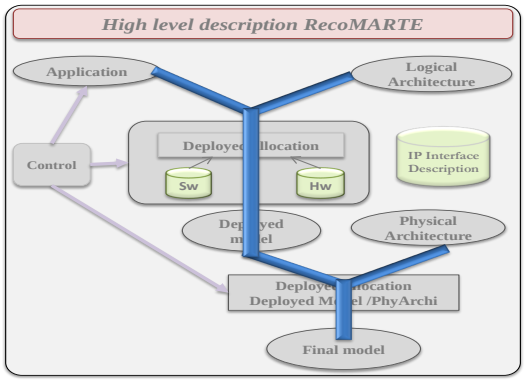


FIGURE 4.3: Vue globale du flot en "Double Y"

4.3.1 Modélisation conjointe à haut niveau

Les différents niveaux de modélisation sont détaillés dans la suite.

4.3.1.1 Niveau Application

La première étape consiste à modéliser les différentes tâches de l'application choisie. En effet, cette modélisation représente un graphe de tâches échangeant des flots de données, ainsi qu'un flot d'événements dédié au contrôle. Par ailleurs, l'application, étant un ensemble de tâches communicant entre elles, doit être modélisée indépendamment de la plateforme d'exécution. La communication entre les tâches est effectuée au moyen d'événements (envoi/réception). Un événement peut contenir des données qui sont envoyées d'une tâche à une autre ou bien transmises à partir de l'environnement externe.

Selon l'influence des facteurs externes sur l'exécution des tâches, nous pouvons distinguer deux types d'application : statique ou dynamique (reconfigurable).

Application statique

Dans le cas où l'exécution des tâches est totalement indépendante des facteurs externes, alors on parle d'un ordonnancement déterministe ce qui caractérise une application statique. La figure 4.4 illustre un graphe de tâches d'application statique. L'ordonnancement des tâches ne varie pas dans le temps et chacune des tâches est identique à chaque appel de celle-ci.

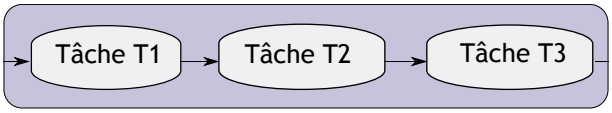


FIGURE 4.4: Graphe de tâches d'une application statique

Chaque tâche exécute toujours le même algorithme et envoie toujours les données résultantes

à la même tâche avec laquelle elle communique. Il s'agit du même motif utilisé (*pattern* en anglais) depuis la tâche T1 jusqu'à arriver à la tâche T3 qui envoie la donnée finale à un certain système externe.

Parmi les propriétés de l'ordonnancement déterministe d'une application statique, c'est que le calcul à réaliser, pour certaines données d'entrée, est toujours le même, quel que soit les données précédentes ou le changement d'environnement.

Application reconfigurable

Une application dynamique (ou reconfigurable) se caractérise par un comportement qui change suite à des facteurs externes saisies par l'application. En d'autres termes, l'application s'adapte à l'environnement et on parle alors de reconfigurabilité. La reconfigurabilité peut être considérée selon différents niveaux de granularité en fonction du type de changement apporté sur les tâches à exécuter.

– **Reconfiguration de l'Application** : Le plus haut niveau de granularité consiste à reconfigurer toute l'application comme le montre la figure 4.5. Cependant, le comportement dynamique que nous traitons réside à l'intérieur de l'application, ce qui élimine ce cas de figure.

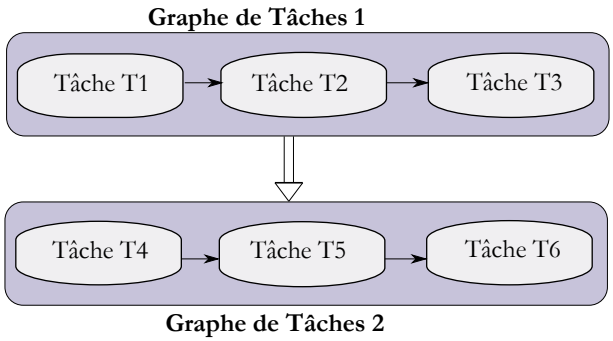


FIGURE 4.5: Reconfigurabilité : niveau Application

– **Reconfiguration d'une tâche** : La figure 4.6 illustre un graphe de tâches d'une application reconfigurable où la tâche T2 peut être substituée par la tâche $T2^{'}$. Le choix de la tâche à exécuter est décidé selon certains facteurs externes : une entrée différente au système, une donnée précédente qui déclenche la reconfiguration du système, etc.

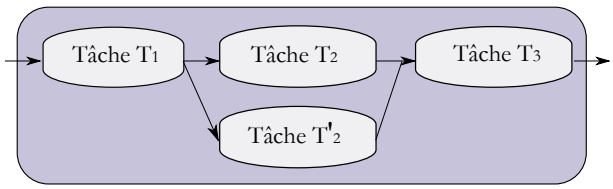


FIGURE 4.6: Reconfigurabilité : niveau Tâche

– **Reconfiguration d'une opération** : Le troisième niveau de granularité (Figure 4.7) est présent lors d'un changement interne d'une propriété d'une tâche tel que le changement de paramètres de certains algorithmes. De même que le cas précédent, la décision de reconfiguration est donné par une certaine analyse de données externes.

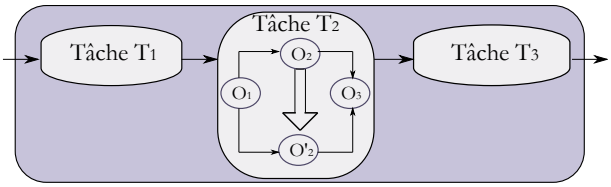


FIGURE 4.7: Reconfigurabilité : niveau Opération

La modélisation de l'application est étroitement liée à la notion de composant UML. Modéliser un composant revient à spécifier sa structure et décrire son comportement.

En effet, la représentation structurelle des composants définit en gros la nature des composants ainsi que les interfaces des communication entre les tâches. En revanche, la spécification comportementale de ces tâches est une notion impérative pour la spécification de la reconfiguration dynamique.

4.3.1.2 Niveau de l'architecture logique

A ce niveau de conception, la modélisation d'architecture logique peut être considérée comme une collection de composants (processeurs, bus, mémoire,etc.) connectés entre eux via des ports. Il s'agit d'une vue logique de ces différents composants avec un minimum d'informations sur leurs caractéristiques. En outre, certains de ces composants peuvent être considérés comme "reconfigurables". Ceci veut dire qu'un composant peut, au cours de son exécution, changer soit son comportement, ou bien son fonctionnement, voire les deux. Dans ce cas, il faut donner de l'importance sémantique lors de l'association d'une tâche sur un composant d'architecture.

4.3.1.3 Niveau de l'allocation Déployée

L'association ou l'allocation connue dans le modèle en Y et sur laquelle est basé également le profil MARTE, désigne la connexion des éléments applicatifs vers la plateforme. En d'autres termes, à chaque tâche de l'application modélisée, un composant de l'architecture lui est alloué (ou associé). Cependant, l'objectif de notre approche est d'offrir une représentation intermédiaire qui permet de passer d'une description de haut niveau vers une description de modèle exécutable. Sauf qu'à ce niveau, certaines informations décrivant les caractéristiques des composants ou leurs fonctionnalités, manquent dans une association simple. De ce fait, le niveau d'allocation déployée présenté dans notre flot de conception, vient enrichir, en quelques sortes le mécanisme d'allocation simple de MARTE. En effet, à ce niveau de description, les concepteurs doivent être capables de relier, de manière précise, les composants logiciels et matériels avec leurs propriétés intellectuelles (*Intellectual Properties : IP*) correspondantes. Plus précisément, certaines informations nécessaires doivent être fournies à ce niveau de conception dans le but de faciliter d'une part, le paramétrage et la configuration des IPs, et d'autre part leur ré-utilisation (IP reuse). En effet, dans la conception des SoCs, une fonctionnalité peut être implémentée

selon plusieurs manières. Par exemple, la fonctionnalité d'une tâche applicative a la possibilité d'être implémentée en tant qu'un accélérateur matériel en utilisant un langage de description matérielle (Hardware Description Languages : HDL), ou encore elle peut être optimisée pour un processeur et dans ce cas elle sera écrite en langage C/C++. Ainsi, les IPs sélectionnées sont sauvegardées dans une bibliothèque d'IPs et peuvent être classées en IP software ou hardware. Cette bibliothèque présente une description standard des composants reconfigurables selon le standard IP-XACT du consortium SPIRIT.

Par conséquent, le modèle obtenu au niveau de l'allocation déployée est un modèle enrichi par rapport au modèle d'allocation MARTE car il contient des informations spécifiques à la technologie des IPs sélectionnées. Il s'agit donc d'une fusion entre l'allocation simple de MARTE et le niveau de déploiement de UML. Le modèle déployé de l'application sur l'architecture logique (*Deployed model*) constitue non seulement le modèle résultat du premier Y mais aussi le point d'entrée du second Y de notre flot de conception. Par conséquent, la modélisation de l'allocation déployée conduira à la définition de nouveaux concepts qui seront donc intégrés dans RECOMARTE et détaillés dans la section 5.5.

4.3.2 Modélisation ciblée FPGA : Niveau de l'architecture physique

Ce niveau de modélisation se rapproche plus de la plateforme spécifique à un FPGA. Le modèle d'architecture physique permet au concepteur de définir explicitement les régions physiques statiques et reconfigurables dans la plateforme de l'FPGA choisi. En effet, la modélisation de la plateforme physique permet de réduire le fossé entre les diagrammes UML et l'agencement physique et la topologie de l'architecture ciblée. A ce niveau de conception, on se rapproche du niveau matériel et la modélisation devient plus dépendante de la technologie. Le fait de décrire le placement des différentes zones physiques, statique ou reconfigurable, veut dire que le concepteur a bien choisi le type de l'FPGA qu'il va utiliser. De plus, la description de l'architecture physique pourrait éventuellement fournir une aide supplémentaire aux outils commerciaux pour la Reconfiguration dynamique partielle (Partial Dynamic Reconfiguration : PDR) tel que l'outil Xilinx PlanAhead. Ainsi, le concepteur peut tout d'abord modéliser la disposition physique de l'FPGA. Ensuite, au niveau de simulation, il peut estimer avec précision si cette disposition est faisable et peut également déterminer le nombre de ressources physiques consommées. Les résultats de la simulation permettront au concepteur, par la suite, de modifier et d'améliorer ses modèles selon une stratégie d'exploration de l'espace de conception (Design Space Exploration : DSE). En conséquence, la modélisation de l'architecture physique nous ramène à définir de nouveaux concepts qui seront détaillés dans la section 5.4.5.

4.3.3 Le niveau Contrôle pour la Modélisation de la reconfiguration dynamique des FPGAs

La reconfiguration dynamique est la caractéristique clé dans la conception et l'implémentation des FPGAs, qui doivent être en mesure de faire face à l'environnement de l'utilisateur final ainsi que ses exigences. En effet, la reconfiguration dans les systèmes complexes, est assurée par

un mécanisme de contrôle qui peut être un RTOS, un middleware ou un contrôleur de reconfiguration comme le contrôleur ICAP de Xilinx. Dans un flot de conception basé sur un modèle en Y, le contrôle peut être (traditionnellement) intégré au niveau d'application, architecture et leur association. Le mécanisme du contrôle de la reconfiguration est généralement basé sur des modes et peut dépendre de certains changements tels que :

- les changements possibles qui peuvent avoir lieu lors de l'exécution des tâches,
- les changements dus aux exigences du concepteur,
- les contraintes exigées par les ressources matérielles de la plateforme ciblée,
- les critères de qualité de service tels que la qualité de la communication, le temps et la surface consommés pour la reconfiguration, les niveaux de consommation d'énergie, etc.

D'où l'importance de modéliser le contrôle de manière à ce qu'il soit applicable sur l'aspect applicatif et architectural. Les travaux effectués dans les projets MOPCOM [43] et GASPARD [26], se rapprochent dans le sens où tous les deux font appel à la sémantique des automates de mode UML qui est tout à fait appropriée pour la modélisation comportementale. Cependant, notre approche se propose d'aller plus loin dans l'exploitation des machines à états afin de représenter des aspects plus poussés de la reconfiguration. Ainsi, la modélisation du contrôle sera détaillée dans la section 5.6.

4.3.4 Modèle final

Le modèle final décrit la phase finale du flot de conception. En effet, à ce niveau de description, le concepteur a défini, d'une part, le modèle déployé de l'application sur l'architecture logique et dispose, d'autre part, des informations nécessaires et suffisantes sur les zones reconfigurables de son FPGA. Il peut donc effectuer un mapping du modèle déployé sur les différentes zones physiques tout en respectant un certain nombre de contraintes. Ce mapping est également considéré, dans notre méthodologie, comme une allocation déployée plutôt physique. Par ailleurs, les contraintes à respecter lors de ce mapping peuvent concerner plusieurs facteurs tels que la surface consommée, le temps d'exécution d'un module ou encore des critères de qualité de service (QoS).

Enfin, plus la description du modèle final est correcte et respecte parfaitement les différentes contraintes, plus le modèle se rapproche plus du système réel. Ce modèle subira, par la suite, des transformations appropriées afin de générer les fichiers utilisés par l'environnement de conception EDK de Xilinx.

4.4 Transformations et Génération de fichiers

Le premier objectif de l'approche de conception dirigée par les modèles est de décrire et spécifier un système donné, à un haut niveau d'abstraction. Ensuite, cette approche vise à transformer ces modèles vers d'autres représentations. Selon notre flot de conception, deux chaînes de transformations sont possibles.

La première chaîne consiste à transformer le modèle déployé vers une représentation standard de composants reconfigurables basée sur IP-XACT. Plusieurs travaux ont proposé la réalisation des transformations de MARTE vers IP-XACT. Cette approche considère un méta-modèle assez complet, ce qui rend, dans la plupart des cas, la manipulation des composants extrêmement lourde. Pour ce faire, nous présentons dans [65] une approche basée sur une description simplifiée des aspects importants à partir d'une librairie MARTE. Cette librairie contient des transformations des composants IP-XACT vers des modèles de déploiement MARTE. Le méta-modèle de déploiement utilisé dans ces travaux a permis de réaliser ce mapping et de promouvoir la ré-utilisation des IPs dans notre méthodologie. Notre approche a également fourni la possibilité de paramétrer les IPs sélectionnées. Les fichiers générés suite à cette transformation sont utilisés par l'environnement Xilinx Embedded Development Kit (EDK) [104, 105]. Les fichiers concernés sont le Microprocessor Hardware Specification (MHS) et le Microprocessor Peripheral Description (MPD).

Par ailleurs, le recours au modèle final comme point de départ pour la deuxième chaîne de transformation, permet de générer plus de fichiers à utiliser par l'environnement EDK. En effet, la description finale obtenue à la fin du flot double Y est enrichie par des informations sur l'architecture physique de l'FPGA, à savoir les coordonnées et l'agencement des zones physiques, statique et reconfigurable. Grâce à ces informations, le fichier User Constraint File (UCF) est généré en plus des deux autres fichiers MHS et MPD.

Tous ces fichiers seront, par la suite, utilisés par l'outil PlatGen [107] afin de générer la plateforme du système sur puce. En effet, cet outil génère une description HDL *Top-level* tandis que les fichiers HDL pour les modules reconfigurables sont recueillis à partir d'une bibliothèque indépendante. Ensuite, la modélisation *Top-level* ainsi que chaque module reconfigurable seront séparément synthétisés. Enfin, les *Netlists* générés seront utilisés comme entrées à l'outil Xilinx PlanAhead [100] où l'implémentation du système sera réalisée.

4.5 Intégration dans le flot général de FAMOUS

Notre flot de conception basé sur le double Y précédemment décrit, est enfin intégré au sein du flot complet du projet FAMOUS, comme le montre la figure 4.8. Ce chapitre résume la première contribution de la partie "Modélisation haut-niveau", du flot intégral à savoir la réalisation d'un flot de conception en prenant compte de la reconfiguration dynamique.

Le modèle complet établi en RecoMARTE, résultat de notre flot de conception, représente le point d'entrée nécessaire aux autres axes de recherche du projet. Il répond ainsi aux besoins exigés par les acteurs de FAMOUS.

4.6 Conclusion

La complexité de conception des systèmes embarqués est en augmentation continue au fur et à mesure que la technologie est en évolution ainsi que les attentes des utilisateurs pour de

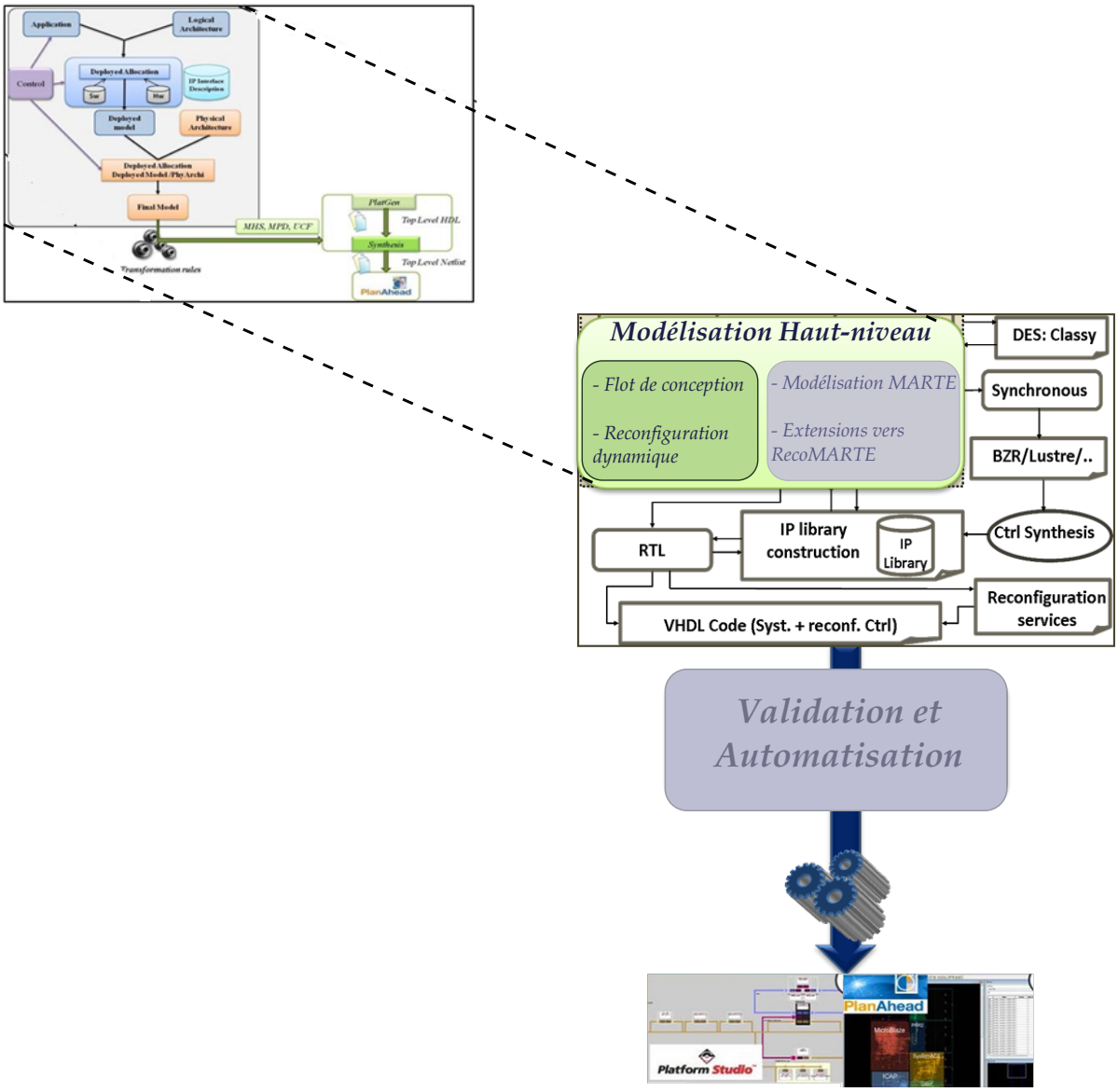


FIGURE 4.8: Contribution 1 : Définition du flot de conception et intégration dans le projet FAMOUS

meilleurs produits ; tous ces facteurs ne cessent de demander des produits de meilleure qualité avec un temps de mise sur le marché assez réduit.

Les méthodologies actuellement utilisées pour développer des systèmes électroniques embarqués complexes sont plus des démarches de conception que de réelles méthodologies formalisées. Il semble donc inéluctable d'élever le niveau de représentation des systèmes vers plus d'abstraction et de proposer une méthodologie formalisée et basée sur des standards pouvant répondre au mieux à ces challenges afin d'améliorer sensiblement la productivité.

Ainsi, nous avons présenté dans ce chapitre notre méthodologie de conception des systèmes reconfigurables basés sur FPGA, visant à diminuer la complexité grâce à la modélisation de différents niveaux d'abstraction. Étant basée sur les différents standards OMG : UML, MDA et MARTE, notre approche permet de faciliter la réutilisation et assurer la productivité de la conception.

Tout d'abord, nous avons présenté nos motivations pour le besoin de définir une méthodologie de conception et nous avons justifié notre choix d'utiliser le profil UML MARTE en particulier. Ce profil a prouvé sa capacité à combler les manques des autres méthodes de modélisation et s'est distingué en tant qu'outil idéal pour adresser la conception des SoCs avec un haut niveau d'abstraction. Par la suite, nous avons décrit le flot de modélisation de haut niveau sur lequel repose notre approche. Pour chaque niveau du flot, nous avons présenté les différentes étapes de modélisation. Ensuite, les modèles subiront des transformations de modèle vers modèle selon l'approche IDM, jusqu'à la génération de fichiers permettant d'implémenter la description globale du système.

Cependant, le profil UML MARTE ne dispose pas encore de concepts clairement définis pour la modélisation des systèmes dynamiquement reconfigurables. Dans ce contexte, nous présenterons dans le chapitre suivant, notre contribution dans MARTE adressant les différentes extensions ajoutées dans le but d'obtenir un profil plus riche et complet, qui sera baptisé RecoMARTE.

CHAPITRE 5

DE MARTE VERS RECOMARTE

5.1 Introduction

Dans le chapitre précédent, nous avons introduit notre flot de conception basé sur différents niveaux d'abstraction. Notre approche de conception est basée principalement sur des standards OMG : UML, MDA, MARTE et a pour but de réaliser l'implémentation de système à travers des raffinements de modèles (transformations).

Dans ce chapitre, nous présentons notre méthodologie de modélisation ciblant les systèmes dynamiquement reconfigurables, plus précisément les FPGAs. Cette méthodologie est basée sur le flot de conception présenté dans le chapitre précédent. Le profil MARTE est utilisé avec certaines extensions afin de prendre en compte la modélisation des concepts de la reconfiguration dynamique. Ainsi, ces extensions sont présentées, justifiées et définissent le nouveau profil obtenu baptisé RecoMARTE. Ensuite, pour chaque niveau d'abstraction, nous présentons les concepts utilisés et nous illustrons nos propos avec des exemples.

5.2 Exigences de base

Certaines conditions doivent être respectées afin de garantir la bonne utilisation des nouveaux concepts du profil étendu RecoMARTE. Les extensions du profil MARTE sont proposées à différents niveaux d'abstraction et spécifiées sous plusieurs formes (paquetage, stéréotypes, etc.). Les aspects suivants doivent être pris en compte :

- *Conformité avec le profil MARTE* : les nouveaux concepts ajoutés prenant en compte l'aspect de la reconfiguration dynamique, doivent être compatibles avec MARTE. En effet, la spécification des concepts doit respecter la sémantique et la description utilisées dans la description du profil MARTE.
- *Compatibilité avec les outils de modélisation* : La modélisation graphique utilisant le nouveau profil RecoMARTE, doit être compatible avec les outils de modélisation supportant le profil MARTE, exemple : Papyrus [1], Rhapsody [2], etc.
- *Précision de la sémantique* : Les concepts étendus doivent être précis et clairs, ce qui facilitera, d'une part, la documentation et la communication, et d'autre part, les transformations de modèles qui permettent la génération de code.
- *Présentation du profil et méta-modèle* :les nouvelles extensions sont présentées de deux manières différentes : sous forme de stéréotypes dans le profil RecoMARTE afin de modéliser notre système selon les différents niveaux d'abstraction ; et sous forme de méta-modèle Ecore pour l'utiliser lors de la phase de transformation de modèles.

5.3 Le profil HLAM pour la modélisation de l'application

Les différentes vues proposées par UML sont complémentaires les unes des autres. Elles permettent de mettre en évidence différents aspects d'un logiciel à réaliser et de séparer les aspects fonctionnels des aspects architecturaux, ou les aspects statiques des aspects dynamiques. Afin de modéliser l'application, précédemment présentée dans le chapitre 4, nous donnons sa spécification structurelle et comportementale. Une tâche d'une application est étroitement liée à la notion de composant UML. Ainsi, modéliser un composant revient à spécifier sa structure et décrire son comportement.

1. http ://www.papyrusuml.org/
2. http ://www-03.ibm.com/software/products/us/en/ratirhapfami/

5.3.1 Représentation structurelle

La vue structurelle d'une application est représentée par un `diagramme de classes`. Celui-ci favorise la structuration des données et tentent d'identifier les objets/composants constituant le programme, leurs `attributs`, `opérations` et `méthodes`, ainsi que les `associations` qui les unissent.

Grâce au concept de composant UML, une tâche indépendante peut être modélisée ainsi que ses communications avec d'autres tâches. Les connexions entre les différentes tâches sont modélisées grâce à des ports formant ainsi un graphe. Ainsi, le profil UML MARTE fournit le moyen de modéliser ces tâches grâce au stéréotype «*RtUnit*» du paquetage MARTE HLAM (*High-Level Application Modeling*). Ce stéréotype, illustré par la figure 5.1, permet de spécifier les propriétés temps-réel, soit la sémantique d'exécution d'une tâche.

Un «*RtUnit*» est similaire à une classe active en UML mais avec plus de détails sur la description des sémantiques. Il possède une ou plusieurs ressources d'ordonnancement (*GRM : :Scheduling : :SchedulableResource*).

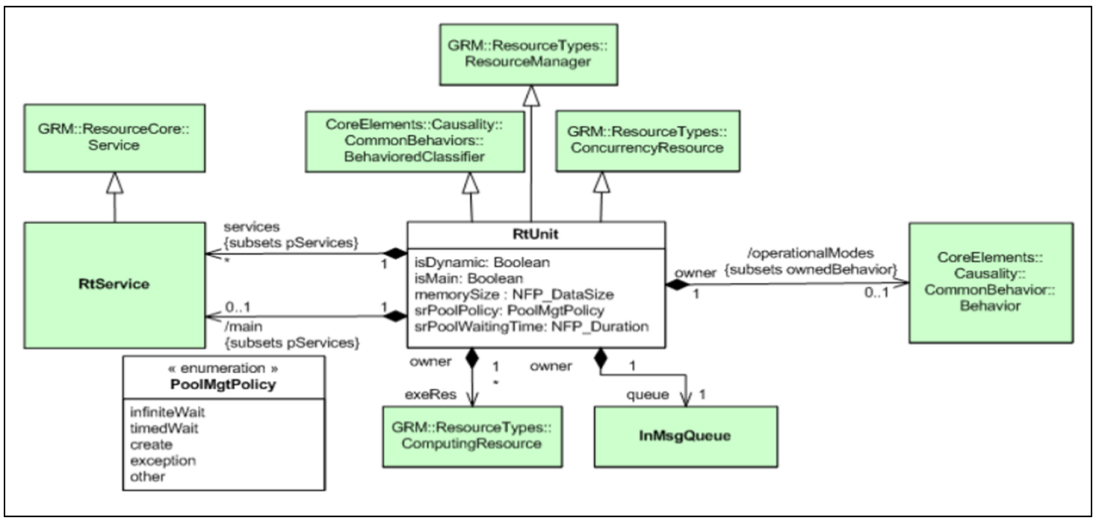


FIGURE 5.1: Le stéréotype «*RtUnit*» du paquetage MARTE HLAM

Un «*RtUnit*» possède aussi un contrôleur de concurrence et de comportement qui gère les contraintes des messages en fonction de son état courant ainsi que les contraintes d'exécution concurrentes rattachées aux messages. En outre, une application possède au moins un «*RtUnit*» principal dont l'attribut booléen *isMain* est mis à Vrai.

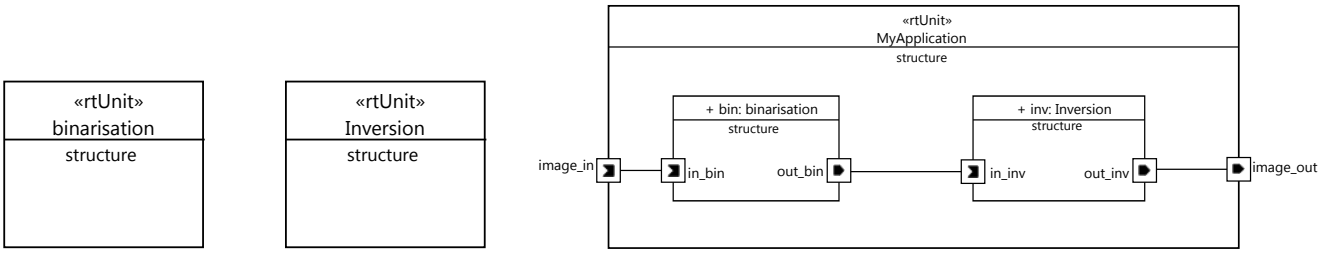
Dans les travaux de MOPCOM [94], les auteurs ont également choisi cette approche pour modéliser leur application. Cependant, la réalisation de la communication entre les tâches du modèle d'application est asynchrone. A la différence des les systèmes modélisés dans le cadre de FAMOUS, qui sont de nature synchrone.

En outre, les applications modélisées dans les travaux du projet GASPARD2 sont orientées traitement parallèle intensif. Ainsi, les composants de ces applications ne sont pas stéréotypés mais disposent de ports stéréotypés «*FlowPort*», du paquetage MARTE GCM (*Generic Component Model*). Cette description permet de spécifier le parallélisme des communications entre les

tâches. Le concept d'exécution parallèle et répétitive, introduite dans GASPARD2 par le modèle de calcul Array-OL [34], se trouve également en annexe dans le profil MARTE RSM (*Repetitive Structure Modeling*).

Étant donné que l'approche FAMOUS est plus générique, nous avons conservé les stéréotypes précités afin de modéliser l'application. Les tâches sont donc stéréotypées «*RtUnit*» et la communication s'effectue via des «*FlowPort*» afin de permettre la mise en place d'un modèle de calcul adapté à l'approche synchrone. Un exemple illustré par la figure 5.2 montre la modélisation d'une application *myApplication* qui se compose de deux tâches *Binarisation* et *Inversion*. Ces deux tâches sont stéréotypées «*RtUnit*» comme le montre la figure 5.2(a). L'instanciation des deux tâches est illustrée par la figure 5.2(b), elles communiquent via des ports d'entrée et de sortie stéréotypés «*FlowPort*».

Le concept du `FlowPort` a permis la modélisation des flux de données orientés communication entre les composants ou les messages qui écoulent à travers les ports représentent des éléments de données. La vue du méta-modèle du paquetage GCM est illustré par la figure 5.3. Un `FlowPort` spécifie les éléments d'entrée et de sortie qui peuvent écouler entre un composant structurel et son environnement. Cette spécification permet également de lui associer un ensemble de propriétés dites `FlowProperty` ayant chacune sa propre direction.



(a) Définition de «*RtUnits*» (b) Instanciation et modélisation des communications

FIGURE 5.2: Exemple de modélisation d'une application en MARTE

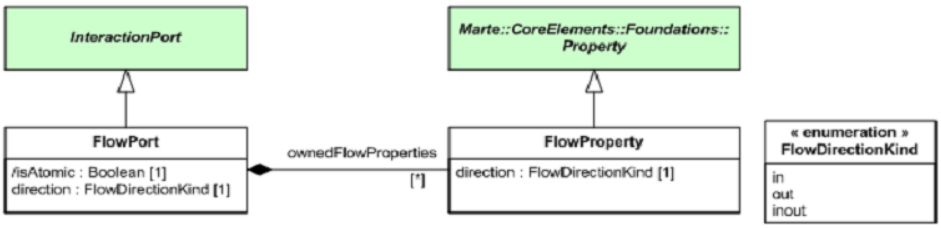


FIGURE 5.3: Vue du méta-modèle MARTE GCM autour du concept `FlowPort`

5.3.2 Représentation comportementale

En UML, la vue dynamique d'un composant est exprimée par les `diagrammes d'états`. Cette vue est plus algorithmique et orientée «traitement », elle vise à décrire l'évolution (la dynamique) des objets complexes du programme tout au long de leur cycle de vie. Ces diagrammes

indiquent en quoi les changements d'états d'un objet sont induits par des `événements`.

Il est nécessaire de spécifier dans le modèle d'application qu'une tâche peut être reconfigurable. Cependant, MARTE ne propose pas ce concept. Nous avons donc rajouté le stéréotype «*ReconfigurableRtUnit*» qui hérite du stéréotype «*RtUnit*» comme le montre la figure 5.4.

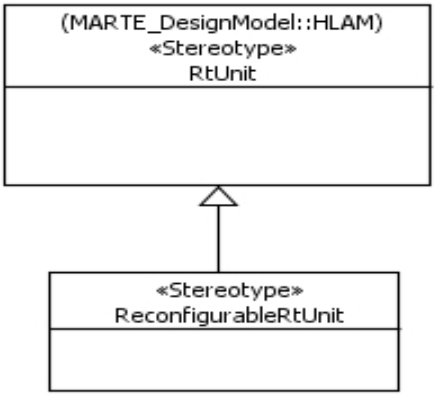


FIGURE 5.4: Le stéréotype «*ReconfigurableRtUnit*» de RecoMARTE

Ainsi, une application peut contenir des tâches statique et reconfigurable. Afin de différencier la nature des différentes tâches, il est inéluctable d'utiliser ces concepts : la tâche statique sera stéréotypée «*RtUnit*» et la tâche reconfigurable sera stéréotypée «*ReconfigurableRtUnit*». La figure 5.5 illustre un exemple où on considère que la tâche *Binarisation* est reconfigurable alors que la tâche *Inversion* est statique. Comment modéliser le comportement d'une tâche reconfigurable ?

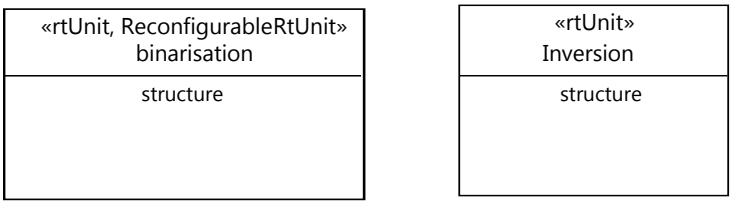


FIGURE 5.5: Modélisation de tâches statique et reconfigurable en RecoMARTE

5.3.2.1 Discussion

Certaines remarques peuvent être discutées à ce stade :

- Le stéréotype rajouté ne contient pas, pour le moment, des attributs spécifiques. Ceci n'est pas gênant car nous avons besoin d'avoir cette information pour différencier les deux types des tâches d'une application. Cette information sera utile lors de la phase d'association et du déploiement. Ainsi, ce stéréotype, tout comme d'autres stéréotypes MARTE, reste ouvert à tout ajout d'attributs (tagged values) qui peuvent le rendre plus spécifique.
- Nous aurions pu étendre directement le stéréotype «*RtUnit*» avec un attribut (tagged value) de type booléen *isReconfigurable*. Ainsi, nous pouvons spécifier lors de la modélisation

des tâches de l'application, si la tâche est reconfigurable ou non, ainsi nous gardons cette information présente au niveau modèle. Ce concept est présent au niveau du méta-modèle RecoMARTE est illustré par la figure 5.6

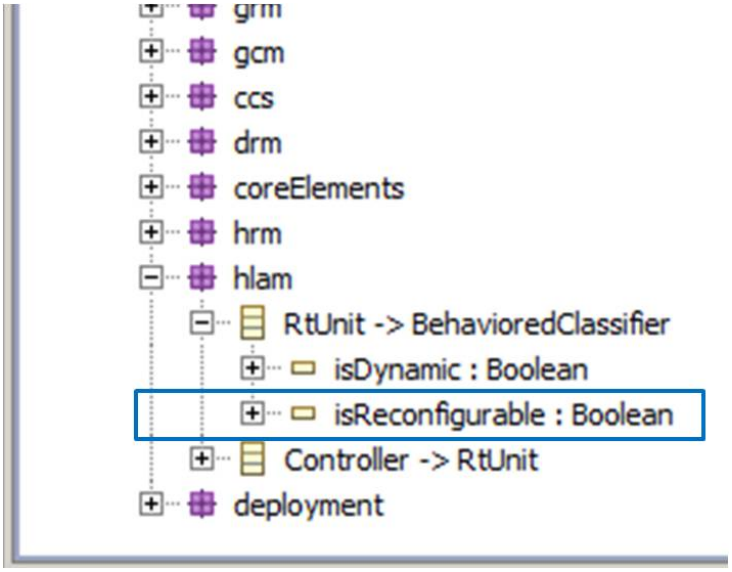


FIGURE 5.6: Extension du paquetage HLAM au niveau méta-modèle RecoMARTE

Notre choix de définir un nouveau stéréotype plus tôt qu'un attribut se justifie pour ces raisons :

- D'abord, notre objectif est de définir un nouveau profil qui étend MARTE sans pour autant toucher ou modifier les concepts de base, ce qui différencie notre travail des autres approches, exemple [41].
- Suite à un souci d'outillage, l'extension directe du profil MARTE cause d'énormes erreurs et il devient difficile d'appliquer le profil étendu sur des modèles. Ainsi, il est recommandé d'appliquer le profil MARTE en parallèle avec le profil RecoMARTE afin de garantir le bon déroulement de la modélisation et la transformation des modèles.
- Contrairement au profil, les extensions des nouveaux concepts vont étendre le méta-modèle MARTE. La définition des méta-modèles source et cible facilite l'écriture des règles de transformation de modèles (section 6).

5.3.2.2 Concepts du paquetage CoreElements

Dans une application, la description comportementale de ses tâches est une notion clé pour la spécification de la reconfiguration dynamique. Le comportement est ainsi défini par une machine à état UML qui décrit les différents états `(mode)` de la tâche. Dans la spécification MARTE, le modèle du comportement est aligné avec la sémantique de base de UML dans le sens où il n'existe aucun comportement désincarné. En effet, tout comportement (`Behavior` en anglais) émane des actions des entités structurelles. Ainsi, dans UML, un comportement est une sorte de classe, il est donc possible pour un comportement d'être, lui même, son propre

contexte structurel. La notion de comportement de base d'un système a été étendue afin de spécifier le comportement des systèmes embarqués temps réel. Il s'agit d'un type particulier lié à la notion de `operational mode`. C'est pourquoi il est nommé `modal behavior` par rapport à sa relation avec un mode.

L'ensemble des extensions des concepts de base UML pour la modélisation des comportements est représenté sous forme de stéréotypes dans un paquetage nommé «*CoreElement*» de MARTE. La figure 5.7 illustre ces extensions.

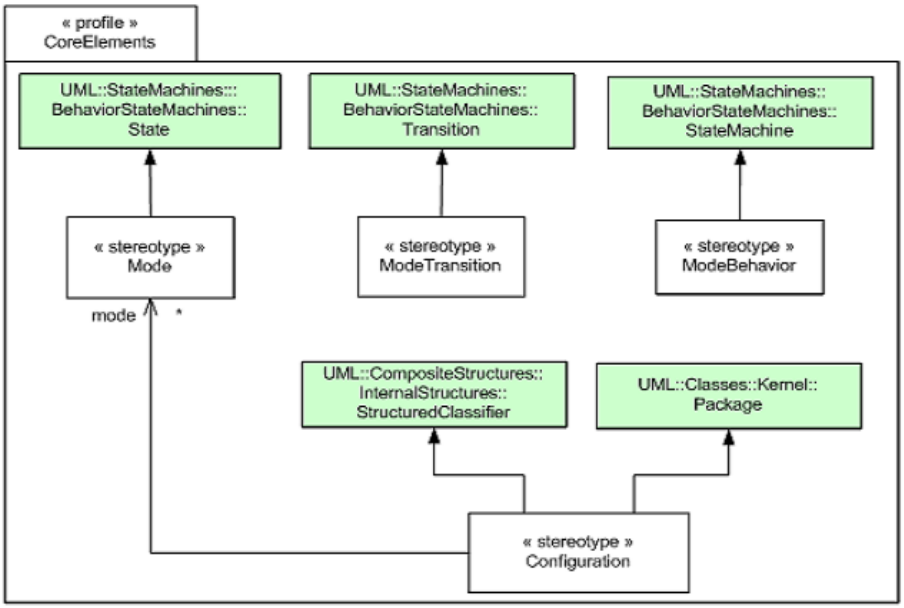


FIGURE 5.7: Vue globale du paquetage *CoreElements*

Le «*ModeBehavior*» spécifie un ensemble de modes mutuellement exclusifs. Ceci veut dire qu'à un instant donné, un seul mode peut être actif. Le «*Mode*» identifie un segment opérationnel au sein d'un système d'exécution caractérisé par une configuration donnée. Le fait qu'un *mode* soit actif implique qu'un ensemble d'entités du système sont actives durant ce fragment opérationnel.

Le stéréotype «*ModeTransition*» décrit le système modélisé sous un échange ("transition") de modes. Une transition de modes peut être produite en réponse à un déclencheur `UML::Trigger`. Ainsi, ce `Trigger` est lié à un événement `UML::Event` qui identifie les conditions qui ont causé l'action de déclenchement.

Une *Configuration* caractérise un ensemble d'entités participantes pouvant être associées à un système, un sous-système ou tout autre élément composé. Elle est représentée par le stéréotype «*Configuration*» sous forme d'une structure composite UML. Cette structure permet de représenter graphiquement des sous-composants (tâches d'une application ou des composants d'une plateforme)et leurs connexions, des valeurs de propriétés ou des paramètres définissant par exemple des facteurs de qualité de service (QoS).

Ainsi, toute implémentation effective d'une ou plusieurs tâches dans une application peut être définie grâce à ces concepts. En reprenant l'exemple ci dessus, la tâche reconfigurable

Binarisation peut être exécutée dans deux modes différents selon le seuil exigé par l'utilisateur. Le `ModeBehavior` qui contient les deux modes est représenté par la figure 5.8. Dans ce cas, quand le mode B1 est actif, la sémantique de la configuration MARTE *Config_1* impose à la tâche de Binarisation l'implémentation *bin_type1*, idem pour le deuxième mode B2, comme le montre la figure 5.9.

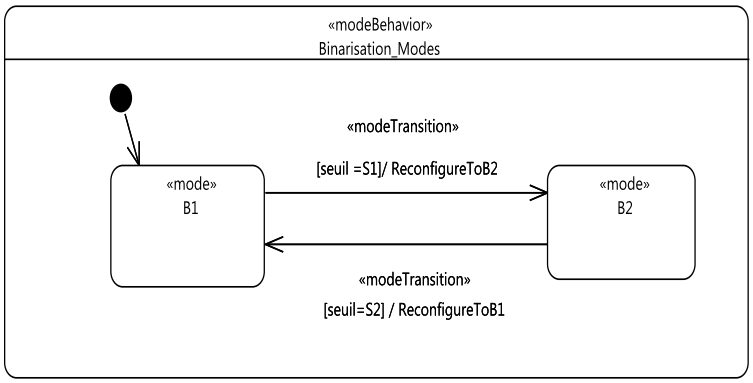


FIGURE 5.8: Exemple d'une machine à état typée `ModeBehavior`

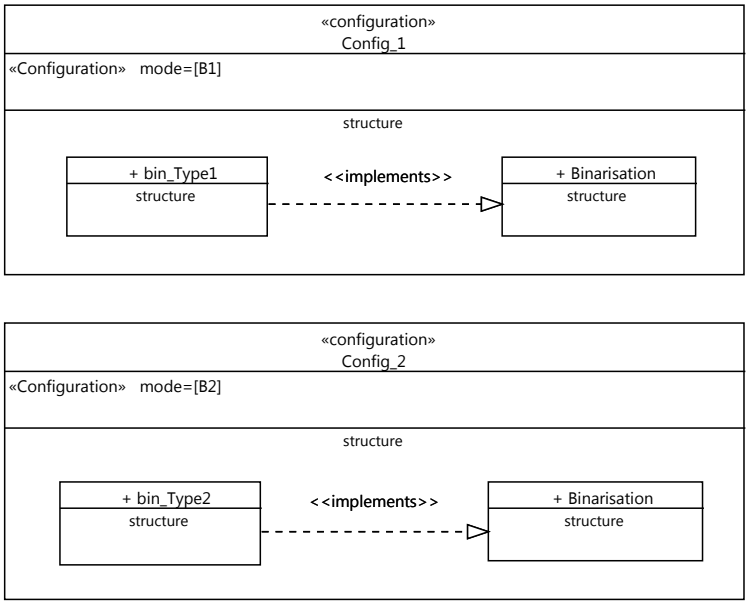


FIGURE 5.9: Exemple de modélisation des Configurations MARTE associées au ModeBehavior

5.4 Le profil HRM pour la modélisation de l'Architecture

5.4.1 La vue générale du profil

Le modèle d'architecture logique décrit un ensemble de composants reliés entre eux via des ports. La modélisation de l'architecture utilise les concepts du paquetage MARTE HRM

(Hardware Resource Modeling) illustré par la figure 5.10. Ce profil exploite particulièrement la librairie des types basiques des NFPs (Propriétés Non Fonctionnelles). Cette librairie regroupe une variété assez considérables de types complexes et ayant des unités, tels la taille de donnée `NFP_DataSize`, la durée `NFP_Duration` ou encore la puissance électrique `NFP_Power`. Ces types sont évidemment indispensables à la modélisation du matériel.

HRM est un profil UML qui regroupe la plupart des concepts décrivant les composants matériels sous une classification hiérarchique. Cette terminologie catégorise les composants selon leur nature, fonctionnalité, technologie et forme. Il regroupe ainsi le modèle logique et le modèle physique qui en font deux spécialités différentes. Ces deux sous-profils sont contenus dans le modèle général local `HwGeneral` et d'autres paquetages.

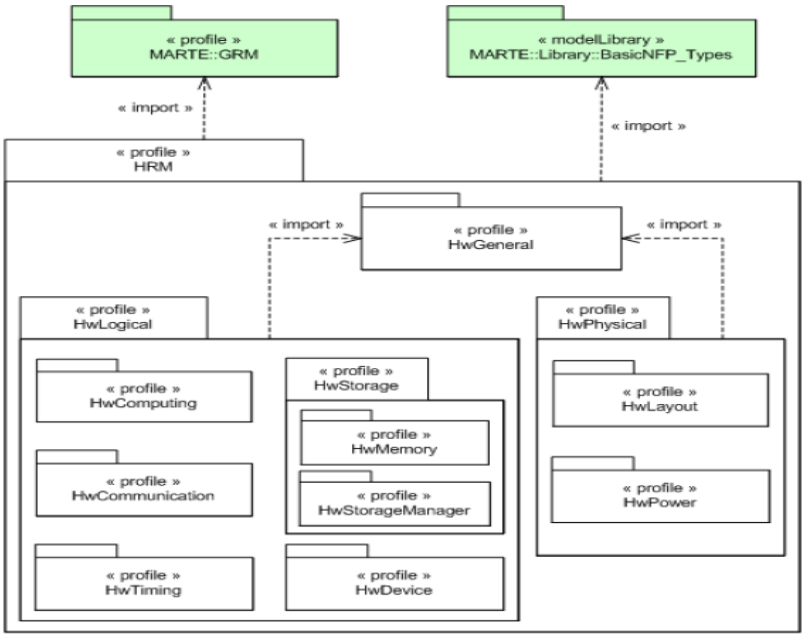


FIGURE 5.10: La structure générale du profil HRM (Hardware Resource Model) et de ses sous-profils

Le modèle général, illustré par la figure 5.11, bénéficie des extensions du profil GRM (General Resource Model) et fournit ainsi une classification fonctionnelle des ressources. Il présente ainsi une base commune pour les modèles logique et physique.

Le concept `HwResouce` est un concept générique qui décrit une entité générique du matériel. Il peut également encapsuler d'autres ressources matérielles. Ce mécanisme de composition permet des raffinements successifs selon différents niveaux de granularité. Le meilleur exemple qui illustre bien ce mécanisme de composition des ressources matérielles c'est l'FPGA. En effet, l'FPGA contient souvent de nombreux processeurs embarqués, une certaine quantité de RAM et peut également être configuré en plusieurs unités avec des fonctions différentes. Typiquement, un `HwResource` fournit un ou plusieurs `HwResourceService` et requiert certains services de la part des autres ressources.

Les sous-profils `HwPhysical` et `HwLogical` contiennent le second étage de paquetages. Ces paquetages, eux même présentés sous formes de sous-profils, sont répartis de la manière sui-

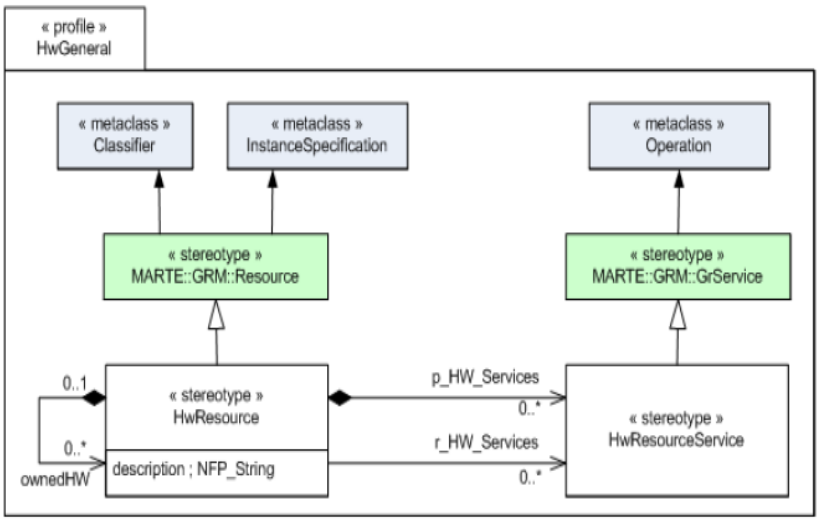


FIGURE 5.11: Les détails du profil `HwGeneral`

vante : «*HwComputing*», «*HwCommunication*», «*HwStorage*» (qui contient lui même «HwMemory» et «HwStorageManager») , «*HwTiming*» et «*HwDevice*» appartiennent au sous-profil logique alors que «*HwLayout*» et «*HwPower*» sont du côté du sous-profil physique.

5.4.2 Le sous-profil logique

Afin de modéliser l'architecture logique de notre système, nous faisons appel aux stéréotypes du sous-profil `HwLogical`. Nous utiliserons donc les concepts physiques plus loin dans la description de la plateforme physique (section 5.4.5).

Le but du modèle logique est de fournir une classification fonctionnelle des entités matérielles. Cette classification doit distinguer entre les différents types de ressources utilisées pour le calcul, le stockage mémoire, la communication, le temps et les ressources auxiliaires. Elle est principalement basée sur les services que chaque ressource offre et peut éventuellement prendre compte leur nature.

La taxonomie logique est commune à de nombreux travaux antérieurs [58, 26]. Elle n'est pas catégorique et les concepts appartenant au sous-profil logique, ne sont pas nécessairement incompatibles. Une ressource matérielle pourrait avoir de nombreuses fonctions au sein de la même plate-forme matérielle. Tel est le cas d'un DMA (Direct Memory Access) qui peut gérer la mémoire et participer également au contrôle de la communication entre les ressources.

5.4.3 Modélisation d'un exemple

La figure 5.12 illustre un exemple d'un modèle d'architecture logique qui se compose d'un accélérateur, un processeur, un bus et un contrôleur de mémoire de données. Tous ces composants sont reliés entre eux via des ports.

Les stéréotypes utilisés respectivement sont :

– «*HwComputingResource*» : il s'agit d'un concept générique à haut niveau décrivant une

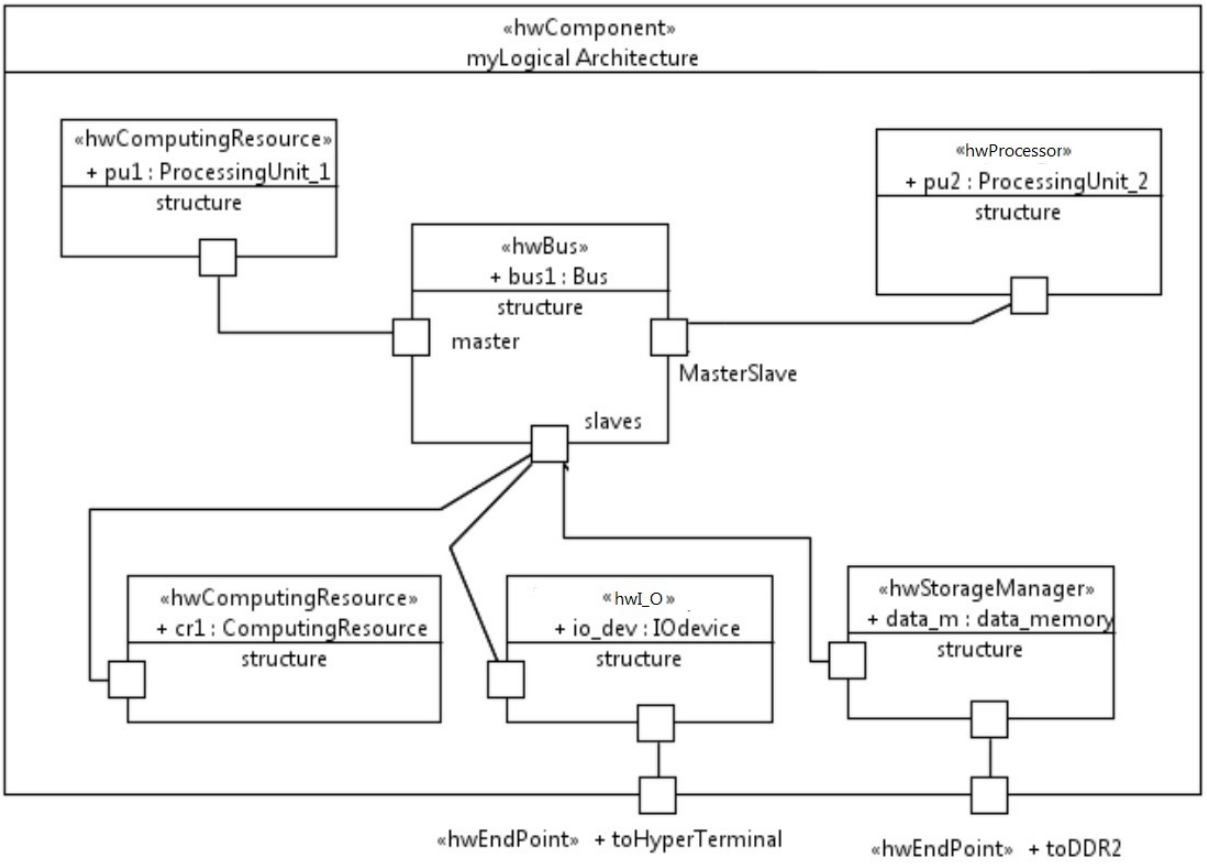


FIGURE 5.12: Modélisation d'une architecture logique avec MARTE HRM

ressource active d'exécution, exemple : Xilinx ICAP (Internal Configuration Access Port).

- «*HwProcessor*» : représente un processeur qui implémente au moins un jeu d'instructions. Il peut contenir des unités de gestion de mémoire et du cache organisé selon des catégories et des niveaux. Le processeur Softcore Xilinx MicroBlaze en est un exemple.
- «*HwBus*» : est un type particulier du `Hw_Media` du paquetage `Hw_Communication`. Un bus est caractérisé par ses adresses et la largeur du mot transporté. Il peut être synchrone ou non, bus série ou parallèle, tel est l'exemple du bus Xilinx PLB (Processor Local Bus).
- «*Hw_IO*» (Input/Output) est une ressource générique qui représente tout composant qui interagit avec l'environnement externe à la plateforme. Il peut interagir dans un sens unique, soit de l'environnement vers le composant (exemple : caméra), soit dans le sens inverse (exemple : un haut-parleur) ou bien à double sens à l'exemple d'un UART (Universal Asynchronous Receiver Transmitter).
- «*HwStorageManager*» : dénote un contrôleur mémoire qui gère l'accès et/ou le contenu des mémoires qu'il contrôle. Le contrôleur de mémoire DDR2 en est un exemple.
- «*HwEndPoint*» : utilisé pour certains ports de l'architecture qui se connectent à l'extrémité d'un média : *ToHyperTerminal* et *ToDDR2*. Ce concept symbolise donc un point de connexion d'une `HwResource`.

A ce niveau d'abstraction, seuls les composants ainsi que leur connexion sont définis. Les caractéristiques spécifiques à chaque composant seront identifiées au niveau suivant qui est le

déploiement. A ce moment, les IPs seront également attribuées à chaque composant.

5.4.4 Extension du concept FlowPort du paquetage GCM

Le concept FlowPort de MARTE est utilisé pour modéliser les ports des composants élémentaires au niveau de l'application et de l'architecture. Ce concept étend la méta-classe UML : :Port et a été illustré précédemment par la figure 5.3.

Au niveau de la modélisation de l'architecture, il est nécessaire de spécifier le type de port qui relie les composants aux autres composants ou à l'environnement extérieur de l'FPGA. Par conséquent, les ports portent un nom et certaines informations sur le type de port afin d'établir une distinction entre les types de connexions. Cette information est utilisée lors de la transformation des modèles MARTE vers une description IP-XACT ainsi que les transformations de IP-XACT vers le fichier MHS.. La figure 5.13 présente la définition d'un nouveau stéréotype «*ExtendedFlowPort*» qui étend le stéréotype «*FlowPort*» du paquetage GCM et ayant l'attribut portKind de type énumération. La figure 5.14 montre un extrait du méta-modèle GCM avec l'extension proposée. Les types de port sont cités sous forme d'une énumération *PortKind* :

- un port peut être de type *singlePort* pour spécifier une connexion point à point
- le type *busInterface* est choisi si le composant élémentaire correspondant à ce port est relié à un bus, comme le port qui relie le composant *pu_1* au bus.
- et enfin le port de type *ioInterface* est utilisé lors d'une connexion entre le composant élémentaire et les ports d'entrée/sortie (inout/output) de l'FPGA, tel est l'exemple du port qui relie le composant *io_device* à un hyper terminal.

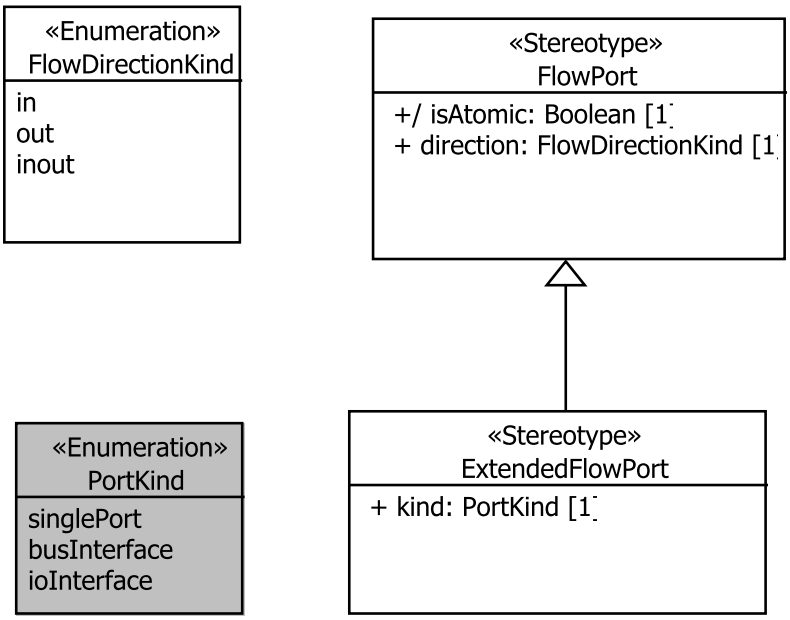


FIGURE 5.13: Ajout du nouveau stéréotype qui étend «*FlowPort*»

La figure 5.15 illustre un exemple d'une connexion d'un composant élémentaire avec le port IO de l'FPGA via un port de type *ioInterface*. Cet exemple est une vue partielle du diagramme

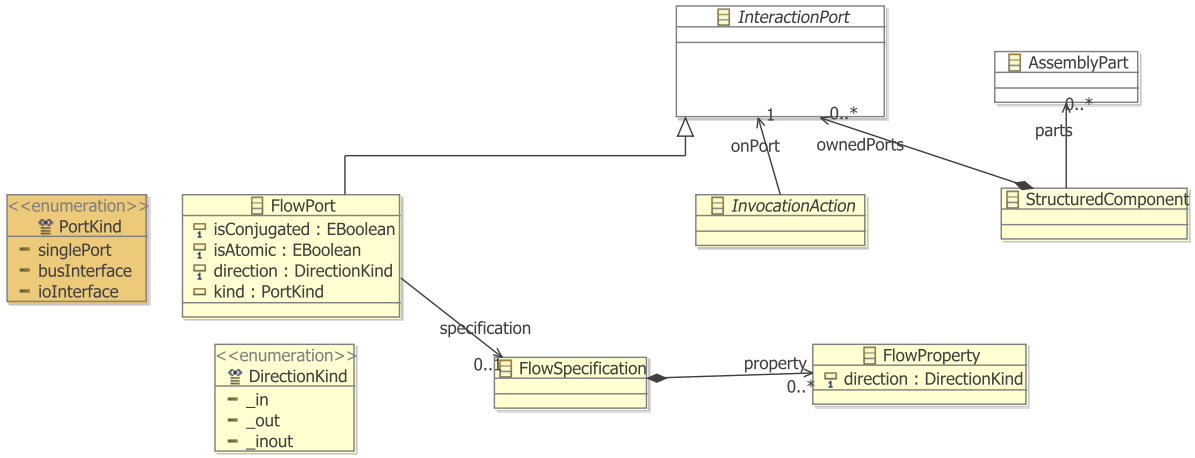


FIGURE 5.14: Extrait du méta-modèle du paquetage GCM avec extension du concept FlowPort

UML modélisant l'architecture logique présentée précédemment.

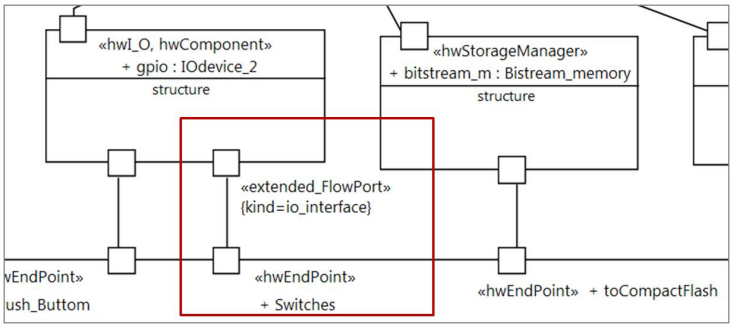


FIGURE 5.15: Exemple d'utilisation de l'extension du FlowPort

5.4.5 Extension du sous profil physique pour la modélisation de la plateforme

Le sous-profil physique a pour objectif de fournir les mécanismes permettant de projeter les composants matériels sur la plateforme physique. Généralement, le modèle physique représente les ressources matérielles en mettant en valeur leurs propriétés physiques telles que la surface, le poids, le coût, etc. Dans MARTE HRM, le sous-profil physique contient deux paquetages `HwLayout` et `HwPower`. Le premier permet d'identifier les ressources du matériel selon leurs formes, leurs dimensions, leurs positions dans la plateforme ainsi que leurs conditions environnementales requises. Alors que le deuxième paquetage décrit ces ressources selon leurs consommations d'énergie, leurs dissipations de chaleurs et leurs radiations électromagnétiques.

Dans ce qui suit, nous nous intéressons au paquetage HwLayout pour la modélisation de la plateforme physique de l'FPGA.

5.4.5.1 Présentation du package HwLayout

Il s'agit d'un paquetage qui permet de classifier les composants matériels selon leurs formes, et les arranger dans des grilles rectilignes. Cependant, la plateforme physique de l'FPGA

est agencée selon des colonnes de ressources physiques hétérogènes. Ainsi, les composants matériels ne peuvent pas être placés dans des grilles rectilignes contiguës. Pour ce faire, nous introduisons, dans la suite, des extensions à ce paquetage pour nous permettre d'agencer et placer les composants.

Le concept principal de ce paquetage est `HwComponent` qui, d'après la figure 5.16, est une spécialisation de la métaclasse `HwResouce`, précédemment présentée dans le modèle général. HwComponent dispose de plusieurs propriétés physiques, peut être composite et peut également prendre plusieurs formes : puce, carte, port, etc.

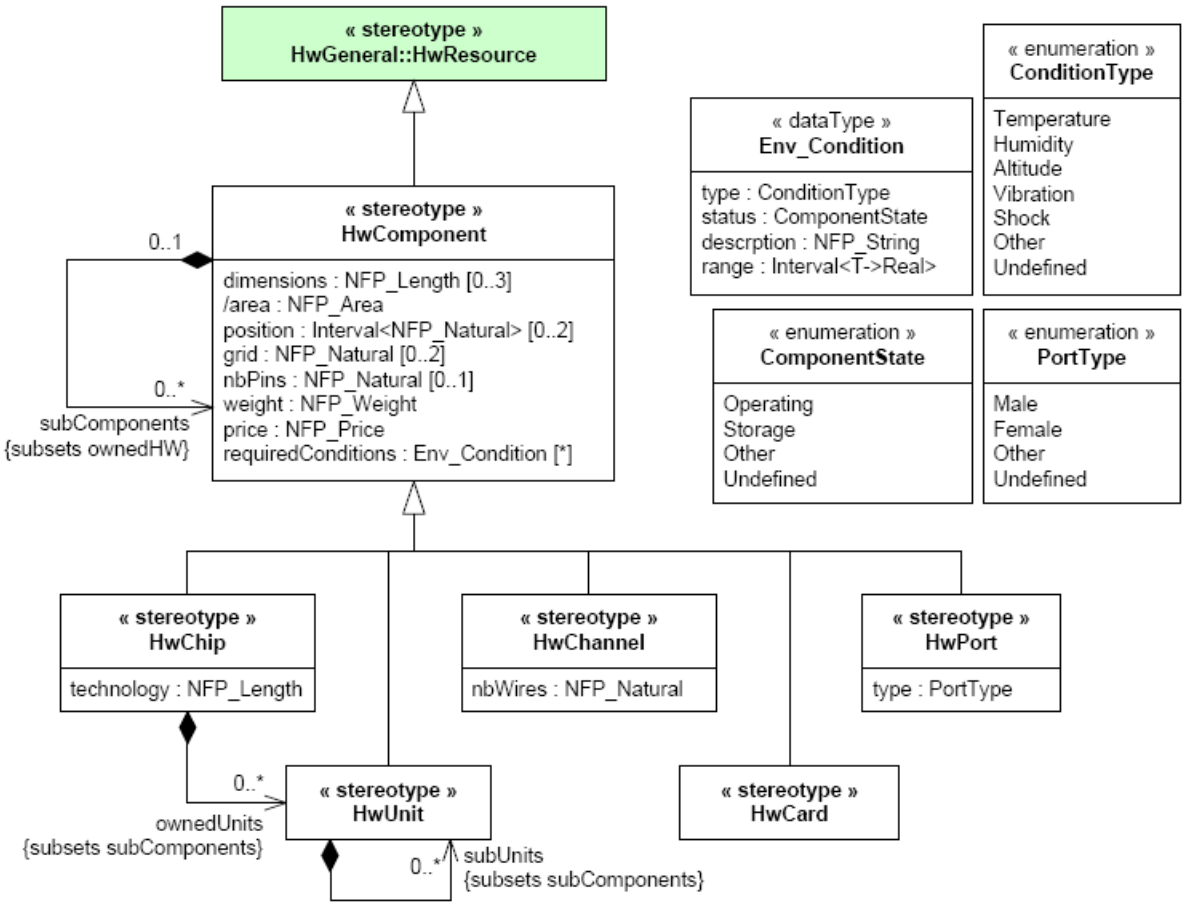


FIGURE 5.16: Le modèle de disposition dit HwLayout

5.4.5.2 *Modélisation des régions physiques*

La modélisation de la plateforme physique permet de réduire le fossé entre les diagrammes UML et l'agencement physique et la topologie de l'architecture ciblée. A ce niveau de conception, on se rapproche du niveau matériel et la modélisation devient plus dépendante de la technologie. Le fait de décrire le placement des différentes régions physiques, statique ou reconfigurable, veut dire que le concepteur a bien choisi le type de l'FPGA qu'il va utiliser.

Lors de la phase de conception de la plate-forme matérielle, il est nécessaire de définir une unique région statique et une (ou plusieurs) région(s) reconfigurable(s). Plus la région

reconfigurable est petite, plus l'overhead (en termes de temps de reconfiguration, surcoût de consommation) de la reconfiguration sera limité sur le fonctionnement du système embarqué. Les différentes architectures matérielles pouvant être chargées dans une même zone reconfigurable sont stockées sous forme de fichiers binaires (bitstream en anglais).

Rappelons que les composants dits *reconfigurables* sont les composants sur lesquels les tâches reconfigurables sont exécutées. En outre, pour chaque composant de l'architecture, un fichier UCF est créé. Ce fichier contient principalement des contraintes ainsi que des informations sur le placement (les coordonnées) des régions reconfigurables (RR). Ces contraintes affectent la manière d'implémenter le modèle logique dans la plateforme cible. Ce fichier peut être modifié par le concepteur afin d'améliorer ou de personnaliser les contraintes lors de la phase initiale de conception.

Afin de modéliser les régions physiques de l'FPGA, nous introduisons la notion de `HwRegion` dans le sous-profil physique du paquetage HRM, illustrée par la figure 5.17.

Une HwRegion peut être statique ou reconfigurable, d'où la définition des stéréotypes «*HwStaticRegion*» et «*HwReconfigurableRegion*».

Par définition [106], la `logique statique` est tout élément de l'FPGA qui ne fait pas partie de la partition reconfigurable. Ces éléments ne peuvent jamais être reconfigurés et sont toujours actifs lors de la reconfiguration des partitions reconfigurables.

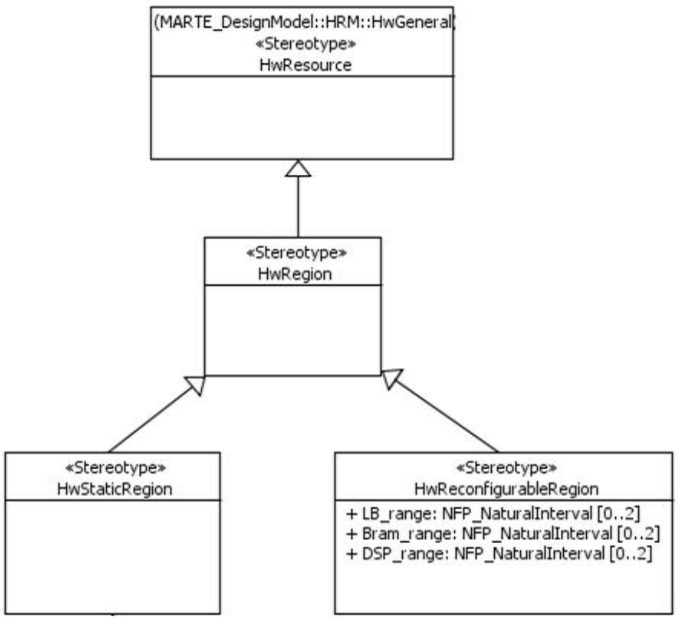


FIGURE 5.17: Définition des concepts dans le profil RecoMARTE pour la modélisation des régions physiques de l'FPGA

En choisissant la technologie de l'FPGA, le concepteur identifie les différents composants qui seront placés dans la région statique. Certaines technologies imposent l'utilisation de quelques composants. Prenons l'exemple de Xilinx Virtex-4 [102], cet FPGA dispose déjà d'un processeur hardcore dit "PowerPC" qui est déjà embarqué dans le circuit, figé et ne peut pas être supprimé mais peut être désactivé par l'utilisateur, lors de la création du fichier système.

De la même manière que le concepteur définisse les composants qu'il placera dans la région statique, il identifie également les caractéristiques des régions reconfigurables. Le cas d'utilisation le plus connu de la reconfiguration dynamique et partielle dans le domaine des FPGAs, est basé sur des co-processeurs ou des accélérateurs matériels reconfigurables. En effet, l'utilisation des co-processeurs et des accélérateurs matériels reconfigurables permet d'économiser l'espace sur l'FPGA en comparaison avec des solutions qui implémentent toute une application avec des modules statiques.

Ainsi, selon les besoins requis par l'application ou exigés par l'utilisateur, le nombre de composants reconfigurables est défini et par conséquent le nombre des régions reconfigurables (RRs). Une RR est caractérisée par sa forme rectangulaire et est identifiée par son placement sur la plateforme de l'FPGA selon des coordonnées. Ces coordonnées sont définies en terme de certaines ressources physiques qui sont : *LB : Logic Blocs* (Blocs logiques nommés CLB (Configurable Logic Block) selon le constructeur Xilinx ou LE (Logic Element) selon Altera), des *BRAMs : Blocs de mémoire RAM* et des *DSPs :Digital Signal Processing*. Ces différents blocs sont alignés sous forme de colonnes ainsi le placement des RRs se fait selon des "Ranges". Pour ce faire, nous définissons des tagged values au sein du stéréotype «*HwReconfigurableRegion*» pour définir des coordonnées de deux points qui limitent la zone rectangulaire selon le repère des LBs (*LB_range*), des BRAMs et des DSPs. Le type de ces attributs est *NFP_NaturalInterval* [0..2] comme le montre la figure 5.17.

Un extrait du fichier UCF est présenté par la figure 5.18 montrant la syntaxe utilisée pour définir les coordonnées des RRs.

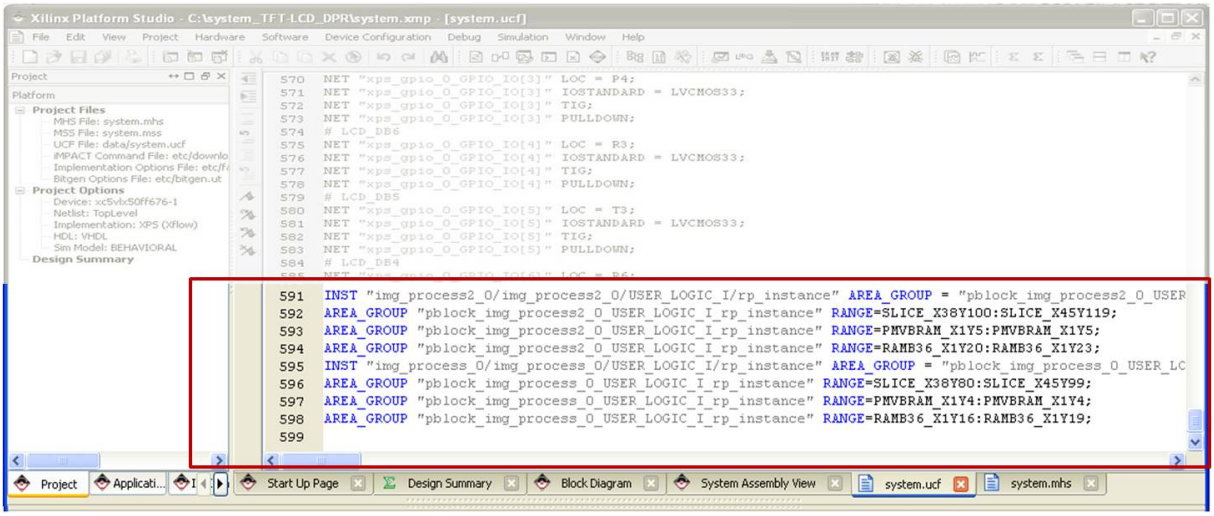


FIGURE 5.18: Extrait du fichier UCF pour le placement des RRs

Deux cas de figures s'offrent au concepteur afin de récupérer cette information.
- S'il dispose d'un certain niveau de connaissances sur la manipulation des plateformes FPGA ;
- S'il a déjà synthétisé et simulé un projet avec les outils Xilinx, il peut ainsi effectuer une exploration de l'espace de conception dirigée par les modèles (Figure 5.19) afin de récupérer ces informations.

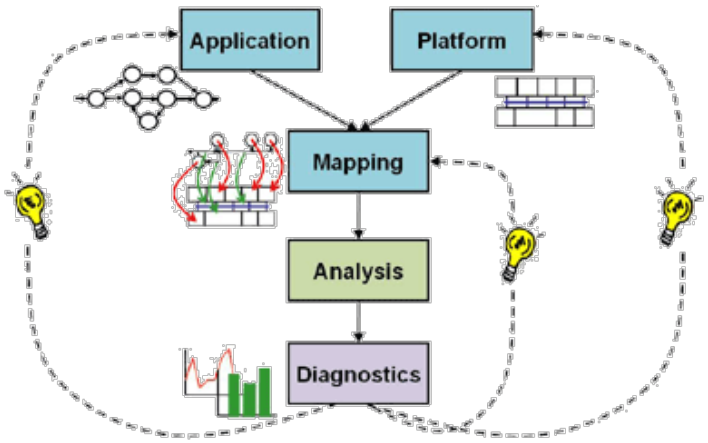


FIGURE 5.19: La DSE dirigée par les modèles [13]

5.4.5.3 *Placement des ports physiques*

Après avoir défini les ports logiques qui permettent de connecter les composants élémentaires de l'architecture logique, il est nécessaire de modéliser alors les ports physiques de la plateforme de FPGA. Ces ports définissent les entrées/sorties de l'FPGA et leurs noms sont enregistrés dans le fichier de contraintes UCF, comme le montre l'extrait de la figure 5.20.

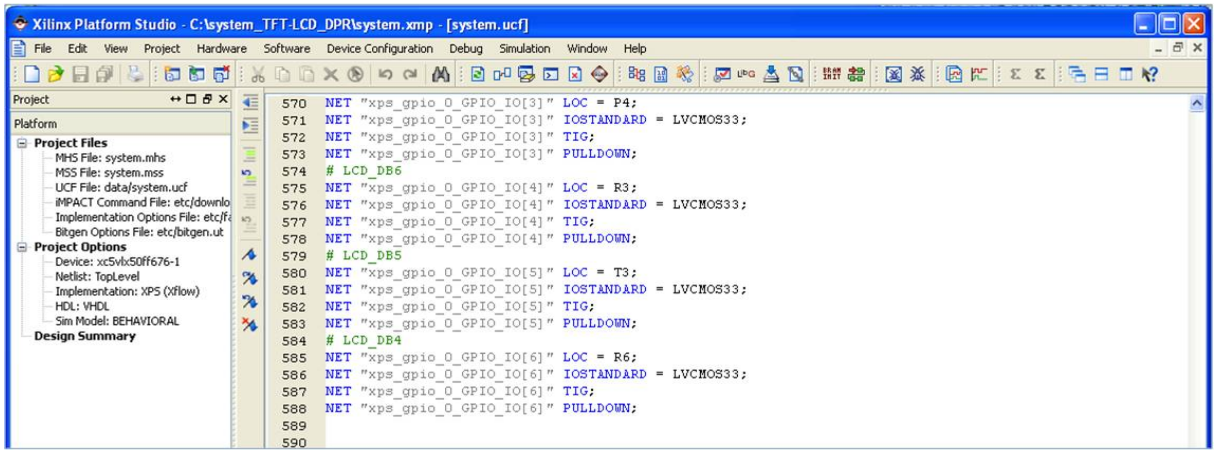


FIGURE 5.20: Extrait du fichier UCF pour le placement des ports physiques

Deux cas de figures se présentent :

- Si le concepteur souhaite placer un composant (exemple : Uart) qui a été automatiquement généré par l'outil, alors son placement physique est automatique et pas besoin d'intervention manuelle.
- Dans le cas où il a besoin d'ajouter un composant qui sera relié, via ses ports, à l'environnement extérieur de l'FPGA, alors il est nécessaire de spécifier les différents pins alloués pour le port I/O de ce composant. Ensuite, l'outil de génération de code se charge de générer des instructions dans le fichier UCF. Ces instructions correspondent aux différents noms des pins attribués dans le modèle d'architecture physique.

Le stéréotype qui permet de modéliser les ports physiques est «*HwPort*» du sous-profil

HwPhysical. Les ports physiques peuvent être définis manuellement sans passer par l'outil. Le concepteur doit juste saisir le nom du(des) pin(s) à allouer aux ports concernés. Pour ce faire, nous rajoutons l'attribut *allocatedPins* de type NFP_String afin de saisir les noms des pins nécessaires au placement du port logique I/O sur le port physique de l'FPGA. Cette extension est illustrée par la figure 5.21.

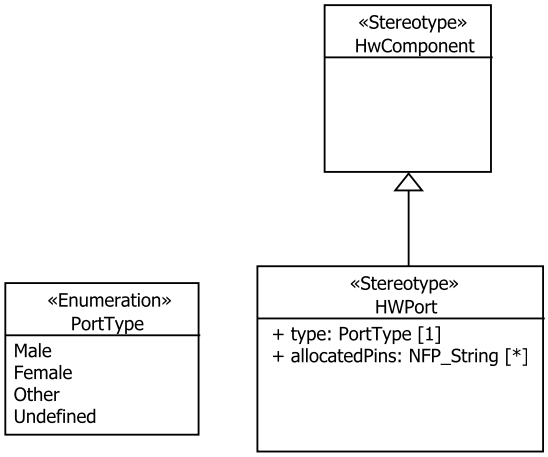


FIGURE 5.21: Extension du stéréotype «*HwPort*» pour le placement des ports physiques

5.4.5.4 Sauvegarde du contexte et modélisation de membrane

Dans le monde du logiciel, les applications auto-adaptatives peuvent modifier leur comportement de manière dynamique et autonome. Ceci est possible en utilisant l'introspection, la recomposition, l'ajout ou la suppression de composants. Le but étant de s'adapter aux changements qui peuvent avoir lieu au sein de leur environnement d'exécution [83].

Un composant est vu comme une boite noire qui peut être changée et remplacée par une autre, ayant les mêmes interfaces. Ceci est une notion clé à suivre dans le monde du matériel. En effet, ce travail constitue un autre axe de recherche intégré dans le projet FAMOUS. L'objectif de ce travail [97] est d'améliorer la productivité de la conception et d'assurer la cohérence en gérant les changements de contexte et de stockage modulaire en utilisant des contrôleurs matériels distribués.

Le support qui en résulte est un support matériel pour gérer la reconfiguration dynamique partielle ainsi que la communication. Ce support assure le changement et la sauvegarde de contexte au moment de l'exécution de la reconfiguration du système.

Ainsi, le modèle proposé est flexible (réutilisé par plusieurs applications), modulaire et générique. Il est inspiré d'une approche assez connue dans le domaine du développement logiciel : les modèles basés sur les composants. De plus, ce modèle sera implémenté sous forme d'une membrane matérielle connectant les IPs au reste du système.

En conséquence, le point d'entrée avant l'implémentation de la membrane est une modélisation RecoMARTE. La membrane est considérée comme une région physique appartenant

à la partie statique de l'FPGA. Chaque membrane est attachée à un IP (bitstream) à un moment donné. Cependant, comme tout IP est susceptible d'être remplacé par un autre, alors des sauvegardes régulières du contexte de la tâche concernée sont effectuées.

La figure 5.22 illustre le nouveau concept de la membrane rajouté dans le profil RecoMARTE.

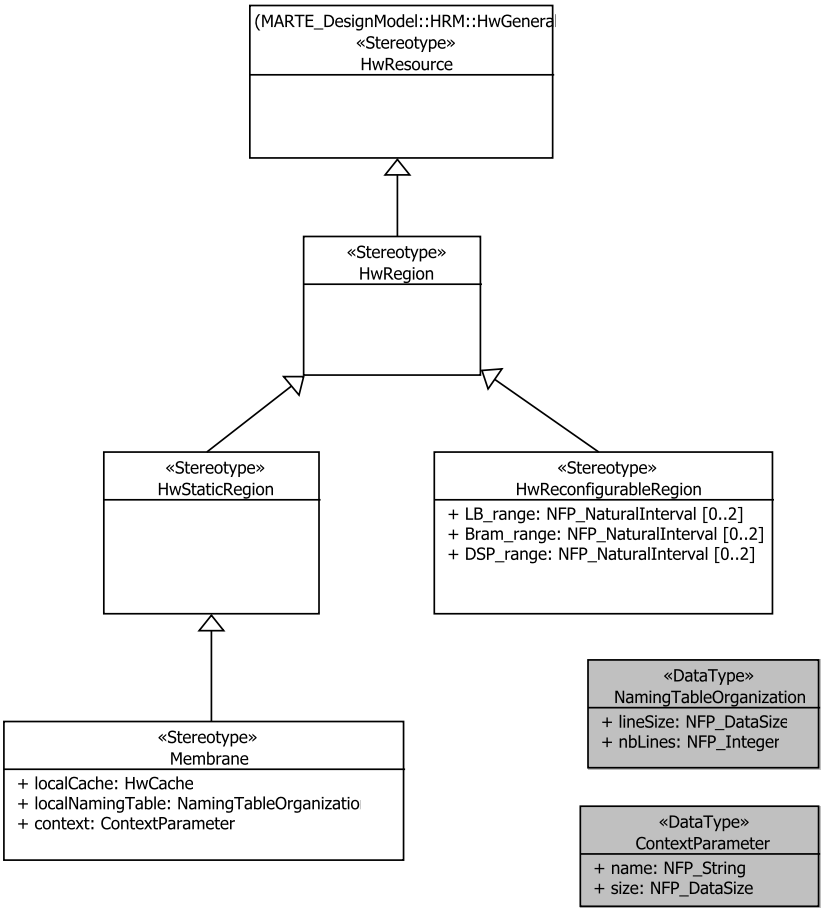


FIGURE 5.22: Définition du concept Membrane pour la sauvegarde du contexte dans le profil RecoMARTE

Puisqu'il s'agit d'une région physique statique, alors le stéréotype «*Membrane*» hérite du stéréotype «*HwStaticRegion*» du sous-profil physique de HRM. Les informations nécessaires à la définition de la membrane sont présentes sous forme de tagged values :

- *localCache* : utilisé afin d'enregistrer fréquemment les données utilisées pour un accès rapide. Il est typé *HwCache* (Cf. Figure hwcache), ce qui lui permet de varier selon son niveau, son type et sa structure.
- *Local Naming Table* : il s'agit d'une table de nommage locale à la membrane. C'est un ensemble de lignes de bits organisées selon le DataType *NamingTableOrganization* qui détermine 1)le nombre de lignes et 2) la taille de chaque ligne.
- *Context* : chaque contexte est défini par une paire de paramètres spécifiques (nom ; taille). Ceci permet de relier chaque donnée du contexte à son IP qui lui correspond.

5.4.6 La vue générale du méta-modèle HRM étendu

Après avoir présenté en détails tous les concepts ajoutés au paquetage HRM, nous illustrons par la figure 5.23 la vue générale du méta-modèle HRM avec les extensions.

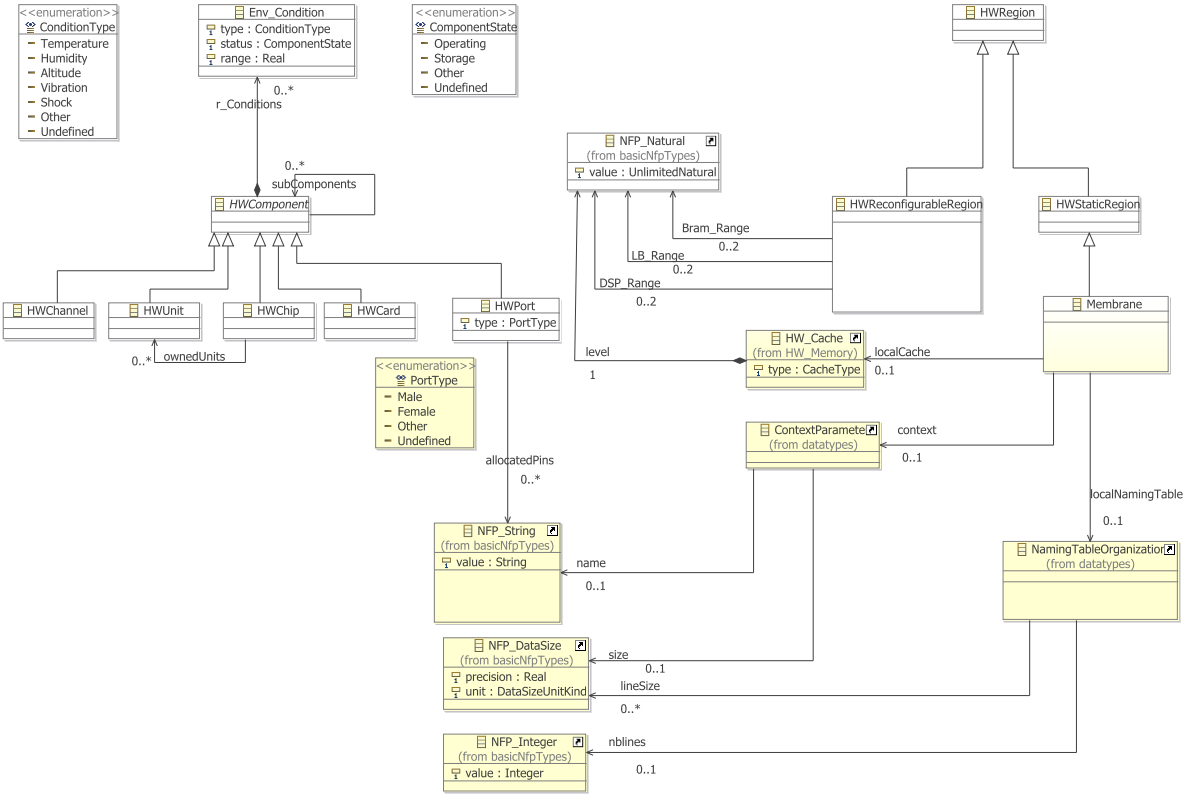


FIGURE 5.23: Vue générale du méta-modèle physique du paquetage HRM avec les extensions

5.5 Le profil du Déploiement

5.5.1 Allocation dans MARTE

Le concept d'allocation dans MARTE, permet au concepteur d'établir un lien entre une application et une architecture utilisée comme un support d'exécution. En effet, le concepteur peut spécifier l'allocation d'une tâche sur une ressource de calcul (exemple : processeur). Ce concept présente le souci majeur de la conception des systèmes embarqués temps-réel.

L'allocation peut être spécifiée selon trois types différents : structurel (`Structural`), comportemental (`behavioral`) ou hybride (`hybrid`). L'allocation structurelle représente une association entre un groupe d'éléments structurels et un groupe de ressources. Alors que l'allocation de type comportemental est une association entre un ensemble d'éléments comportementaux et un service fourni par la plateforme d'exécution. Enfin, l'allocation hybride peut avoir lieu dans le cas où un service implicite est défini de manière unique pour une ressource. Plus loin dans un niveau de détails plus raffiné, l'allocation comportementale traite le `mapping` entre les actions de type UML et des ressources et services.

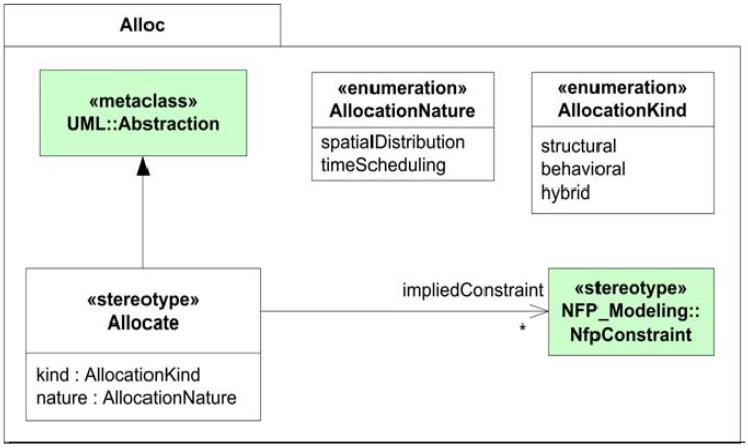


FIGURE 5.24: Le stéréotype «*Allocate*» du profil MARTE

Le stéréotype «*Allocate*» est une dépendance (UML `dependency`) basée sur `UML::Abstraction`. Il s'agit d'un mécanisme permettant d'associer des éléments d'un contexte logique ou des éléments du modèle d'application, à des éléments décrits dans un contexte plutôt physique, ou un modèle de la plateforme d'exécution. La dépendance de l'allocation peut être utilisée soit, pour spécifier une seule allocation possible et dans ce cas un outil de DSE (Design Space Exploration) peut déterminer la meilleure allocation, soit pour spécifier une allocation effective dans le système. En outre, plusieurs types de contraintes UML peuvent être appliquées sur cette dépendance y compris les contraintes typées «*NFP_Constraint*» du paquetage NFP (Non-functional Properties).

L'association nommée "*impliedConstraint*" a pour but d'identifier explicitement les contraintes qui s'appliquent uniquement si (ou quand) l'allocation est effectuée. Si ce n'est pas le cas, les types de contraintes peuvent aider à déterminer si l'allocation est requise, offerte, etc.

Le modèle d'allocation pourrait offrir différentes alternatives d'allocation pour une application donnée de sorte qu'il ait un choix réel sur la manière d'allouer les fonctions (tâches) de l'application sur les différentes parties de l'architecture. Par la suite, le `mapping` peut être raffiné avec plus de précision lors des transformations de modèles réalisées par les techniques d'analyse. La nature de l'allocation comprend à la fois la distribution spatiale et l'ordonnancement temporel dans le but de lier différentes opérations algorithmiques aux services et ressources de calcul et de communication correspondantes.

Les allocations spatiale et temporelle doivent être mutuellement et globalement cohérentes afin d'assurer l'exécution correcte de l'application par son déploiement sur l'architecture d'exécution.

Les différents aspects de l'application et de l'architecture sont représentés en se basant sur le concept composant. Cependant, cette représentation est dépauvrie de toute description interne. Il s'agit d'un modèle de haut niveau qui ne contient pas les informations nécessaires à la génération du modèle exécutable de bas niveau. D'où le besoin d'associer chaque composant à une implémentation déjà existante sous forme de code, dite Intellectual Property (IP). Ce

niveau de modélisation, absent dans MARTE est utilisé dans UML et bien d'autres travaux comme GASPARD2 et il est appelé `Déploiement`. Étant donné que nous sommes contraints à utiliser le standard MARTE dans notre projet, nous définissons alors un niveau de modélisation intermédiaire que nous appelons `Allocation déployée`. Cette phase intermédiaire nous permet d'exprimer les liens entre la modélisation à haut niveau d'abstraction et les IPs.

Dans cette thèse, nous avons proposé un ensemble de concepts afin d'exprimer cette phase ce qui étend le profil MARTE en introduisant le paquetage `Deployment`. Notre objectif est de fusionner la notion d'allocation de MARTE avec la notion de déploiement. Ainsi, l'allocation est effectuée sur des modèles déployés, dont les composants sont associés à des IPs.

L'allocation représente soit une allocation possible, dans ce cas, un outil d'exploration spatiale peut déterminer quelle est la meilleure allocation, ou bien une allocation effective dans le système. Le contexte dans lequel la dépendance *allocate* est utilisée doit être suffisante pour savoir dans quel cas nous sommes. Quand ce n'est pas le cas, le genre de contraintes peut aider à déterminer si l'allocation est nécessaire, offerte, etc.

Le paquetage Deployment proposé est disponible sous forme de profil et de méta-modèle. Le profil est utilisé pour la conception via les outils de modélisation UML alors que le méta-modèle est utile pour la phase de transformations de modèles. Dans ce qui suit, nous utilisons plutôt le profil afin de bénéficier d'une représentation graphique des exemples.

Les principales contributions de ce paquetage sont les suivantes :

- *La réutilisation des IPs* : qui correspondent aux différents composants du système. Réutiliser des IPs logicielles ou matérielles est très important pour la productivité de la conception. En effet, lors du développement d'un nouvel FPGA (éventuellement un SoC), la plupart des blocs de construction élémentaires ne sont pas développés ou créés de zéro mais peuvent être acquis auprès de tiers, responsables de la vérification des IPs.
- *Création des bibliothèques d'IPs* : ceci doit être possible à ce niveau d'abstraction. L'ajout de ce paquetage dans le standard MARTE accorde une attention particulière aux différentes sources d'acquisition d'IP. En effet, l'utilisation des IPs provenant d'une bibliothèque doit être simple et intuitive, au mieux possible. De même pour les IPs qui sont décrites indépendamment à partir d'un modèle FPGA spécifique. En particulier, l'évolution dans la bibliothèque d'IPs doit être indépendante du modèle de l'FPGA. En outre, cette bibliothèque fournit une distinction claire entre les IPs logiciels et matériels selon les niveaux d'abstraction et les différentes technologies. Dans nos travaux, cette bibliothèque est décrite selon le standard IP-XACT du consortium SPIRIT (qui sera présenté plus loin dans le chapitre 6.
- *Abstraction des fonctionnalités associées* : tout composant lié à un IP est une abstraction de la fonctionnalité relative à l'IP. Cependant, cet IP est spécifique à une plateforme (technologie) cible spécifique. Par exemple, pour une simulation ciblée au niveau TLM SystemC, il est préférable d'utiliser un composant matériel auquel on associe une IP du SystemC TLM. Par contre, pour cibler un niveau plus détaillé tel que le RTL, il est recommandé de choisir un IP écrit en langage HDL (VHDL).

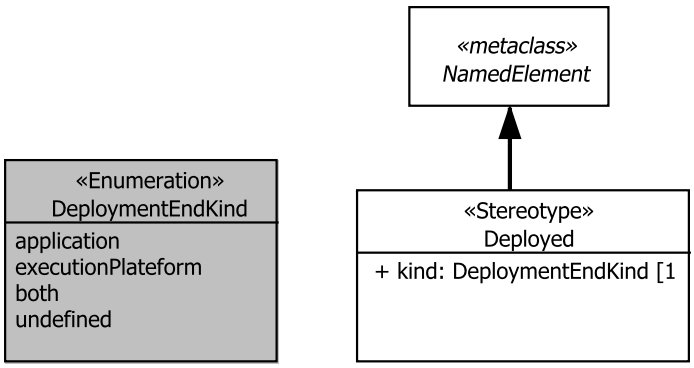


FIGURE 5.25: Le stéréotype «*Deployed*» du profil RecoMARTE

– *Le choix de l'implémentation de l'IP* : spécifique à une cible de compilation donnée. Il est possible de décrire l'IP selon plusieurs langages de programmation (C, vhdl, etc.) et des les classer en IP software ou hardware.
– *Enrichir le modèle* : avec les informations nécessaires pour que l'intégration de ces IPs soit automatique lors de la génération de code et la compilation.
– *Extension du profil MARTE :* Ce paquetage étend le profil MARTE afin de compléter le flot depuis la conception des modèles jusqu'à la génération de code. Ce paquetage est défini donc dans le nouveau profil étendu RecoMARTE

Le paquetage Deployment comporte principalement trois nouveaux concepts (sous forme de stéréotypes au niveau du profil) complémentaires qui sont «Deployed», «IP» et «CodeFile», que nous détaillons dans ce qui suit.

5.5.2 Le stéréotype *Deployed*

La première étape consiste à identifier ce qui peut être déployé à savoir un composant, un comportement ou une structure. Ensuite, il faut identifier la cible du déploiement, la vue matérielle qui peut être une ressource ou un service. Le stéréotype «*Deployed*» (figure 5.25) est utilisé pour ce faire dans le diagramme de déploiement des composants. Ainsi, chaque composant du modèle d'application ou d'architecture sera stéréotypé «*deployed*» et associé à un IP. Le tagged value *kind* identifie le type du composant déployé ; qu'il soit une tâche d'une application, un composant d'une architecture ou autre, selon l'énumération *DeploymentEndKind*.

A ce niveau, les tâches de l'application seront allouées sur les composants du modèle d'architecture. En effet, les tâches sont stéréotypées «*deployed*» ayant l'attribut {kind=application} et les composants de l'architecture seront également stéréotypés «*deployed*» avec {kind=executionPlateform}. Le lien qui associe le modèle d'application sur l'architecture est de type «*allocate*». Afin de guider l'utilisateur dans la création de ses modèles sans erreur, certaines contraintes peuvent surgir à ce niveau.

La première contrainte concerne le lien d'allocation :

if *source.stereotype =* `deployed` ***and*** *deployed.kind =application*
then *target.stereotype =* `deployed` ***and*** *deployed.kind=executionPlatform*

Dans le cas d'un modèle d'allocation physique, c'est-à-dire allouer les composants de l'architecture logique sur l'architecture physique (région statique et/ou reconfigurable), nous avons également besoin de spécifier quelques contraintes pour que le modèle soit correct.

condition *: self = component.stereotype =* `deployed` ***and*** *deployed.kind=executionPlatform*
if *model.object[*`allocate`*].select(target=self* ***and*** *source.stereotype(*`ReconfigurableRtUnit`*)* $\neq$ *null)*$\neq$ *null*
then *(model.object[*`allocate`*].select(source = self).target.stereotype(*`HwReconfigurableRegion`*) =null) = null* ***(1)***
and *(model.object[*`allocate`*].select(source = self).target.stereotype(*`HwReconfigurableRegion`*))* $\neq$ *null* ***(2)***

Cette contrainte résume que pour un composant donné C1 sur lequel on a alloué une tâche T1 stéréotypée «*ReconfigurableRtUnit*», il faut que C1 : (1) soit alloué sur au moins une région stéréotypée «*HwReconfigurableRegion*» (la contrainte spécifie qu'aucun composant n'ayant pas le stéréotype HwReconfigurableRegion) et (2) ne soit pas alloué sur une région non stéréotypée «*HwReconfigurableRegion*» (au moins une région stéréotypée HwReconfigurableRegion).

Nous traitons également le cas d'une tâche non reconfigurable c'est-à-dire elle n'est pas stéréotypée «*ReconfigurableRtUnit*», elle sera allouée sur un composant qui ne doit pas être placé sur une zone reconfigurable, c'est-à-dire placé sur une zone statique. La contrainte est illustrée comme suit :

condition *: self = component.stereotype =* `deployed` ***and*** *deployed.kind=executionPlatform*
if *model.object[*`allocate`*].select(target=self* ***and*** *source.stereotype(*`ReconfigurableRtUnit`*) = null)*$\neq$ *null*
then *(model.object[*`allocate`*].select(source = self).target.stereotype(*`HwStaticRegion`*) =null) = null*
and *(model.object[*`allocate`*].select(source = self).target.stereotype(*`HwStaticRegion`*))* $\neq$ *null*

5.5.3 Attribution des IPs

La définition de composant présentée par [86] permet la séparation entre le *comportement* d'un composant et son *interface*. En effet, le comportement du composant définit la fonctionnalité ou la réalisation exécutable du composant. Il s'agit d'une sorte d'association du composant avec une *implémentation* telle qu'un code compilable, une forme binaire, etc. ; en fonction du modèle du composant. Cette notion permet de lier le composant à des implémentations de tiers ou à une propriété intellectuelle (Intellectual Property IP). De plus, dans le domaine de développement logiciel pur, un composant peut être vu comme une implémentation logicielle à exécuter sur un dispositif logique ou physique. Alors que l'interface d'un composant représente les propriétés externes du composant qui sont visibles par les autres parties du système et utilisées pour

la conception et le développement du système. Ces interfaces fournissent des informations supplémentaires relatives à l'interaction du composant soit avec son environnement, soit avec d'autres composants ou bien avec des propriétés extra-fonctionnelles telles que le débit, le temps d'exécution, etc. Cette itération permet une identification plus précise des propriétés du système complet, lors de la phase de conception initiale.

Ainsi, une fois le modèle de déploiement des tâches de l'application sur les composants de l'architecture logique est établi, le concepteur doit être capable d'associer, de manière précise, chaque composant déployé à son IP correspondant. En effet, cette association est importante afin de générer puis d'exécuter un système complet depuis le haut niveau de modélisation. De plus, les informations obtenues devront non seulement générer le squelette du code, mais aussi assurer l'exécution correcte du système. Certes, le code doit être correct, sans erreurs et suffisamment efficace afin d'être directement compilé et exécuté pour une éventuelle simulation ou synthèse.

5.5.3.1 Présentation du profil

Le principe de base du niveau de déploiement est de fournir aux concepteurs les moyens nécessaires pour relier les blocs de construction de base, ce que nous appelons les composants déployés, aux différentes implémentations qui incarnent leur comportement. Plus précisément, l'information fournie devrait être assez suffisante pour que l'intégration du code dans ces IPs soit effectuée de manière automatique, au moment de la génération et l'exécution du code. Ainsi, il devrait être possible d'associer au moins un IP à chaque composant déployé.

Pour ce faire, nous définissons un nouveau stéréotype «*IP*» qui est une méta-classe associée au concept «*deployed*». Ce concept contient un ensemble d'attributs utilisés afin de décrire les informations de base d'un IP à un haut niveau d'abstraction. Ces attributs permettent également de relier les descriptions de haut niveau à leurs équivalents dans la description IP-XACT. Comme le montre la figure 5.26, nous avons décidé de garder le minimum d'attributs étant donné que le concepteur, à ce niveau d'abstraction, n'a pas besoin de connaître tous les paramètres et les détails d'implémentation de l'IP.

Ces attributs sont détaillés comme suit :

– ***id*** : de type *Identifier* qui est un `DataType` dans MARTE, tout comme les NFPs (*Non-Functional Properties*) qui sont appliquées dans MARTE à travers des `DataType`. En effet, le concept UML `DataType` (Data Type metaclass) est un type de classificateur, similaire au concept de `class` UML. La seule différence est que les instances du DataType sont identifiées uniquement par leur valeurs. Tel une classe, le DataType peut avoir plusieurs attributs. Dans notre cas, le tuple *Version, Library, Name* et *Vendor* permettent d'identifier de manière unique et précise l'IP depuis la bibliothèque des IPs. Ce tuple est également utilisé dans la norme IP-XACT pour nommer les descriptions des composants. Dans le cas de l'environnement EDK (Embedded Development Kit) de Xilinx, les valeurs du nom et la version sont obtenues à partir du fichier MPD (*Name* et *Hw_Ver*).

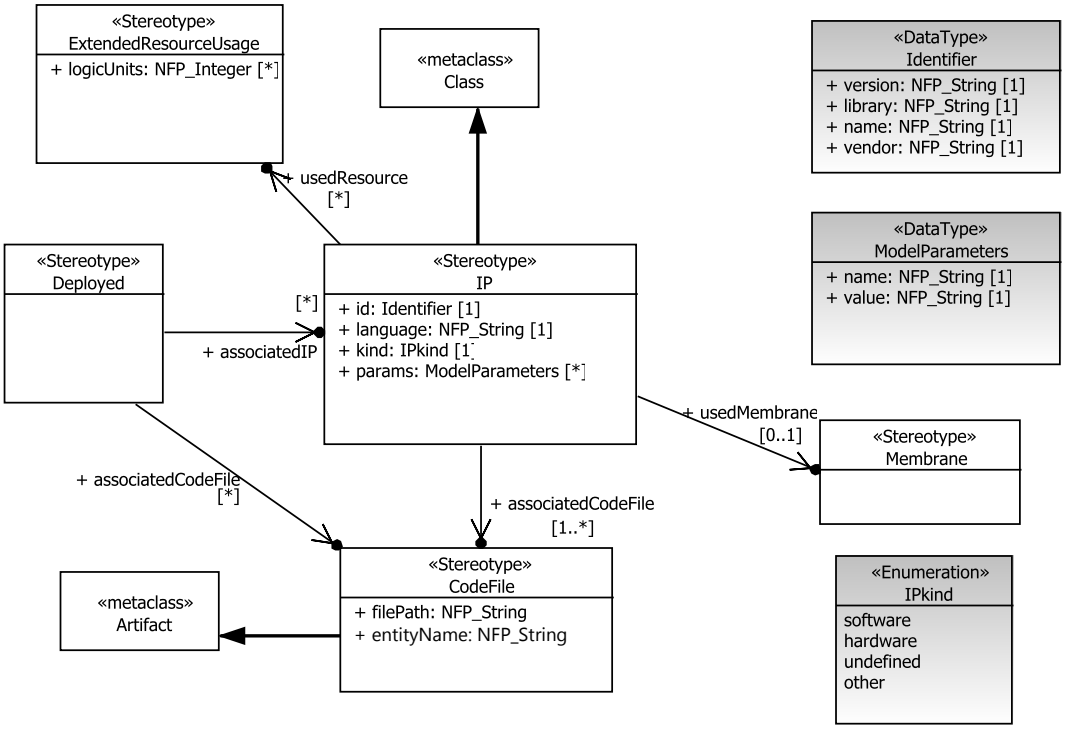


FIGURE 5.26: Le stéréotype «*IP*» du profil RecoMARTE

- ***Language*** : La fonctionnalité d'une tâche applicative peut être écrite en langage de description matérielle HDL (VHDL ou Verilog) si elle sera implémentée sur un accélérateur matérielle, ou encore elle peut être optimisée pour un processeur et dans ce cas elle sera écrite en langage de haut niveau tel que C/C++. Cette information est importante pour savoir s'il y a besoin d'adapter le langage de l'IP avec celui du système généré.
- ***Kind*** :cet attribut prend une valeur de l'énumération *IPkind* qui peut être Hardware ou Software ou autre. Dans [70], l'auteur définit un IP software comme étant un IP qui décrit les tâches d'une application même s'il est écrit en langage HDL. Contrairement à notre définition, nous considérons que : un IP de type matériel (Hardware) est un IP dont le langage d'implémentation (*language*) est une description matérielle qu'il soit un IP d'une tâche applicative ou d'un composant de l'architecture. De même pour un IP Software. Cette information est utile afin d'identifier les paramètres qui doivent être utilisés dans le flot, puisque le type d'implémentation détermine leur configuration.
- ***Params*** : est de type DataType *ModelParameters* qui est une paire de paramètres (nom,valeur) spécifiques à chaque composant. Étant donné que dans notre cas d'étude, nous visons des FPGAs de chez Xilinx. Parmi les fichiers de configuration que nous utiliserons, le fichier MPD (*Microprocessor Peripheral Description*), dans lequel nous récupérons les informations liées à la personnalisation et le paramétrage des IPs. Ceci sera détaillé dans le chapitre 6.

La figure 5.27 illustre un exemple de l'association d'un composant UART déployé et de son IP correspondant. Le composant de l'architecture logique est maintenant stéréotypé «*deployed*» ;

la relation d'association "associatedIP" spécifie le nom de l'IP qui lui est associé. Cette association est illsutrée par le commentaire rattaché au composant. D'un autre côté, l'IP correspondant *UartLite* est modélisé par une classe stéréotypée «IP» en spécifiant certains attributs nécessaires à l'identification de l'IP tels que l'*Identifier* et d'autres paramètres à spécifier pour le paramétrage de l'IP.

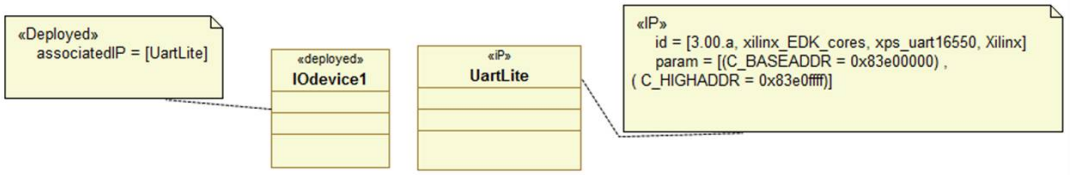


FIGURE 5.27: Modélisation de l'association d'un composant déployé et son IP correspondante

5.5.3.2 *Extension du paquetage* `ResourceUsage`

Étant donné qu'il est nécessaire de déterminer les ressources consommées par une implémentation, nous associons la classe IP au concept de `ResourceUsage` du paquetage GRM (*Generic Resource Model*). Cette information est utile lors de la décision du choix de l'implémentation de l'IP sélectionnée. Les ressources consommées sont spécifiées en terme de nombre d'unités logiques telles que des éléments logiques (*Logical Element*) ou des *Slices*, tout dépend de la plateforme FPGA choisie. Cependant, ce type de ressource n'existe pas dans ce paquetage. Nous rajoutons donc ce concept sous forme stéréotype «*ExtendedResourceUsage*» qui hérite du stéréotype «*ResourceUsage*» du paquetage GRM et est associé au stéréotype «*IP*» précédemment présenté par la figure 5.26. Le nouveau stéréotype dispose d'un attribut *LogicUnits* de type *NFP_Interger* afin de spécifier les ressources de l'FPGA consommées par cet IP. Le stéréotype correspondant est illustré par la figure 5.28.

5.5.4 Le concept Code File

L'objectif du niveau de déploiement est double. En premier lieu, il contribue à la phase finale de la génération de code du modèle haut niveau d'un SoC. En second lieu, il fournit les sémantiques permettant d'assurer la compilation et l'exécution correcte du code. Pour ce faire, il offre tous les fichiers sources nécessaires relatifs à un IP sélectionné. Ainsi, nous introduisons le concept du `CodeFile`. La figure 5.26 illustre le stéréotype «*CodeFile*» qui est une méta-classe `UML::Artifact`. Un codeFile est utilisé pour représenter un seul fichier en spécifiant deux attributs. Il est impératif de spécifier la valeur de l'attribut *FilePath* qui contient le chemin vers le répertoire physique où se trouve le fichier source sur la machine sur laquelle le code du système sera généré. Le deuxième attribut *EntityName* spécifie le nom de la boite noire qui contient les différentes implémentations. Par exemple, un IP matériel peut avoir plusieurs implémentations, il faut donc 1) définir plusieurs Codefiles correspondant au nombre d'implémentations 2)tous les codeFiles possèdent la même valeur de l'attribut *entityName* puisqu'il s'agit du même IP

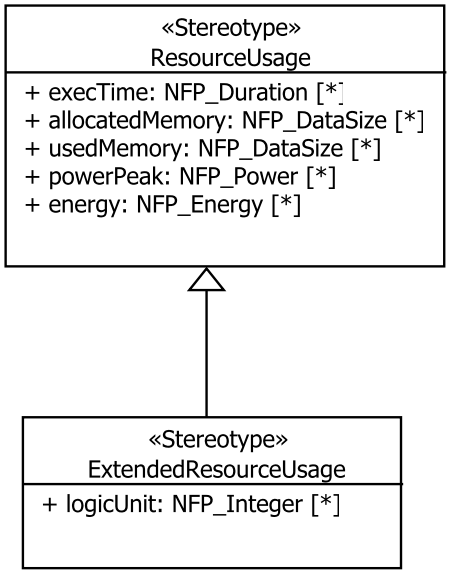


FIGURE 5.28: Extension du concept «*ResourceUsage*» du profil MARTE GRM

(même boite noire avec la même interface) 3) chaque Codefile est identifié par un chemin unique spécifié par l'attribut *filePath*.

Le composant déployé peut être directement associé à un CodeFile sans pour autant être associé à un IP. D'où l'association directe entre «*deployed*» et «*CodeFile*».

Le code file peut être écrit en langage matériel (HDL) ou en C. Il peut être également sous forme de fichier d'extension **.ucf* (User constraint file). En effet, un cas particulier peut se présenter où l'utilisateur souhaite rajouter un composant. C'est plus utile de lui attribuer directement son codeFile sous forme de fichier.ucf à partir d'une bibliothèque déjà existante. Ce cas de figure sera détaillé lors des transformations dans le chapitre 6.

5.5.5 Bilan du méta-modèle du Déploiement RecoMARTE

Tous les concepts précédemment détaillés sont présentés dans la figure 5.29. La relation entre le concept IP et la Membrane pour la sauvegarde du contexte est également illustrée par cette figure. En effet, un IP reconfigurable peut être associé à, au plus, une membrane, et ceci doit être prévu lors de la définition de l'architecture physique.

5.6 Le Contrôle de la reconfiguration dans RecoMARTE

L'objectif principal du projet FAMOUS est d'améliorer la modélisation MARTE des systèmes embarqués dynamiquement reconfigurables, en particulier les FPGAs. Ainsi, il faut, tout d'abord définir les modèles des systèmes reconfigurables. Ces modèles identifient l'exécution des tâches concurrentes de l'application, disposant de plusieurs implémentations. Ces implémentations sont allouées par la suite soit, en tant que tâches matérielles (accélérateur matériel) sur un FPGA, soit en tant que tâches logicielles sur un processeur. De plus, un système de métriques doit être

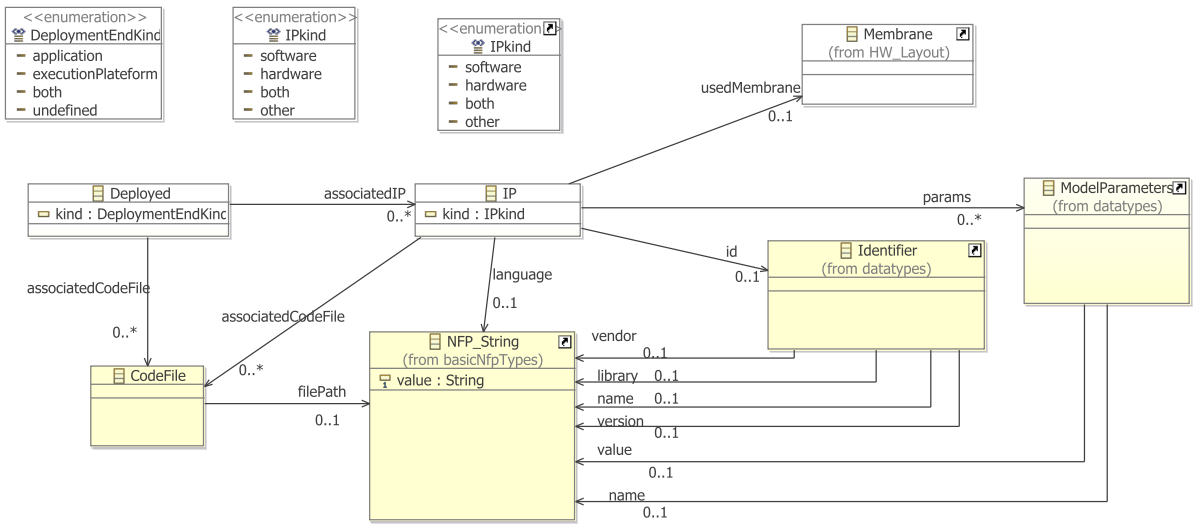


FIGURE 5.29: La vue générale du méta-modèle Deployment du profil RecoMARTE

établi à des fins d'analyse. Ceci permettra de spécifier des propriétés non fonctionnelles (issues par exemple d'outils de profiling) aux différentes configurations du système.

Dans un système, la notion de configuration représente une combinaison de modes actifs de ce système à un instant donné. Ainsi, intervient le contrôleur qui choisit quelle configuration charger, en leur attribuant des priorités. Il décide également de la suspension ou la reprise des tâches en cours d'exécution.

Il est donc nécessaire de définir le flux événements traités par les ModeBehaviors. Dans notre cas, ces éléments de modélisation sont les plus adéquats pour modéliser le comportement du système. Le contrôleur est un composant qui va gérer les ModeBehaviors des tâches qu'il va contrôler. Pour ce faire, nous définissons le stéréotype «*Controller*» qui hérite du stéréotype «*RtUnit*» du paquetage HLAM de MARTE. Le choix du RtUnit est dû au fait qu'il permet de modéliser des unités temps réel qui ont un ensemble de ressources à ordonnancer. La sémantique d'exécution suit donc celle définie par les propriétés du stéréotype. A chaque exécution, ce RtUnit est considéré comme une ressource d'exécution qui gère un ensemble de ressources (les ModeBehaviors des tâches reconfigurables) et qui traite différents messages (les événements) afin de produire une configuration en sortie ; soit la combinaison de modes actifs.

Le stéréotype «*Controller*», illustré par la figure 5.30, a un attribut nommé *ControlledElements* qui permet de spécifier les ModeBehaviors des tâches à contrôler. Cet attribut est un tableau de *ModeBehavior*. Le rôle du contrôleur est de garantir que la configuration globale du système respecte un ensemble de contraintes globales. Ainsi, le deuxième attribut du «*controller*» permet d'identifier un fichier de contraintes globales que le concepteur doit définir. Cet attribut *contractFile* est de type NFP_String. Les contraintes à respecter sont décrites sous la forme de contrainte «*nfpConstraint*».

Le méta-modèle du paquetage HLAM est illustré par la figure 5.31. Cette figure contient le concept RtUnit étendu ainsi que le concept du contrôle de la reconfiguration dynamique.

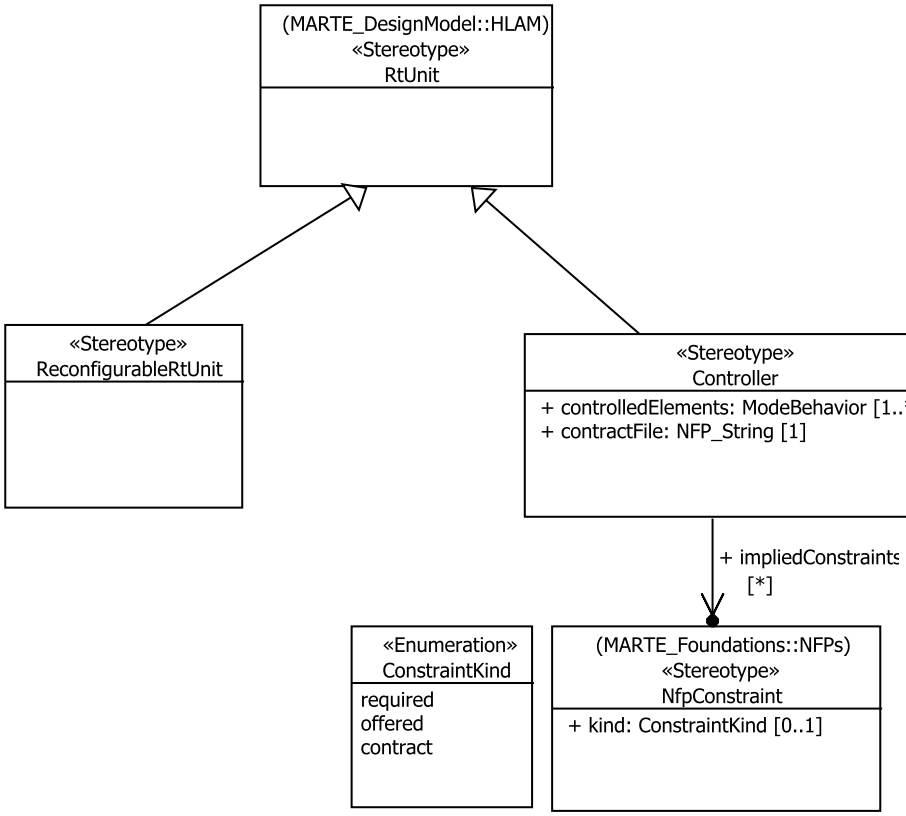


FIGURE 5.30: Définition du contrôleur dans RecoMARTE

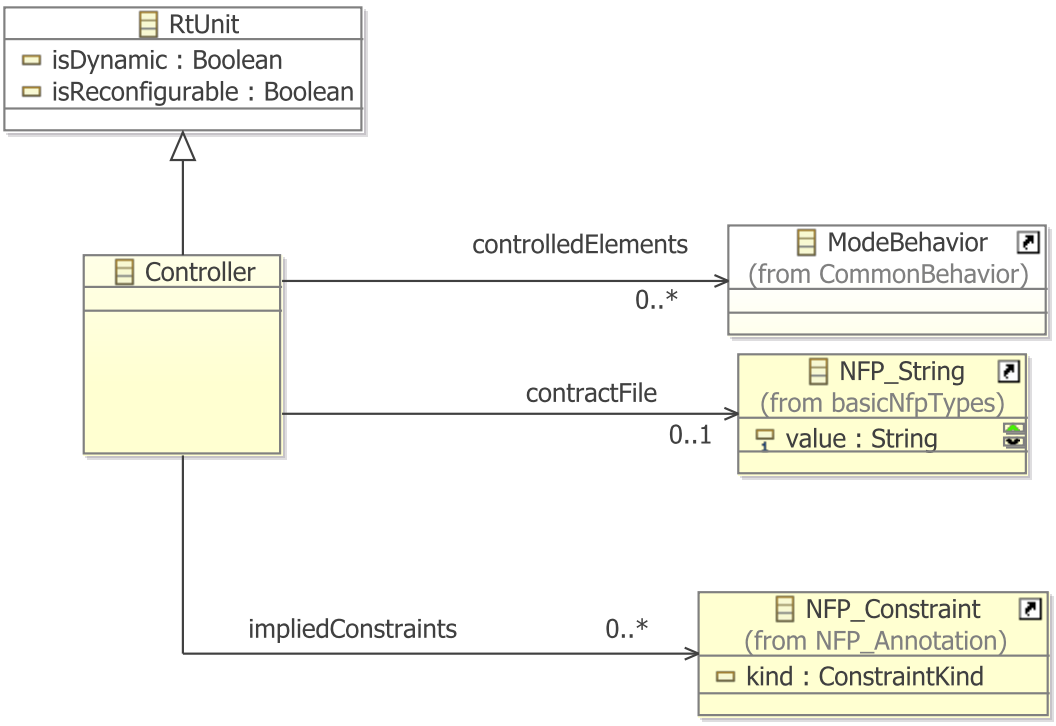


FIGURE 5.31: méta-modèle du paquetage HLAM étendu avec les concepts de la RD

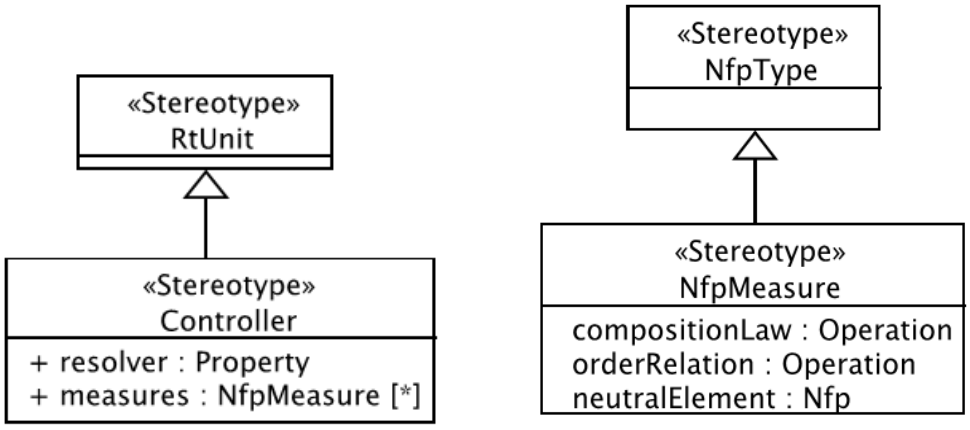
Afin de vérifier la validité des contraintes imposées, le concepteur doit utiliser une technique lui permettant de synchroniser les différents automates de modes contrôlés.

Notre partenaire, acteur du projet FAMOUS, a eu recours à l'utilisation de la Synthèse du contrôleur [35]. Cet acteur est parti d'une modélisation MARTE du contrôle de la RD et a proposé des transformations appropriées lui permettant de cibler une représentation synchrone, en langage BZR, de la partie contrôle. De plus, ses travaux visent deux aspects. D'une part, la sécurité du contrôle dans le sens où des espaces de configurations accessibles sont obtenus, formellement par synthèse ; et d'autre part l'optimisation à implémenter par le concepteur afin de produire un ordre de reconfiguration à partir d'un espace accessible.

Afin de garantir la validité des contraintes spécifiées, pour toute évolution du système, des variables contrôlables sont utilisées dans ses travaux. Ces variables sont identifiées par le contrôleur et sont ajoutées aux conditions des transitions des automates de modes en les combinant avec les événements de l'environnement (dites les variables non contrôlables).

En partant d'une modélisation en MARTE, le code synchrone est généré et le contrôleur est synthétisé en langage C.

L'utilisation de la technique de la synthèse du contrôleur a nécessité l'introduction de concepts supplémentaires à ceux présentés précédemment, pour la modélisation du contrôleur. Afin de calculer les variables contrôlables, le composant *Resolver* a été introduit en tant que propriété dans le stéréotype «*Controller*», comme le montre la figure 5.32. L'utilisation de ces variables donne la possibilité d'interdire une transition ou de la forcer, quelque soit la valeur des événements d'entrée. De plus, cette approche définit certaines contraintes sur les poids de configurations. Cette notion a été introduite sous forme de propriété non fonctionnelle *NfpMeasures* et traite la notion d'énergie consommée (*power*) et la qualité de service (*qos*).



(a) Définition du contrôleur (b) Ajout de la notion *nfpMeasure*

FIGURE 5.32: Extensions proposées par [35]

Un exemple illustrant ces travaux est présenté par la figure 5.33. Il intègre également deux exemple de contraintes décrites par le stéréotype précédemment présenté «*nfpConstraint*».

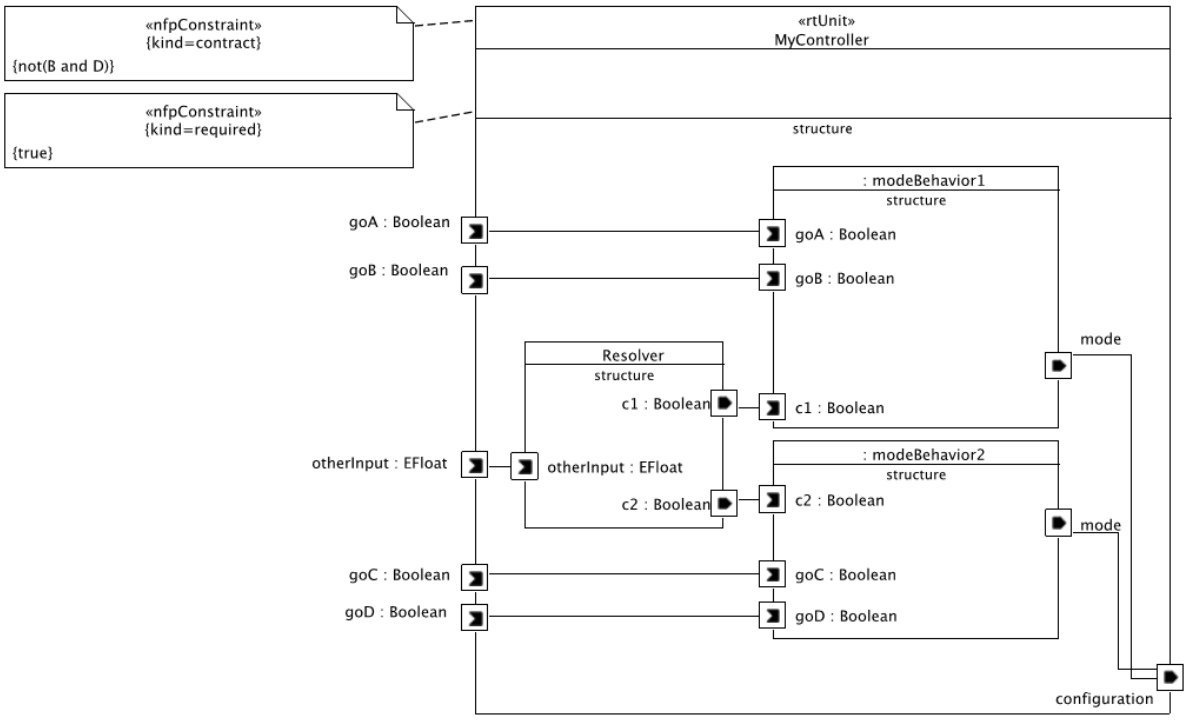


FIGURE 5.33: Présentation du contrôleur intégrant le resolver, dédié à la résolution des valeurs d'événements internes [35]

Ainsi, cette approche, basée sur la synthèse du contrôleur, a fait appel à une technique spécifique pour la synchronisation les automates de modes, d'où la nécessité d'introduire des concepts spécifiques afin d'assurer le respect des contraintes spécifiées par le contrôleur. Le but étant d'intégrer l'approche IDM à la synthèse du contrôleur afin de générer automatiquement le code synchrone et synthétiser le contrôleur en langage C.

5.7 Conclusion

Dans ce chapitre, nous avons proposé une modélisation de haut niveau basée sur le profil MARTE d'un système dynamiquement reconfigurable basé sur FPGA. Nous avons justifié l'absence de certains concepts qui permettent la modélisation de la reconfiguration dynamique. De ce fait, nous avons proposé certaines extensions qui définissent un nouveau profil qui étend MARTE et que nous avons baptisé RecoMARTE.

Notre méthodologie est basée sur le flot de conception haut niveau présenté dans le chapitre 4. Ainsi, nous avons présenté, pour chaque niveau de modélisation, les concepts à utiliser illustrés par des exemples. La méthodologie ainsi que la définition du profil résument la deuxième contribution de la partie "Modélisation haut-niveau", comme le montre la figure 5.34. Dans le but de valider notre méthodologie, nous présentons dans la partie qui suit, les transformations de modèles qui permettent d'automatiser notre flot de conception et validons par la suite nos travaux avec une étude de cas.

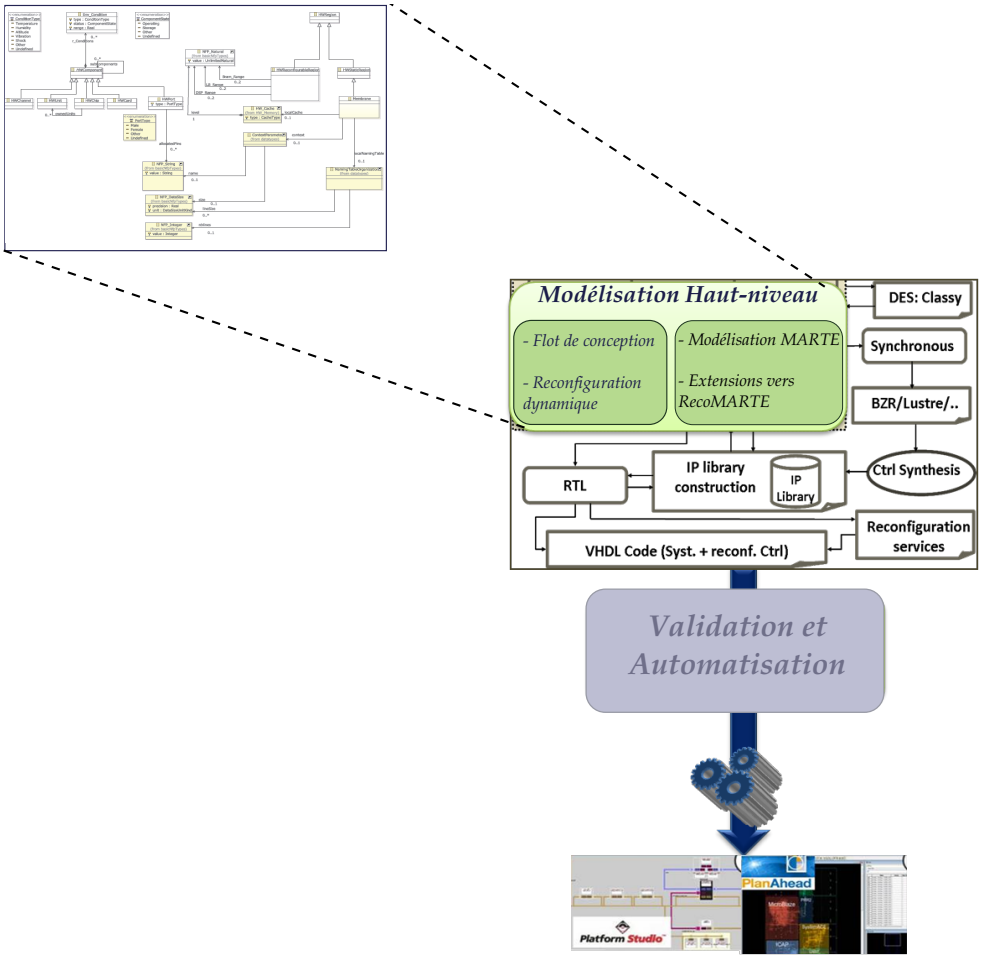


FIGURE 5.34: Contribution 2 : Présentation de la méthodologie et définition du profil RecoMARTE

Troisième partie

Validation et Automatisation du flot

CHAPITRE 6

DE LA MODÉLISATION À LA CHAÎNE DU CO-DESIGN

6.1 Introduction

Dans le but de diminuer la complexité de la conception des systèmes dynamiquement reconfigurables, basés sur FPGA, nous proposons une approche basée sur l'Ingénierie dirigée par les modèles (IDM), afin de générer automatiquement une description complète du système conçu à un haut niveau. Cette approche vise à cacher un grand nombre de détails d'implémentation qui seront générés automatiquement par les transformations de modèles et l'outil de génération de code. Ainsi, la conception de ces systèmes sera plus facile et nous éviterons certaines erreurs

dues à l'implémentation directe à un bas niveau (tel que le niveau RTL). Enfin, notre approche permettra également de réduire le temps de conception et de mise sur le marché des systèmes ciblés.

Nous présentons dans ce qui suit la chaîne de transformations proposée ainsi que le flot de génération de la plateforme. Ensuite, nous détaillons les règles de transformations qui nous permettent de passer d'une modélisation à haut niveau en MARTE vers la génération de fichiers décrivant le système complet dans l'environnement XPS de Xilinx. Enfin, nous présentons un bilan en évaluant notre approche.

6.2 Les transformations de modèles

6.2.1 Chaîne de transformation proposée

Rappelons que par définition [82], une transformation est basée sur un ensemble de règles qui permettent de définir le passage des concepts du méta-modèle source vers les concepts du méta-modèle cible. Elle peut être étendue en ajoutant ou modifiant des règles afin d'obtenir une nouvelle transformation ciblant un modèle différent. Pour la spécification des transformations de modèles, l'OMG a proposé le standard QVT (Query/View/Transformation) [62] qui définit un ensemble de langages déclaratifs et impératifs pour la transformation. Parmi ces langages, on cite QVTO (QVT Operational) [73] qui est un langage impératif implémenté par Eclipse.

A partir du méta-modèle MARTE, différents méta-modèles sont construits. La chaîne de transformations que nous proposons est composée d'un ensemble de transformations qui commencent par une transformation d'UML vers MARTE. Ensuite, les transformations qui suivent ont des méta-modèles cibles qui sont des versions étendues du méta-modèle MARTE. La figure 6.1 montre la chaîne de transformations que nous utilisons pour la génération des FPGAs dynamiquement reconfigurables. Cette chaîne permet d'ajouter progressivement des détails au modèle UML en entrée afin de se rapprocher peu à peu de la cible.

Une première transformation prendra comme entrée le modèle final de MARTE. Cette transformation permet de passer d'un modèle UML stéréotypé avec les concepts du profil MARTE étendu vers un modèle de sortie conforme au méta-modèle MARTE moyennant les extensions proposées. Ensuite, la deuxième transformation RecoMARTE/IP-XACT traduira tous les concepts de RecoMARTE vers un modèle conforme au méta-modèle IP-XACT. Cette transformation permet de promouvoir la réutilisation d'IP dans notre méthodologie en reliant les composants du haut niveau à leurs équivalents dans IP-XACT. De plus, le paramétrage des IPs permettant de contrôler les interfaces, les ports et les bus facilitera le passage vers les fichiers de Xilinx, grâce à la transformation IP-XACT vers les fichiers MHS et MPD. Cette dernière (numéro 4 dans la figure) fait partie des travaux de notre partenaire industriel Sodius et sera également présentée brièvement dans le chapitre.

Enfin, la transformation nommée RecoMARTE2UCF est de type modèle vers texte. Elle permet de traduire certains concepts de la modélisation physique de l'FPGA vers des instructions textuelles saisies dans le fichier UCF.

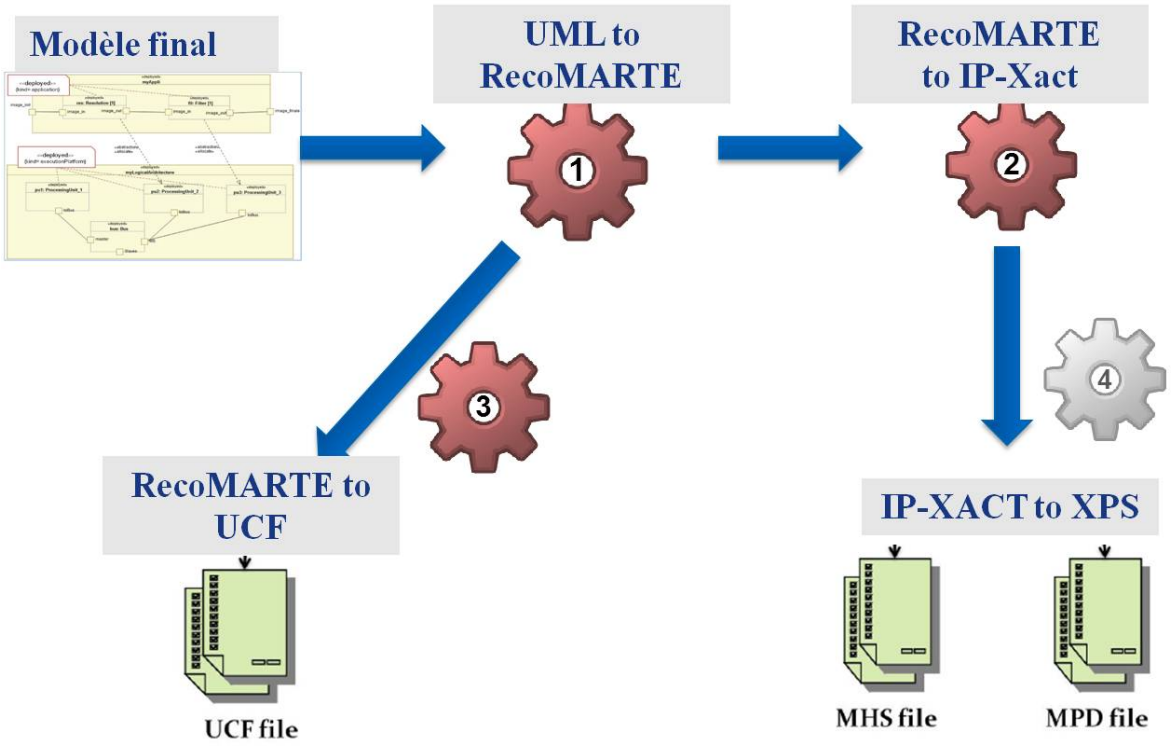


FIGURE 6.1: Chaîne de transformations pour le modèle final de notre flot de conception

6.2.2 Flot de génération de la plateforme : de MARTE vers Xilinx XPS

Nos travaux visent en particulier à faciliter la phase de conception comme étant une entrée au flot de conception des systèmes dynamiquement reconfigurables. Afin de compléter notre flot de conception présenté dans le chapitre 4, nous présentons la suite du flot utilisant les différents outils industriels. Ce flot complet est illustré par la figure 6.2 et est composé de quatre phase principales.

Étant donné que le modèle MARTE final ne contient que certaines informations appartenant à un haut niveau d'abstraction, il est nécessaire d'enrichir progressivement cette description suivant les différents niveaux dans le design flow. Ainsi, le modèle du haut niveau est traduit en un fichier XML utilisé pour enrichir la bibliothèque intermédiaire de description des composants (1). Cette bibliothèque contient des descriptions IP-XACT utilisées pour différentes fins ; la spécification est devenue un standard permettant de promouvoir la notion de réutilisation des IPs dans l'industrie du EDA (Electronic Design Automation), et nous exploitons ses fonctionnalités et ses capacités à générer différents types de sorties utilisées dans le flot de conception des SoCs dans l'environnement EDK de Xilinx. Le standard IP-XACT requière l'utilisation d'un environnement de conception (Design Environment) afin de paramétrer les blocs d'IPs ainsi que leur interconnexion dans le but d'obtenir une description <top level> du système. Pour ce faire, nous avons fait appel, dans nos travaux, à l'outil MDWorkbench de Sodius [4]. Cet outil nous permet d'importer la description de MARTE et IP-XACT de la conception du haut niveau (top level) et des IPs, vers un traitement ultérieur (2). En outre, cet outil utilise différents méta-modèles et chaînes de transformations dans le but de générer des fichiers utilisés dans

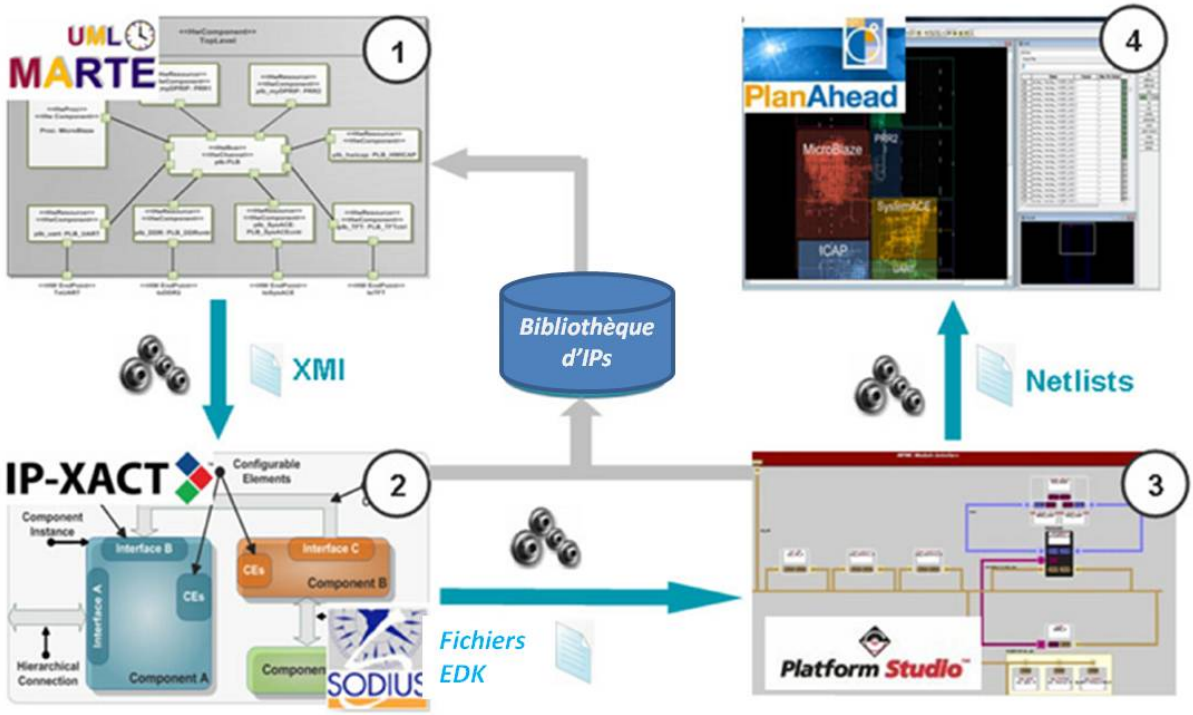


FIGURE 6.2: Flot de conception proposé complété par les outils [65]

le flot de conception EDK de Xilinx. Nous avons ciblé ce flot pour deux raisons : il facilite la conception des systèmes à base de processeurs soft ; et nous visons à modéliser les différents aspects de la chaîne de conception de Xilinx pour les systèmes dynamiquement reconfigurables basés sur FPGA.

La description obtenue de IP-XACT est utilisée afin de générer certains fichiers utilisés par l'environnement EDK de Xilinx (Embedded Development Kit)[104][105]. Parmi ces fichiers nous citons MHS (Microprocessor Hardware Specification) et MPD (Microprocessor Peripheral Description). Ces fichiers sont utilisés par l'outil Platgen [107] afin de générer la plateforme du système sur puce basé sur FPGA. De plus, cet outil génère la description HDL du haut niveau tandis que les fichiers HDL des modules reconfigurables sont rassemblés à partir d'une bibliothèque indépendante. Enfin, la conception haut niveau ainsi que chacun des modules reconfigurables sont synthétisés séparément (3). Ces Netlists sont utilisées par la suite comme des entrées pour l'outil PlanAhead de Xilinx [100],où l'implémentation physique du système PDR basé sur FPGA est effectuée (4).

Nous rajoutons que, certaines données nécessaires à l'implémentation physique de l'FPGA peuvent être déduites à partir du modèle MARTE, comme nous le montrons dans la transformation RecoMARTE 2UCF. Comme précédemment mentionné, le fichier UCF contient des contraintes ainsi que des informations sur le placement (les coordonnées) des régions reconfigurables (RR) et sert d'entrée pour l'outil PlanAhead.

Ainsi, les outils utilisés sont Papyrus [29] pour la modélisation, QVTo [73] pour la transformation de modèles et Acceleo [6] pour la génération de code.

Enfin, il faut noter que l'intégration de nos travaux de développement des transformations

spécifiées dans ce chapitre a été effectuée par notre partenaire industriel Sodius avec l'outil MDWorkbench.

6.3 La transformation de UML vers MARTE/RecoMARTE

A la base, cette transformation traduit le modèle en entrée en un modèle conforme au méta-modèle MARTE tel qu'il est défini par l'OMG. Pour pouvoir générer les systèmes dynamiquement reconfigurables à base d'FPGA, modélisés en utilisant le profil MARTE avec les extensions proposées dans le chapitre 5, nous avons modifié la transformation UML2MARTE.

Ainsi, la transformation UML 2RecoMARTE a comme entrée le modèle UML stéréotypé avec les concepts du profil MARTE et RecoMARTE. La sortie de cette transformation est un modèle conforme au méta-modèle MARTE et RecoMARTE. Le méta-modèle cible de cette transformation est une version étendue du méta-modèle MARTE en introduisant les concepts de la reconfiguration dynamique. Les concepts ajoutés au méta-modèle MARTE ont été présentés dans le chapitre 5.

Nous présentons dans le tableau 6.1 toutes les règles de la transformation UML2MARTE/RecoMARTE qui seront par la suite écrite en langage QVTo. Le format de ce tableau sera utilisé pour le reste des transformations :

- La colonne *Source* identifie le type de source d'entrée de la transformation ; le modèle conforme au méta-modèle d'entrée, dans ce cas il s'agit d'une entité UML : une classe, un port ou autre.
- La colonne *Condition* définit la condition à valider avant de commencer le traitement.
- La colonne *Traitement interne* détaille les opérations à effectuer sur certaines variables internes
- La colonne *Cible* identifie la classe du modèle conforme au méta-modèle de sortie.

Il est possible de nommer les différentes transformations afin de pouvoir les réutiliser (faire un appel) plus facilement (exemple : UMLClass2Membrane). Nous avons rajouté des commentaires pour certaines opérations afin de mieux expliquer le traitement à appliquer.

TABLE 6.1: Spécification des règles de transformation de UML2RecoMARTE

Source	Condition	Traitement interne	Cible
UML Class	stéréotypée «*ReconfigurableRtUnit*»	st ← stéréotype *RtUnit* iR ← st.isReconfigurable	**HLAM : : RtUnit**
UML Class	stéréotypée «*HwReconfigurableRegion*»	st ← stéréotype *HwReconfigurableRegion* LB_r ← st.LB_Range Bram_r ← st.Bram_Range DSP_r ← st.DSP_Range	**HRM : : HwReconfigurableRegion** LB_Range ← LB_r Bram_Range ← Bram_r DSP_Range ← DSP_r
UML Class	stéréotypée «*Membrane*» nom Transfo :*UMLClass2Membrane*	st ← stéréotype *Membrane* LC ← st.localCache LNT ← st.localNamingTable CTX ← st.context	**HRM : : Membrane** localCache ← LC localNamingTable ← LNT context ← CTX
UML Port	stéréotypé «*ExtendedFlowPort*»	st ← stéréotype *FlowPort* k ← st.kind	**GCM : :FlowPort** kind ← k
UML Port	stéréotypé «*HwPort*»	st ← stéréotype *HwPort* Ap ← st.AllocatedPins	**HRM : : HwPort** AllocatedPins ← Ap
UML Class	stéréotypée «*Controller*»	st ← stéréotype *controller* ce ← st.ControlledElements cf ← st.ContractFile	**HLAM : :Controller** ControlledElement ← ce ContractFile ← cf
UML Class	stéréotypée «*IP*» nom Transfo :*UMLClass2IP*	st← stéréotype IP i ← st.id l ←st.Language k ← st.Kind p ← st.Params	**Deployment : :IP** id← i Language ← l Kind ← k Params ← p

% *relation IP/Membrane* %		d ← ModelSource[Dependency]→select(source=self) →target→select(stéréotype =Membrane) if d ≠ null then usedMembrane ← d.target.UMLClass2Membrane()	
UML Class % *relation Deployed/IP* %	stéréotypée «*Deployed*»	st ← Stéréotype *Deployed* k ← st.Kind d ← ModelSource[Dependency]→ select (source= self) if d ≠ null then var vip ← d.target	**Deployment : :Deployed** kind ← k ip ← vip.UMLClass2IP()

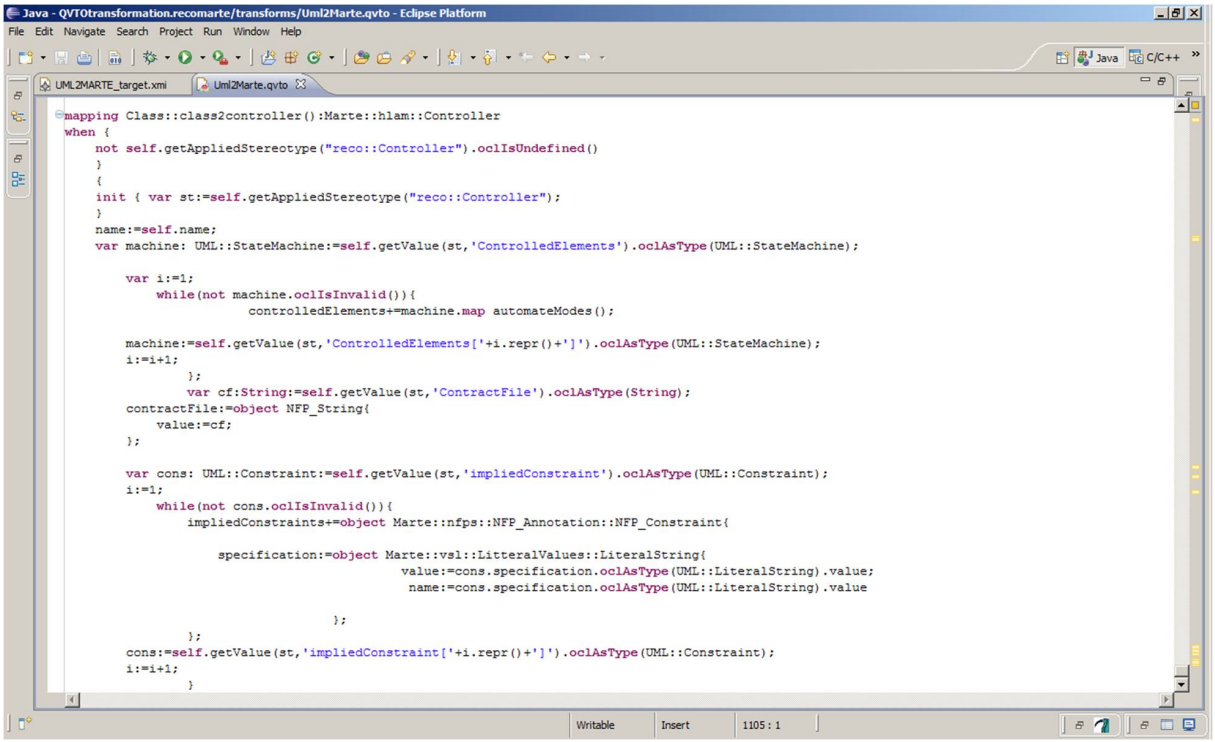


FIGURE 6.3: Extrait de la transformation du concept du *Controller*

Nous présentons dans ce qui suit quelques extraits du code QVTo. La figure 6.3 est un extrait du code qui permet de transformer la classe *Controller*.

La figure 6.4 illustre l'extrait de la transformation de la classe *HwReconfigurabeRegion* du paquetage HRM.

Enfin, un autre extrait de la transformation de la classe IP du méta-modèle IP est illustré dans la figure 6.5.

6.4 La transformation de MARTE vers IP-XACT

6.4.1 Pourquoi IP-XACT ?

La réutilisation et l'intégration des propriétés intellectuelles (IP) hétérogènes à partir plusieurs vendeurs est un enjeu majeur pour la conception des SoCs. Les outils existants tentent de valider les conceptions assemblées par une co-simulation globale au niveau d'implémentation. Cependant, cette solution n'a pas abouti à cause de la complexité croissante ainsi que la taille des SoCs actuels. Ainsi, un besoin clairement défini est de fournir une description multi-niveau d'un système sur puce avec de la vérification, de l'analyse et de l'optimisation afin de bien mener une modélisation sur différents niveaux d'abstraction. De plus, l'interopérabilité des composants IPs est inéluctable durant les différentes étapes ce qui nécessite l'utilisation de la traçabilité entre les différentes couches d'abstraction. Bien que ceci soit en partie favorisée par les nouveaux standards, il est néanmoins insuffisamment pris en charge par les méthodes actuelles. Parmi les standards, on note SPIRIT Consortium IP-XACT [10], Silicon Integration Initiative OpenAccess API [27] et le standard le plus récent UML MARTE défini dans le chapitre 2.

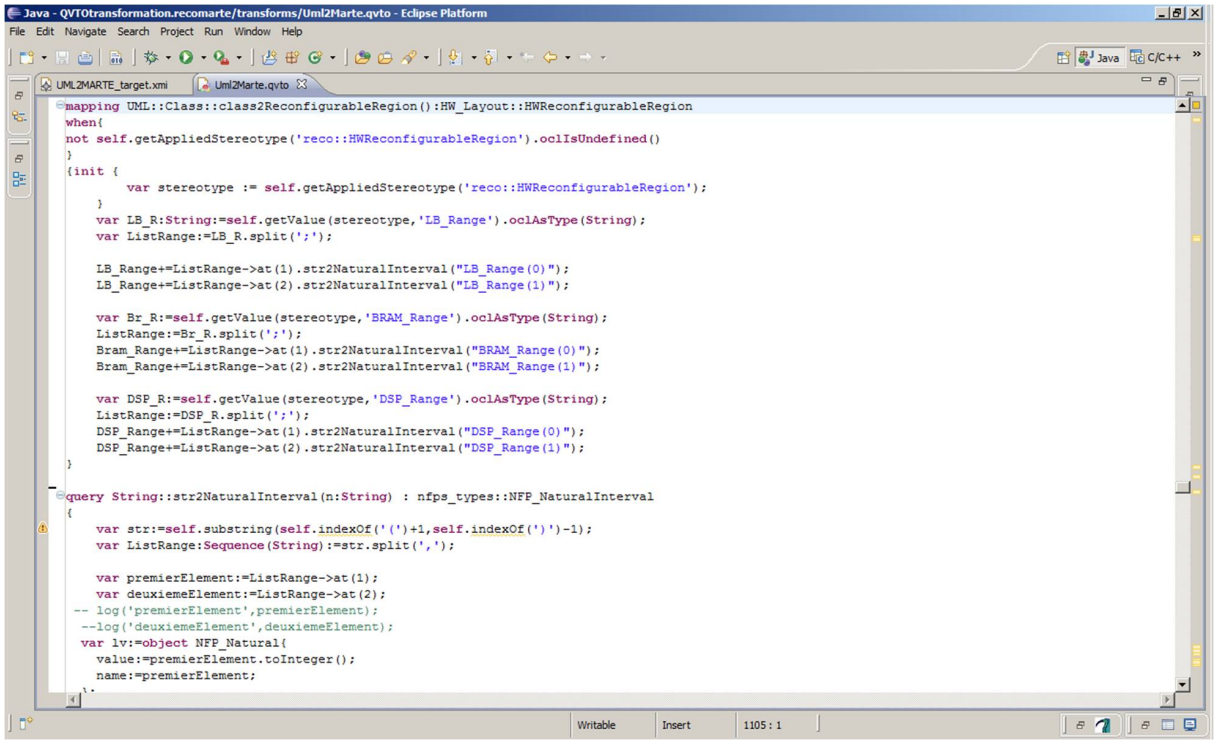


FIGURE 6.4: Extrait de la transformation du concept *HwReconfigurabeRegion*

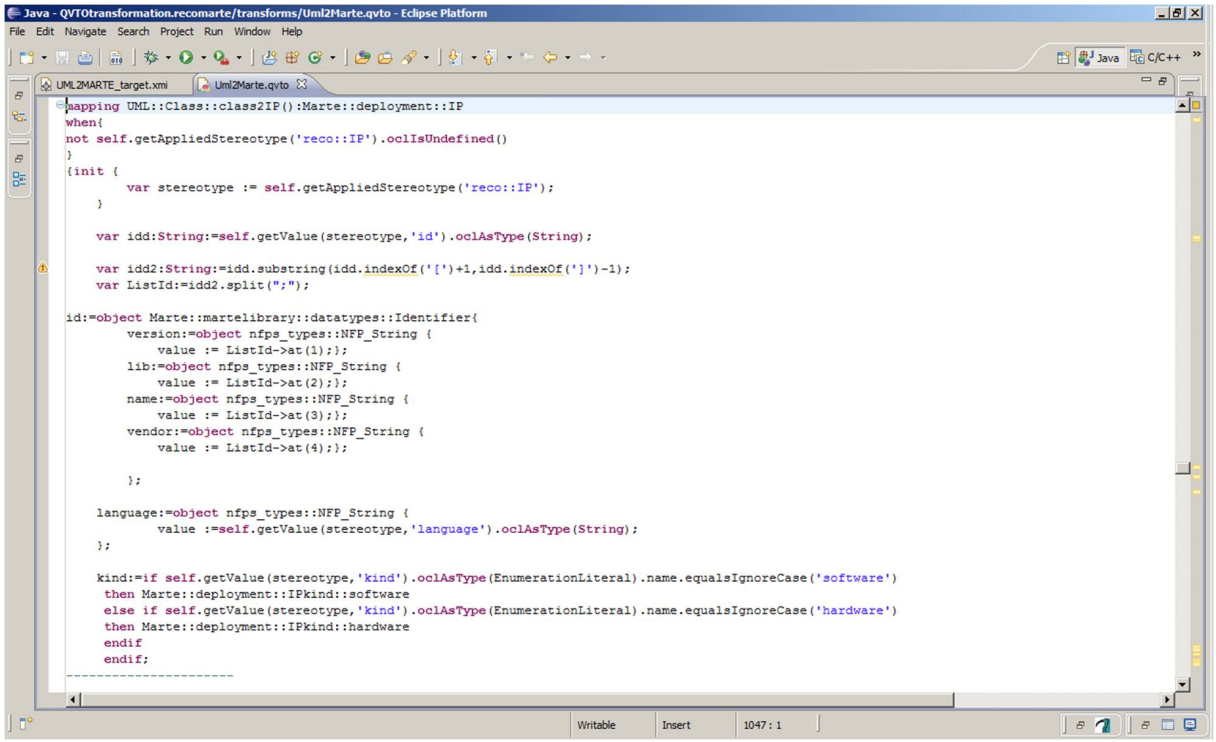


FIGURE 6.5: Extrait de la transformation de la classe IP du méta-modèle IP

Le choix d'utiliser la description IP-XACT, notamment dans nos travaux, est justifié par le fait que IP-XACT fournit une représentation intermédiaire à partir de nos modèles haut niveau MARTE. Il s'agit d'un méta-modèle sur lequel plusieurs outils peuvent être développés, d'où l'intérêt d'unifier MARTE et IP-XACT. Ceci permet à la fin de visualiser des IPs dans les modèles MARTE afin de composer un diagramme structurel qui se transformera par la suite en un design IP-XACT.

IP-XACT est considéré comme un livre électronique de données [92] ; une description de composants et de design créés dans un format standard d'échange de données (XML). L'avantage de ce langage est qu'il est à la fois lisible par l'être humain ainsi que par la machine. IP-XACT décrit la conception des systèmes électroniques et les interconnexions des interfaces IPs dans une spécification standard afin de fournir aux fournisseurs d'IPs, aux intégrateurs de conception et vendeurs de EDA (Electronic Design Automation), une représentation commune. Ainsi, IP-XACT fournit un mécanisme par lequel la réutilisation de la conception peut être une réalité pratique.

Le format de documentation IP-XACT standardisé par le consortium <SPIRIT> offre une méthode universelle de description de blocs d'IP permettant leur intégration automatique dans les designs avec l'aide d'une gamme d'outils compatibles IP-XACT.

6.4.2 Les concepts de IP-XACT utilisés dans notre méthodologie

Le standard IP-XACT définit des données à base du langage XML ainsi que la description du système. En effet, il définit quatre descriptions d'objet principales : définition des bus (en anglais Bus Definition), définition abstraite (abstract definition), composant (component) et la description de conception (design descriptions). Ces quatre éléments sont suffisants pour une description structurelle d'un système ainsi que les IPs qui le composent.

6.4.2.1 La description des composants

La description des composants englobe les informations qui concernent les blocs d'IPs, comme le montre la figure 6.6. Afin de faciliter la compréhension des concepts de IP-XACT, nous avons choisi de les représenter sous formes de blocs plutôt que d'utiliser les schémas fournis par le standard. Dans cette figure, nous avons intégré les concepts les plus répandus pour une représentation structurelle, une implémentation logique et le paramétrage.

La description des composants utilise les interfaces basées sur les bus <busInterface> afin d'interconnecter certains parties à d'autres éléments de la description du design. Les <bus Interfaces> font appel à deux autre objets IP-XACT : les bus et les définitions abstraites <abstract definition>. Ces derniers sont utilisés pour décrire le protocole du bus ainsi que la manière d'implémenter les interfaces.

Dans nos travaux [65], nous avons étendu les descriptions <bus Interface> avec des données sur les vendeurs afin de pouvoir distinguer entre les différents types d'interfaces (non seulement les interfaces basées sur les bus). L'onglet <Parameters> est utilisé non seulement pour configurer

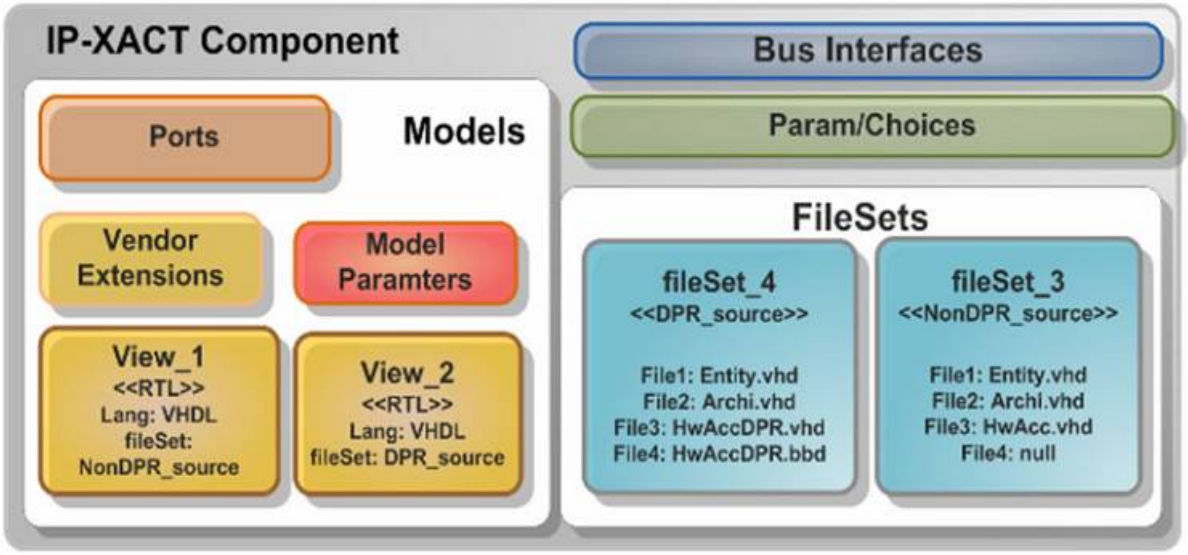


FIGURE 6.6: Les concepts de IP-XACT pour une description de composant [65]

certains paramètres des IPs tels que le type d'implémentation mais aussi pour spécifier le flot de méta-données correspondant. Idem pour l'onglet <choices> qui définit également les paramètres sous forme de listes énumérées des valeurs prédéfinies.

Nous avons introduit <vendor extensions> au sein des paramètres choisis, des *bus interfaces* ainsi que des ports afin de prendre en compte l'intégration de leurs contrôleurs au niveau de la phase de génération. Ceci est important pour le paramétrage et la personnalisation des IPs alors qu'il n'est pas mentionné dans la spécification actuelle IP-XACT.

L'élément de Modèle <Model> permet de décrire les différentes vues, les ports ainsi que les paramètres liés au modèle du composant. Un IP peut contenir différentes vues telles que RTL, TLM, etc.

Les vues sont utilisées conjointement avec l'ensemble de fichiers dit <*FileSets*> et les générateurs d'informations afin de permettre l'automatisation des tâches liées aux composants telles que la synthèse d'FPGA, code source pour la compilation des pilotes. Les FileSets et les éléments des vues pour un composant IP-XACT sont étroitement liés, puisqu'une implémentation donnée, fait référence à certains ensemble de fichiers spécifiques.

Dans notre méthodologie, nous exploitons les moyens que nous offrent les éléments des vues pour la description des composants avec des objectifs différents mais ayant la même interface. De plus, le flot Xilinx pour la conception de la PDR requière comme entrée, les Netlists de la conception haut niveau (top-level), ainsi que les modules reconfigurables individuels. Dans ce cas, la fonctionnalité des modules reconfigurables doit être synthétisée indépendamment, tout en conservant la même interface dans l'implémentation du haut niveau. Ainsi, nous implémentons l'IP de la PDR de manière à ce que la *vue* RTL puisse être sélectionnée comme étant un paramètre dans MARTE, affectant ainsi les paramètres utilisés pour la configuration (d'où l'importance de contrôler l'intégration des paramètres, des interfaces de bus et les ports en fonction d'autres paramètres dans la description du composant). De même, l'élément *vue* (view) pointe vers un

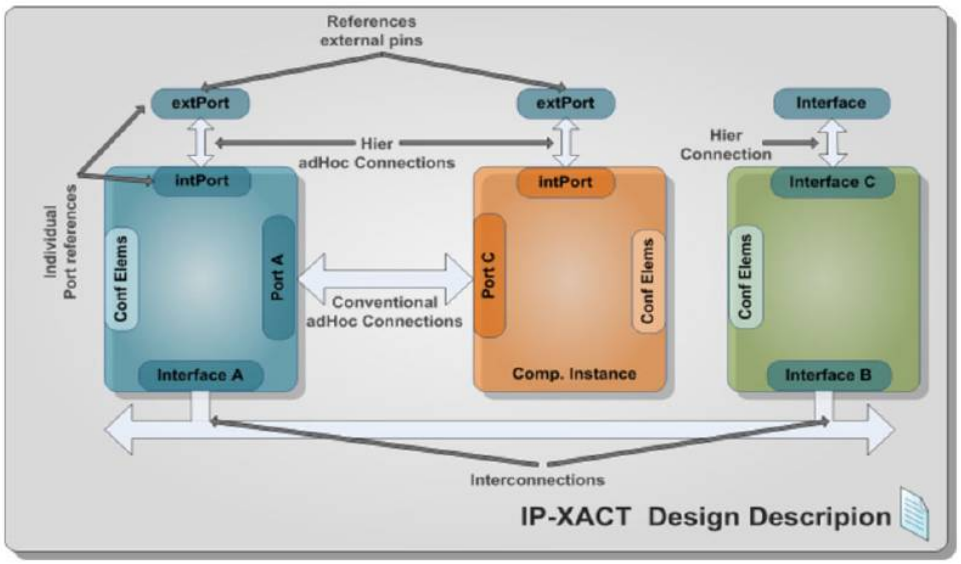


FIGURE 6.7: Les concepts de IP-XACT pour une description de système [65]

groupe de fichiers (FileSets) qui spécifie quel ensemble de fichiers devant être synthétisé pour l'implémentation de l'IP.

6.4.2.2 *La description de la conception*

Un objet de conception IP-XACT décrit une implémentation effective de haut niveau en tant qu'ensemble d'instances de composant qui, peut être configuré à travers de éléments configurables. Les sous-élements dans la conception sont connectés entre les interfaces de bus (qui sont conformes à la définition de bus prédéfini). Il existe trois types de connexions nommées <interconnection> dans IP-XACT : interconnexions, ad-hoc et les connexions hiérarchiques.

Alors qu'une conception IP-XACT, comme le montre la figure 6.7, avec des composants et des interconnexions référencés, décrit la plupart des informations de conception, certaines informations sont toujours absentes, telles que les noms de port exacts utilisés par une interface de bus. Pour résoudre ce problème, une description de composant (appelée *composant hiérarchique*) est utilisée. Cette *description de composant* contient une vue avec une référence à la description de la conception. Ensembles, le composant et description de la conception référencée forment une description hiérarchique à un seul niveau complet.

La description de la conception IP-XACT contient la majorité des informations nécessaires à la génération d'un système décrit dans un langage matériel tels que VHDL, Verilog ou aussi SystemC.

Les descriptions sont adaptées en ajoutant des extensions concernant le fournisseur ou dépendantes des flots des éléments configurables (*ConfigurableElements*). IP-XACT définit les concepts de générateurs et les chaînes de générateur pour accéder aux méta-données contenues dans ces descriptions. Ils sont utilisés afin de configurer les IPs dans la bibliothèque de composants, de générer des pilotes ou de personnaliser les composants. Cette tâche est effectuée par Sodius MDWorkbench, dans lequel nous importons les descriptions IP-XACT du plus haut

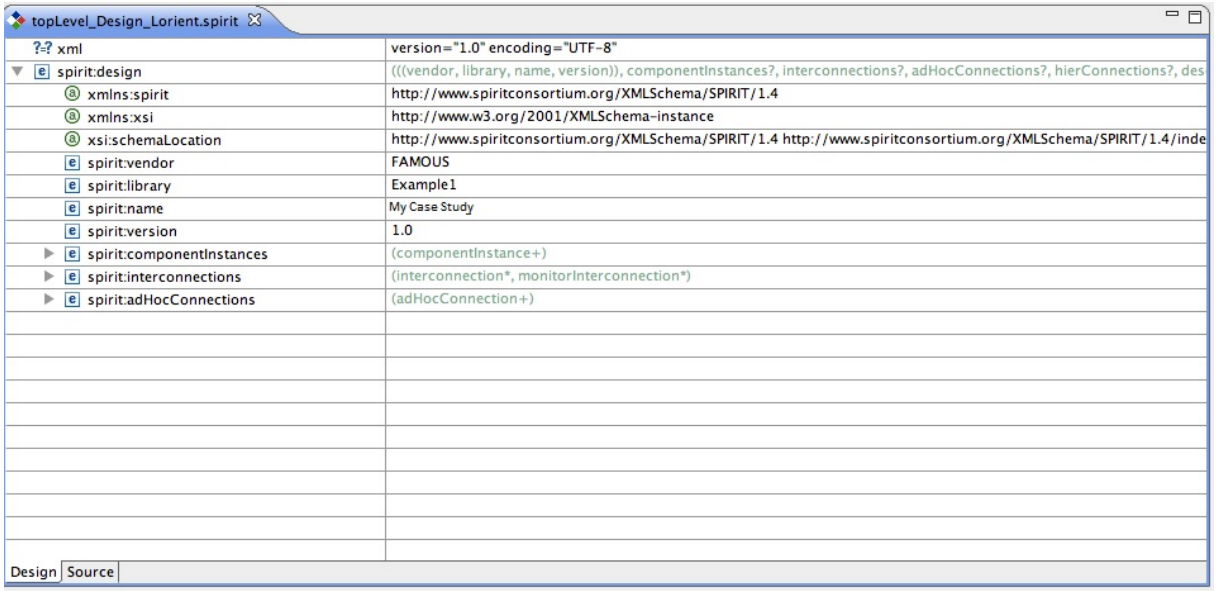


FIGURE 6.8: Schéma XML de description IP-XACT

niveau pour générer le fichier MHS de Xilinx. Ceci sera détaillé dans la section 6.6.

6.4.3 Définition des règles de transformations

Afin de composer une plateforme SoC à base de FPGA en utilisant le profil MARTE étendu (ensuite des niveaux inférieurs suivants dans le flux génération), les instances constitutives des IPs doivent être représentées dans le modèle. Cependant, nous avons décidé de séparer les informations (méta-données) de ces blocs d'IPs dans différents schémas de MARTE (views) afin d'éviter l'encombrement et d'améliorer la productivité du concepteur.

Tous les diagrammes du niveau de déploiement dans RecoMARTE doivent être transformés en une description standard intermédiaire, encodée par une description du standard IP-XACT. Le schéma XML de IP-XACT illustré par la figure 6.8, est généralement divisé en deux grandes parties : les instances des composants déployés (<*componentInstances*>) et les interconnexions (<*interconnections*>). Selon les besoins du flot associé à la modélisation, les interconnexions peuvent encore se diviser en <*adHocConnections*> et <*hierConnections*>.

La figure 6.9 présente un exemple qui montre le passage d'un modèle MARTE vers une description IP-XACT. Le fichier du modèle est composé de quatre sections principales. Chaque section est créée de manière séquentielle : la première section est <*ComponentInstances*> qui est principalement déduite du diagramme de paramétrage des IPs comme illustré par la figure. Chaque composant déployé (stéréotypé «*deployed*»), un IP lui est associé (stéréotypé «*IP*»).

Pour chaque classe d'IP dans le modèle MARTE, un élément <*instanceName*> est créé dans la section <*spirit :componentInstances*> du fichier de conception IP-XACT, avec l'élément <*configurableElements*> correspondant. Les informations permettant d'identifier chaque IP sont présentés par l'attribut <id> dans RecoMARTE, ce qui correspond au <*spirit :componentRef*> de IP-XACT. Les éléments de personnalisation et de paramétrage des IPs sont présentés sous la section

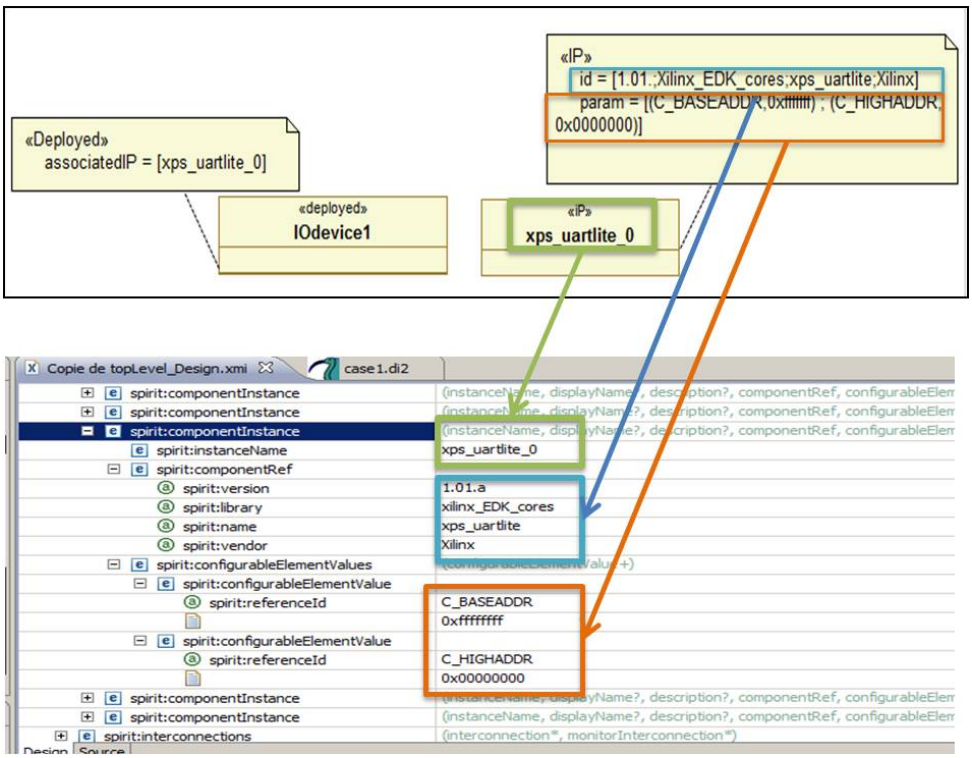


FIGURE 6.9: Concepts de paramétrage d'IP dans IP-XACT et MARTE

<*spirit :configurableElementValues*> de IP-XACT.

D'un autre côté, les informations de connectivité du système sont obtenues à partir de diagrammes MARTE modélisant la plateforme, illustré par l'exemple de la figure 6.10. Ces derniers contiennent les instances des composants déployés avec leurs ports associés. Comme mentionné précédemment, les définitions des ports ont été étendues dans RecoMARTE avec le stéréotype «*ExtendedFlowPort*» qui permet de les classifier en trois groupes : les interfaces connectées au bus (BusInterface), les connexions point à point (singlePorts) et les ports connectés aux ports d'entrée/sortie de l'FPGA (IOinterfaces). Ceci qui permet le mapping des éléments IP-XACT correspondants (busInterface dans la section <interconnections> et les singlePorts internes et externes dans le cadre du sous-élément <adHocConnections> sur la description de la conception).

Ainsi, les connecteurs dans MARTE (connectors) sont mappés sur les éléments d'interconnexion de IP-XACT ou bien sur les interconnexions <adHoc> selon le type des ports les reliant. Enfin, les informations concernant les pins externes sont obtenues à partir des labels des <ports> décrits dans la description des composants IP-XACT.

La figure 6.11 illustre un extrait de la transformation MARTE2IP-Xact dont les règles sont spécifiées dans le tableau 6.2.

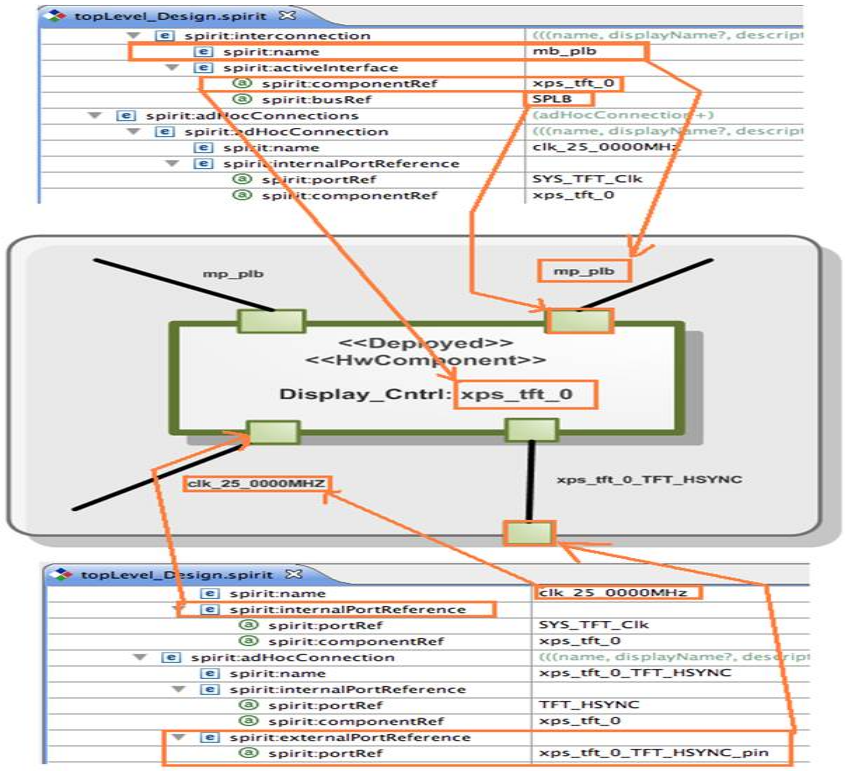


FIGURE 6.10: Concepts d'interconnexion dans IP-XACT et MARTE

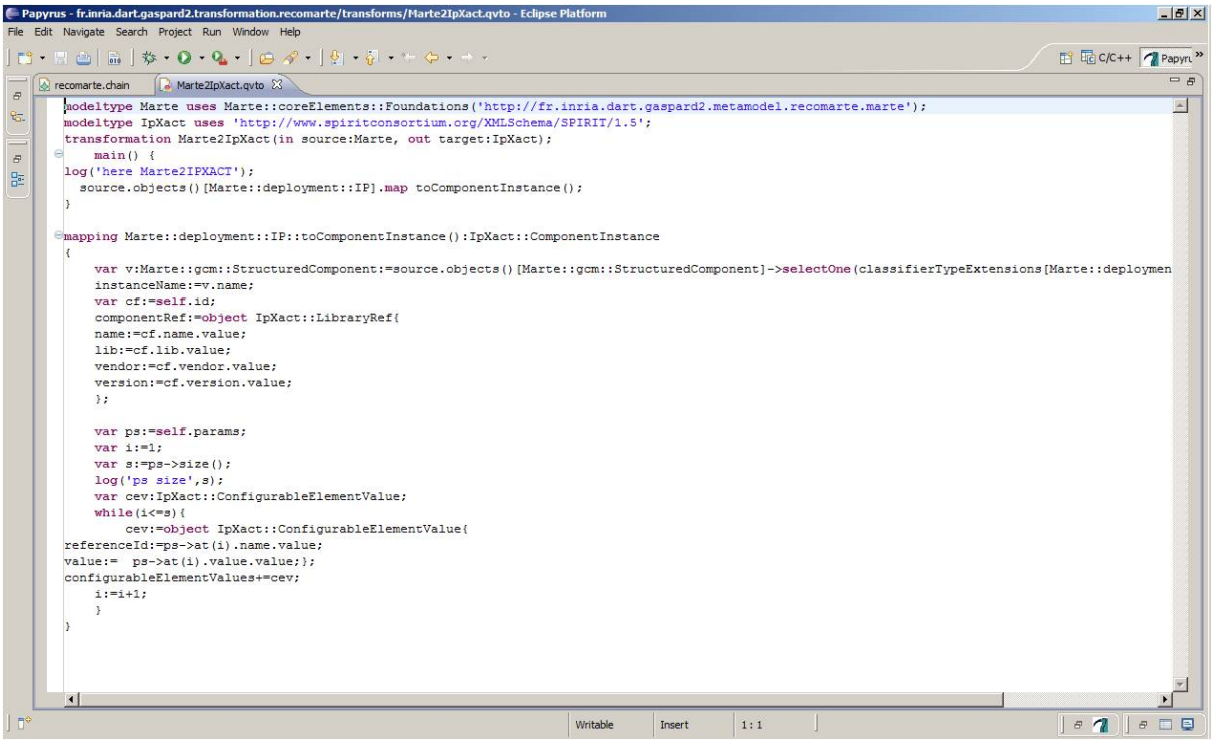


FIGURE 6.11: Extrait de la transformation MARTE/RecoMARTE vers IP-XACT

TABLE 6.2: Spécification des règles de transformation de MARTE vers IP-XACT

Source	Traitement	Cible
MARTE :IP		**Spirit : : Component Instance**
	cf ← self.id	instanceName = self.name componentRef.vendor = cf.vendor componentRef.library = cf.library componentRef.name = cf.name componentRef.version = cf.version
	ps ← self.params	
% définition d'une liste de paramètres %	cevs =configurableElementValue { }	
	for p in ps cev.referenceID ← p.name cev.value ← p.value cevs ← cevs + cev	ConfigurableElementValues ← cevs
% si on utilise une membrane	if usedMmbrane ≠ null	
alors il faut ajouter taille de ligne	cev.referenceId ← «lineSize»	
de même pour nombre de lignes	cev.value ← usedMembrane.localNamingTable.lineSize	
et le contexte %	cevs ← cevs.cev	
Assembly Connector		**Spirit : : Interconnection**
% définition d'une liste d'interconnection %	interconnect : interconononection{ }	
	var ports ← self.ends.endPort	
% tester si le composant est relié à un bus %	sc1 =ModelSource[StructuredComponent].ownedPorts → includes (ports[0])	
% sc = structured Component %	sc2 = ModelSource[StructuredComponent].ownedPorts → includes (ports[1])	

	if sc1.kind = HwBus sc1.ip	
%ai= activeInterface%	ai.busReference =port[1].name ai.componentReference = sc2.name	
%même traitement si sc2.kind= HwBus %		
	var i.name ← sc1.ip.name i.activeInterface ← ai	
%incrémenter la liste %	interconnect + = i	
%traitement du cas d'un single port ou IO%	if ports[0].kind =singlePort var i1 i1.name ← self.name i1.busReference ← ports[0].name i1.ComponentReference ← sc1	**Spirit : : Interconnection**
% incrémenter la liste %	interconnect + = i1	
% puisque 1 connecteur = 2 interfaces %		
% i1 étant la 1ère interface		
% traitement de la 2ème interface i2 %	if ports[1].kind = single port var i2 i2.name ← self.name i2.busReference ← ports[1].name i2.ComponentReference ← sc2	
%incrémenter la liste %	interconnect + = i2	

6.5 La transformation de MARTE vers UCF

La transformation MARTE vers le fichier UCF est une transformation de modèle vers texte. Les principaux concepts qui sont traités : 1)le CodeFile qui doit être de type *.ucf 2) la définition des coordonnées des différentes zones reconfigurables définies par l'utilisateur et 3) attribution des pins d'un port physique.

6.5.1 Le fichier UCF

Le fichier UCF (User constraints file) est un fichier ASCII [103] permettant de 1) spécifier les contraintes (principalement des contraintes de temps) sur la logique du système, 2) décrire les régions reconfigurables et 3) spécifier le placement des ports logiques sur les ports physiques. La figure 6.12 illustre un extrait du fichier UCF montrant les lignes de commandes pour le placement des ports et la spécification des coordonnées des régions reconfigurables.

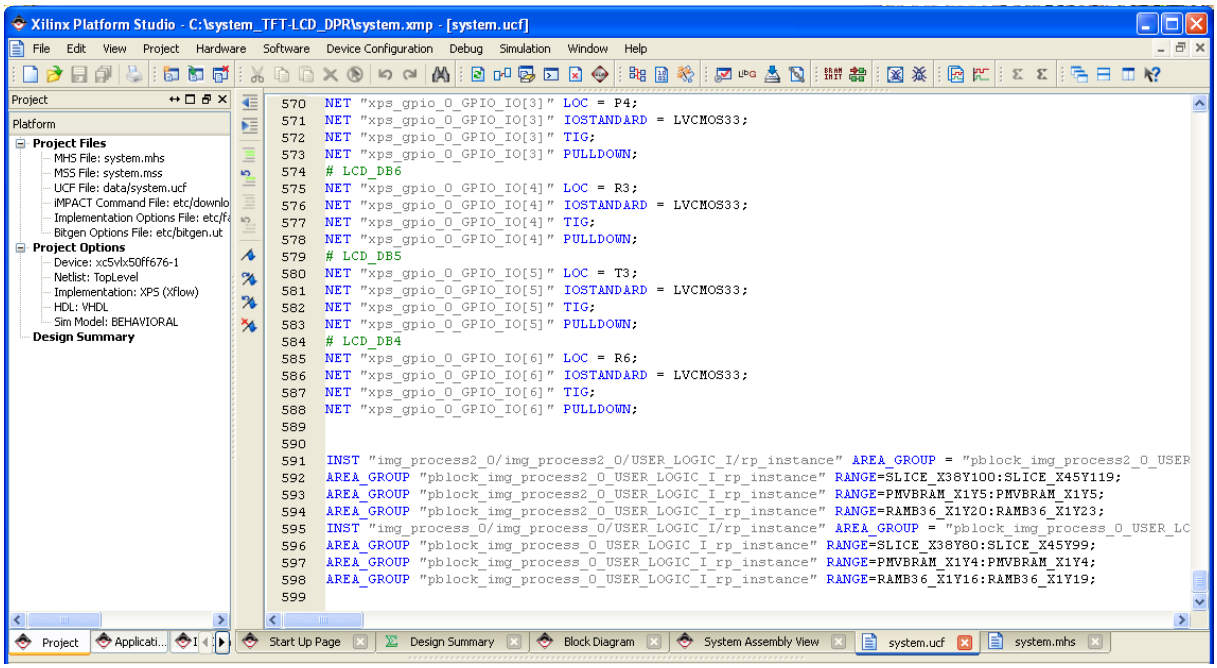


FIGURE 6.12: Extrait du fichier UCF

6.5.1.1 *Spécification des contraintes de temps*

Le concepteur a le choix de générer ce fichier grâce à l'outil ou le créer manuellement et saisir ses propres contraintes via un éditeur de texte. L'outil Xilinx ISE met à la disposition de l'utilisateur des outils graphiques comme le Constraints Editor et PlanAhead pour éditer ce fichier automatiquement. Les contraintes spécifiées dans ce fichier affectent la façon dont la conception logique est implémentée dans le périphérique cible. Il est possible d'écrire directement ce fichier sans passer par PlanAhead, en particulier pour des projets utilisant les mêmes types de connexion. Étant donné que nous ne nous intéressons pas particulièrement à la

spécification des contraintes de temps, nous invitons le lecteur à se référer à la documentation [103] pour plus de détails.

6.5.1.2 *Coordonnées des régions reconfigurables*

La contrainte de type AREA_GROUP est une contrainte d'implémentation de la conception. Elle permet le partitionnement du design dans les régions physiques pour le mapping, l'empaquetage (*packing*), le placement et le routage. Elle est généralement attachée aux blocs logiques dans le design.

La syntaxe requise dans le fichier UCF varie selon le besoin [103] :

- Pour définir un groupe de régions (Area Group) :
  INST "X" AREA_GROUP=*groupname* ;
- Pour attacher des contraintes à un Area Group :
  AREA_GROUP "groupname" RANGE=*range* ;

Sachant que *groupname* est un nom attribué à une partition d'implémentation afin de définir, de manière unique, un groupe. Tandis que *RANGE* : définit l'ensemble des ressources du périphérique qui sont valables pour placer la logique contenue dans le AREA_GROUP, de la même manière que *ranges* sont définies pour la contrainte LOC.

6.5.1.3 *Placement des ports physiques*

La contrainte LOC définit l'endroit où un élément de conception peut être placé dans un dispositif FPGA. LOC précise le positionnement absolu d'un élément de design sur la matrice FPGA. Il peut être un endroit unique (*single location*), un ensemble *range of locations*, ou d'une liste d'emplacements *(locations*). La contrainte LOC peut être spécifiée à partir du fichier de conception et aussi à partir du placement direct des déclarations dans un fichier de contraintes.

6.5.2 Définition des règles de transformation

En se basant sur les définitions précitées, nous avons spécifié les règles de transformations partant des concepts MARTE vers le fichier UCF. Le concept *HwReconfigurableRegion* est appliqué sur une classe qui définit une région reconfigurable dans le modèle physique. Tout d'abord, il faut sélectionner le nom de l'IP qui sera alloué à la région reconfigurable. Ce nom est utilisé pour remplir la ligne de commande du fichier UCF : INST "<% a.associatedIP.name % >. A partir de ce nom, nous définissons le AREA_GROUP selon les différents *RANGE* : Slice, DSP et Bram. Ainsi, les valeurs des coordonnées que l'utilisateur va saisir dans le modèle seront transformées tout en respectant la syntaxe du fichier UCF. Cette règle laisse ainsi tous les détails invisibles pour à l'utilisateur.

En outre, la dernière règle concerne le concept *HwPort* appartenant au sous-profil physique du paquetage HRM. Ce concept est appliqué sur les ports qui définissent les entrées/sorties de l'FPGA et leurs noms sont enregistrés dans le fichier de contraintes UCF. Lors de la définition de ce concept dans le chapitre 5, nous avons identifié deux cas de figures. Un premier cas où le

concepteur souhaite placer un composant (exemple : Uart) qui a été automatiquement généré par l'outil, alors son placement physique est automatique et pas besoin d'intervention manuelle. Le deuxième cas concerne l'ajout un composant qui sera relié, via ses ports, à l'environnement extérieur de l'FPGA. Dans ce cas, il est nécessaire de spécifier les différents pins alloués pour le port I/O de ce composant. Ensuite, l'outil de génération de code se charge de générer des instructions dans le fichier UCF. Ces instructions correspondent aux différents noms des pins attribués dans le modèle d'architecture physique.

Par conséquent, nous définissons cette règle qui permet au concepteur de relier le port de ce composant à une ou plusieurs pins. Il faut ainsi vérifier le nombre de pins utiles pour rattacher ce port moyennant l'expression allocatedPins.Size.

6.6 La transformation de IP-XACT vers MHS et MPD

La conception IP-XACT a été intégrée dans notre "environnement de conception" choisi, Sodius MDWorkbench, dans lequel les transformations de modèle de IP-XACT vers EDK Xilinx est effectuée. L'objectif de cet outil est de générer différents fichiers utilisés par EDK afin de configurer les composants hardware et software d'un système SoC basé sur FPGA. La configuration des composants est effectuée à travers la création du fichier MHS (Microprocessor Hardware Specification) qui contient un ensemble d'instances de certains composants ainsi que leurs paramètres. Ainsi, les transformations à partir de IP-XACT ont été définies dans un format prédéfini utilisé par les outils Xilinx afin d'obtenir les descriptions HDL du haut niveau ainsi que les références vers les IPs correspondants qui sont configurés durant cette phase.

6.6.1 Notre cible : Xilinx Platform Studio

L'environnement Xilinx EDK utilise un certain nombre de fichiers définis dans un format spécifique pour la description de la plateforme dit *Platform Specification Format* (PSF) [107]. Ce format permet de formaliser la description des différents composants dans le flot de conception Xilinx pour les systèmes basés sur des processeurs. Ces fichiers sont considérés comme une abstraction des implémentations des IPs et sont utilisés comme moyens pour configurer les IPs utilisés dans la plateforme via une description de conception.

La description VHDL d'un IP contient juste des informations concernant les ports d'entrées/sorties et dans le meilleur cas, des paramètres de type *generic* permettant au concepteur de paramétrer et personnaliser l'IP. Quand une implémentation VHDL doit être associée à une description de haut niveau (contenant typiquement des paramètres et des interfaces de bus), il est difficile de déterminer quels ports de l'IP appartiennent à une interface de bus précise.

Xilinx fournit un niveau de représentation intermédiaire, à travers le fichier Microprocessor Peripheral Description (MPD), dont un extrait est illustré par la figure 6.13. Ce fichier contient les informations de base d'un IP implémenté en Verilog/VHDL (generics, ports), en y ajoutant les attributs qui dépendent du flot utilisés pour la configuration. Les ports peuvent être regroupés ensemble en utilisant le concept de *bus interface*, offrant au concepteur le moyen de personnaliser

TABLE 6.3: Spécification des règles de transformation de MARTE vers le fichier UCF

Source	Condition et Traitement
MARTE :CodeFile	l'extension du FilePath doit être ***.ucf** Ainsi, le contenu du fichier sera rajouté au fichier **syst.ucf**
MARTE :HwReconfigurable-Region *% chercher le nom de l'IP alloué à la région reconfigurable %* *% la notation <% ...% > est utilisée pour évaluer une expression %*	la classe stéréotypée «HwReconfigurableRegion» a ← ModelSource[allocate] → select(Target=self) a.associatedIP INST "<% a.associatedIP.name % > "
% Ecrire les coordonnées de la zone selon le Range des Slices %	AREA_GROUP = "pblock_ < % a.associatedIP.name % >" if LB_Range ≠ null RANGE = "SLICE_X" LB_Range[0,0] "Y" LB_Range[0,1] ; RANGE = "SLICE_X" LB_Range[1,0] "Y" LB_Range[1,1] ;
% Ecrire les coordonnées de la zone selon le Range des DSPs *% Le même traitement est fait pour les BRAMs %*	AREA_GROUP = "pblock_ < % a.associatedIP.name % >" if DSP_Range ≠ null RANGE = "DSP_X" DSP_Range[0,0] "Y" DSP_Range[0,1] ; RANGE = "DSP_X" DSP_Range[1,0] "Y" DSP_Range[1,1] ;

MARTE :HwPort
% tester sur le nombre de pins utiles pour rattacher le port d'un composant
	if allocatedPins.Size = 1
	"NET " HwPort.name " LOC = < % allocatedPins [0] % > ;
	if allocatedPins.Size > 1
	for (i =0 ; i< (allocatedPins.Size -1) ; i++)
	"NET "HwPort.name " [< % i %>] LOC = < % allocatedPins [i] % > ;
	endFor

```
62  ## Bus Interfaces
63  BUS_INTERFACE BUS = SPLB, BUS_TYPE = SLAVE, BUS_STD = PLBV46
64
65  ## Generics for VHDL or Parameters for Verilog
66  PARAMETER C_BASEADDR = 0xffffffff, DT = std_logic_vector, BUS = SPLB,
67  PARAMETER C_HIGHADDR = 0x00000000, DT = std_logic_vector, BUS = SPLB,
68  PARAMETER C_MEM_WIDTH = 16, DT = INTEGER, RANGE = (8,16), PERMIT = BAS
69  PARAMETER C_SPLB_AWIDTH = 32, DT = INTEGER, BUS = SPLB, ASSIGNMENT = C
70  PARAMETER C_SPLB_DWIDTH = 32, DT = INTEGER, BUS = SPLB
71  PARAMETER C_SPLB_P2P = 0, DT = INTEGER, BUS = SPLB
72  PARAMETER C_SPLB_MID_WIDTH = 3, DT = INTEGER, BUS = SPLB
73  PARAMETER C_SPLB_NUM_MASTERS = 8, DT = INTEGER, BUS = SPLB
74  PARAMETER C_SPLB_NATIVE_DWIDTH = 32, DT = INTEGER, BUS = SPLB, ASSIGNM
75  PARAMETER C_SPLB_SUPPORT_BURSTS = 0, DT = INTEGER, BUS = SPLB, ASSIGNM
76  PARAMETER C_FAMILY = virtex5, DT = STRING
77
78  ## Ports
79  PORT SPLB_Clk = "", DIR = I, SIGIS = Clk, BUS = SPLB
80  PORT SPLB_Rst = SPLB_Rst, DIR = I, SIGIS = Rst, BUS = SPLB
81  PORT PLB_ABus = PLB_ABus, DIR = I, VEC = [0:31], BUS = SPLB
82  PORT PLB_UABus = PLB_UABus, DIR = I, VEC = [0:31], BUS = SPLB
83  PORT PLB_PAValid = PLB_PAValid, DIR = I, BUS = SPLB
84  PORT PLB_SAValid = PLB_SAValid, DIR = I, BUS = SPLB
85  PORT PLB_rdPrim = PLB_rdPrim, DIR = I, BUS = SPLB
86  PORT PLB_wrPrim = PLB_wrPrim, DIR = I, BUS = SPLB
87  PORT PLB_masterID = PLB_masterID, DIR = I, VEC = [0:(C_SPLB_MID_WIDTH-
88  PORT PLB_abort = PLB_abort, DIR = I, BUS = SPLB
89  PORT PLB_busLock = PLB_busLock, DIR = I, BUS = SPLB
```

FIGURE 6.13: Extrait du fichier MPD pour la description des IPs

l'utilisation de certaines interfaces en définissant des attributs tels que : *DataType*, *isValid*, *Permit*, etc.

De même, les paramètres et les options peuvent être établis de manière dépendante des autres paramètres, et rattachés à certains groupes spécifiques (ex : les paramètres qui affectent certaines interfaces parmi d'autres, des paramètres qui ne sont utilisés que lorsque l'utilisateur choisit une fonctionnalité précise, etc.). Un autre aspect important du fichier MPD, est de permettre l'ajout de certaines informations concernant un IP qui lui même est considéré comme une technologie ou un outil spécifique ; il permet ainsi la configuration de l'IP dans des scénarios différents et la personnalisation de leur comportement.

Les implémentations d'un IP qui sont abstraits par les fichiers MPD, doivent être paramétrées à un haut niveau ; ceci est possible à travers les instanciations des composants dans le fichier MHS (Microprocessor Hardware Specification).

6.6.1.1 Le fichier MPD

La figure 6.14 montre le modèle UML du fichier MPD. Dans ce modèle, un périphérique (*peripheral* ou *IP core* dans le jargon Xilinx) est défini comme un élément d'une plateforme, qui contient un ensemble d'attributs. La plupart de ces attributs peuvent correspondre directement

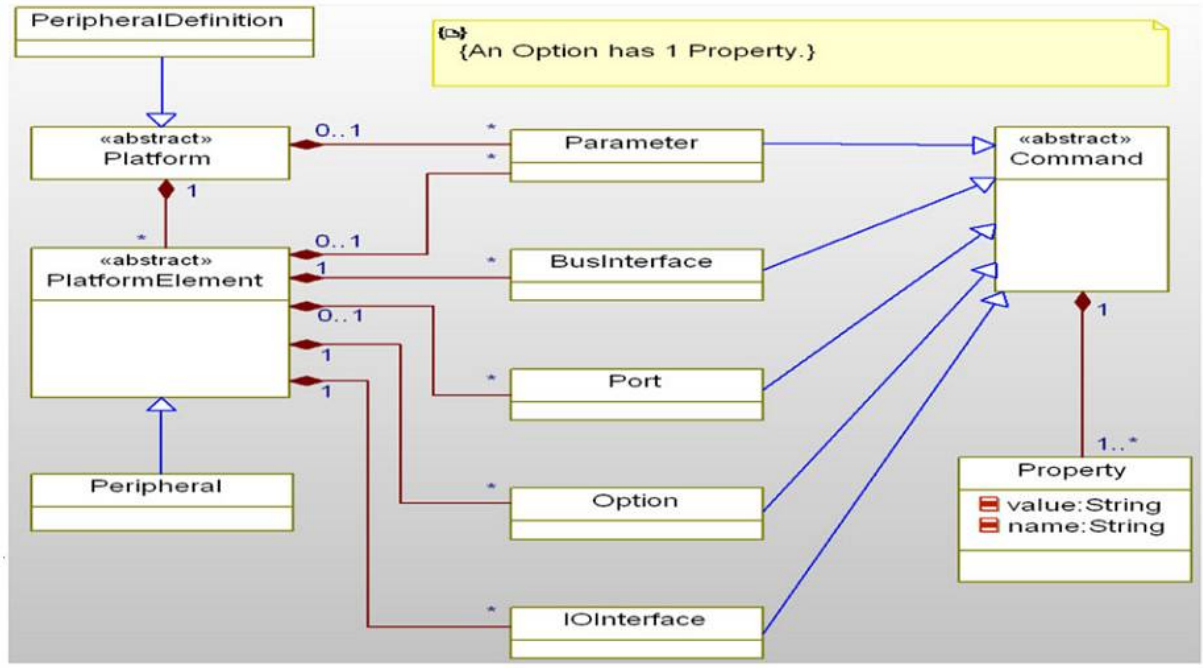


FIGURE 6.14: méta-modèle UML du fichier MPD de Xilinx PSF [65]

aux concepts dans la description de composant IP-XACT. En particulier, les paramètres et les ports sont combinés dans <*ModelParameters*> et les <*Ports*> dans l'élément <*Model*>, qui décrit les détails d'implémentation spécifiques du composant. De même pour les concepts <*busInterface*> et les <*options*>, qui correspondent aux mêmes concepts dans IP-XACT. Ces détails seront présentées dans la section qui suit.

6.6.1.2 Le fichier MHS

Ce fichier est utilisé par les outils Xilinx afin de créer une description top-level de la plateforme matérielle, comme le montre la figure 6.15. La description MHS contient les mêmes éléments du fichier MPD, avec pour seule différence : seuls les paramètres, les interfaces de bus et les ports qui sont nécessaires à la description haut niveau de l'IP, sont affichés. Ce résultat est obtenu en analysant les fichiers MPD et en vérifiant les paramètres valides (qui sont contrôlés par ceux du fichier MHS).

Le fichier MHS est créé à partir d'une description de conception de IP-XACT, qui contient : les instances de composant, les paramètres qui lui sont associés (appelés ConfigurableElements dans le jargon de IP-XACT). Les composants dans EDK sont associés aux fichiers d'implémentation grâce à l'utilisation du nom d'instance ainsi que les valeurs de la version matérielle. IP-XACT fournit un mécanisme pour déterminer les valeurs du dit VLNV (Version, Library, Name, Vendor). Ces informations, précédemment définies dans la transformation MARTE vers IP-XACT, permettent non seulement de relier les composants d'une description MARTE vers les composants de IP-XACT mais également vers leurs contre-parties dans EDK/VHDL. Les interfaces de bus sont déduites des étiquettes <BusRef> associées aux interconnexions des bus

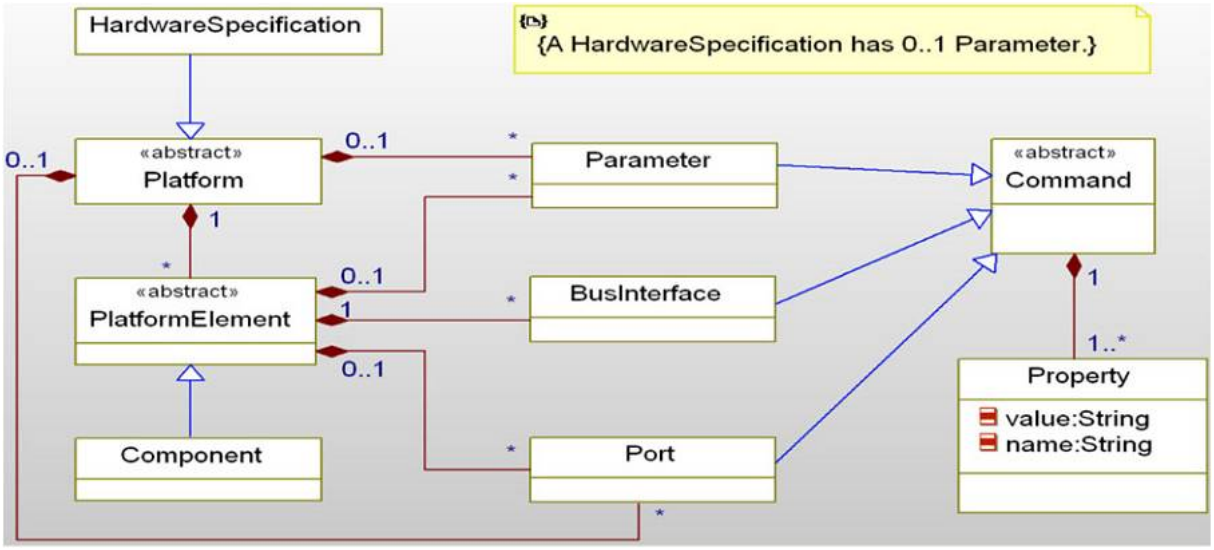


FIGURE 6.15: méta-modèle UML du fichier MHS de Xilinx PSF [65]

entre les instances des composants. La figure 6.16 illustre un extrait de certains concepts des interfaces de bus qui correspondent à ceux du fichier MHS.

6.6.2 Les règles de transformations de IP-XACT <–>MHS et MPD

La figure 6.17 montre un extrait du méta-modèle IP-XACT proposé par Sodius MDWorkbench. Ce méta-modèle est utilisé afin d'effectuer différentes transformations à partir de IP-XACT vers différentes cibles, dans notre cas vers les fichiers Xilinx MHS et MPD. De plus, il permet également de promouvoir la réutilisation des IPs en transformant les descriptions IP-XACT vers des modèles MARTE, comme nous l'avons détaillé dans la transformation MARTE et RecoMARTE vers IP-XACT.

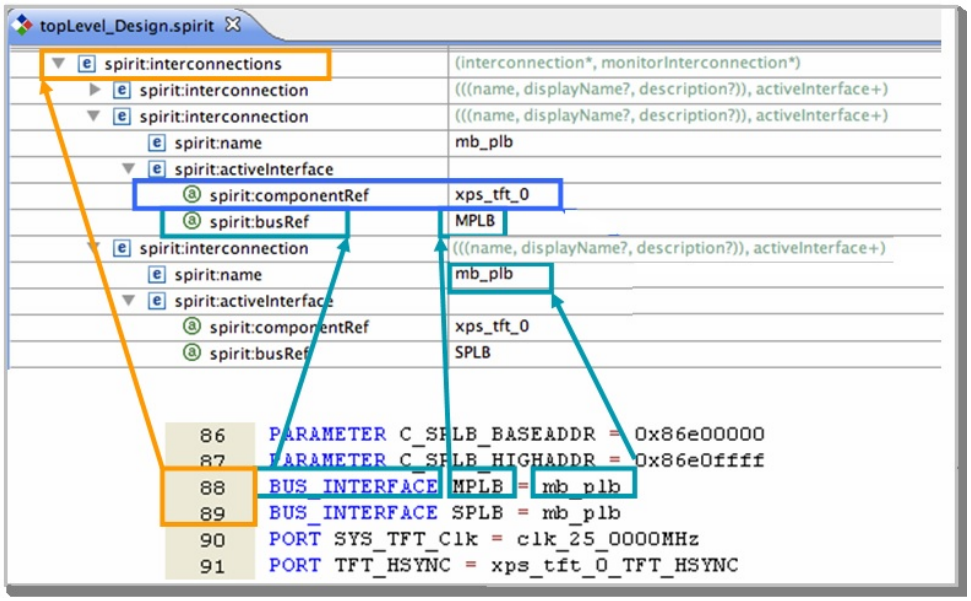


FIGURE 6.16: Correspondance de concepts entre IP-XACT et MHS

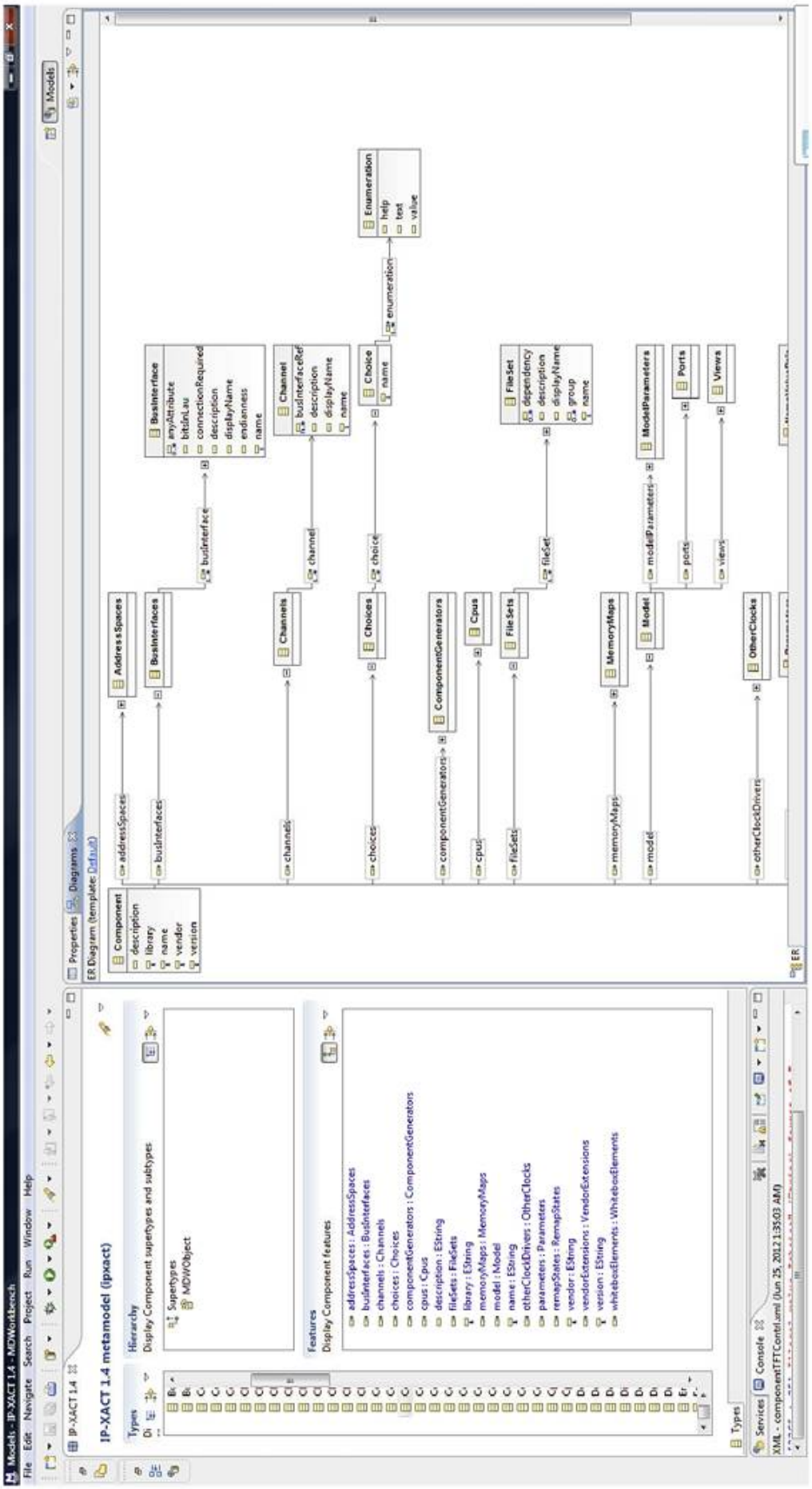


FIGURE 6.17: Exemple du méta-modèle IP-XACT présenté par MDWorkbench de Sodius[65]

MPD Command	IP-XACT Component Counterpart
BUS_INTERFACE	spirit:busInterface if (parameter:interface_type) = ***bus_interface*** spirit:parameters:spirit:
INTERFACE_TYPE	parameter:Interface_Type
BUS	parameter:bus:value
BUS_STD	parameter:bus_std:value
BUS_TYPE	parameter:bus_type:value
GENERATE_BURSTS	parameter:generate_bursts:value
ISVALID	parameter: value = ((Depedency(param_id)) ? 0:1)
IO_INTERFACE	spirit:busInterface if (parameters:interface_type) = ***hier_interface*** spirit:parameters:spirit:
INTERFACE_TYPE	parameter:Interface_Type
IO_IF	parameter:IO_IF
IO_TYPE	parameter:IO_TYPE

MPD Command	IP-XACT Component Counterpart
PORTS	***model:Ports*** spirit:port:spirit
NAME	name
DIR	wire:spirit:direction
VEC	vector:left - vector:right
PORT PARAMS	spirit:port:spirit:vendorExtensions:spirit:Parameters
PORT_TYPE	parameter:portType (bus_signal, hier_signal, ah-hoc_signal) (bus_signal, hier_signal, ah-hoc_signal)
PORT_GROUP	parameter:portGroup
SIGNAL_TYPE	parameter:signalType (clk, interrupt, bus, io, three_st)
ISVALID	parameter:isValid if ((Depedency(param_id)) ? 0:1)
PERMIT	if (param:port_type = "hier_signal" and param:permit = user) PERMIT = base_user
BUS	if (portType = "bus_signal") then parameter:bus_name
IO_IF	if (portType = "hier_signal" or "ad-hoc_signal") then parameter:bundle_name
IO_IS	if (portType = "hier_signal") then parameter:bundle_name

FIGURE 6.18: Règles de transformation de IP-XACT de/vers MPD : busInterface et ports

Les détails et les explications de ces transformations ont été publiés dans [65].

6.6.2.1 *La transformation IP-XACT <=>MPD*

La figure 6.18 illustre les règles de transformation de modèle IP-XACT vers le fichier EDK MPD. Comme c'est déjà mentionné, le fichier MPD contient tous les paramètres, les ports et les interfaces basées sur bus d'un IP.

6.6.2.2 *La transformation IP-XACT <=>MHS*

Les composants dans EDK sont associés aux fichiers d'implémentation en utilisant le nom de l'instance et les valeurs de la version du matériel.

IP-XACT permet, grâce aux informations du VLNV, de relier les composants d'une description MARTE vers IP-XACT et enfin vers EDK/VHDL. La figure 6.19 résume les règles de transformation de IP-XACT vers MHS, concernant les paramètres, les interfaces basées sur bus et les ports.

6.7 Bilan et évaluation

Comme précédemment expliqué dans le chapitre 2, la complexité de la co-conception des SoC a connu une évolution rapide ces derniers temps. Ce qui a créé le besoin de trouver des méthodologies de conception efficace qui prennent en compte les différents challenges à relever à savoir :l'évolution technologique, la complexité, le temps de mise sur le marché, etc. ; tout en réduisant le temps de développement.

De plus, les caractéristiques définissant la reconfigurabilité dans ces systèmes ont augmenté leur flexibilité pour faire face à des environnements et des changements dans les besoins des utilisateurs encore plus exigents, au prix d'une complexité accrue. Afin de répondre à ces exigences, l'élévation du niveau d'abstraction de la modélisation s'avère une solution bénéfique car elle offre une modélisation indépendante des détails d'implémentation, réutilisable et maintenable.

MHS Command	IP-XACT Designt Counterpart
PARAMETERS	**CONFIGURABLE ELEMENTS** Desing:componentInstances:ComponentInstance
INSTANCE	spirit:InstanceName
HW_VER	spirit:componentRef_spirit:Version
PARAM_i = VALUE_i	spirit:configurableElementValue spirit:referenceId = value
BUS INTERFACES	**ACTIVE INTERFACES(Design:interconnections)**
BUS INTERFACE i INTERFACE = bus_std	for each(interconnection) if (componentRef = "instanceName") activeInterface:busRef = interconnection:name
PORTS	**INTERNAL PORT REFERENCES (Design:adHocConnections)**
internal ports PORT_i = VALUE_i	for each(adHocConnection) if (componentRef = "instanceName") internalPortRef:portRef = adHocConnection:name
external ports PORT_i = VALUE_i	for each(adHocConnection) if (componentRef = "instanceName") externalPortRef:portRef = adHocConnection:name

FIGURE 6.19: Règles de transformation de IP-XACT de/vers MHS : paramètres, busInterface et ports

Nous avons donc proposé une approche de co-conception de systèmes sur puce basés sur FPGA qui intègre tous les aspects précités et permet en plus la réutilisation des IPs. Ce qui offre la possibilité aux concepteurs de spécifier leurs systèmes en modélisant l'application, l'architecture et leur allocation déployée qui intègre la représentation intermédiaire de déploiement des IPs. Cette modélisation est représentée graphiquement grâce à l'utilisation d'un langage unifié (UML) et un profil standardisé (MARTE) ce qui augmente la compréhension du système.

Par conséquent, notre approche basée sur une montée en abstraction permet d'éviter de modéliser les détails liés à l'implémentation et de réutiliser un même modèle d'application pour différentes cibles d'exécution. De plus, elle a fait usage de l'approche basée sur les composants qui favorise la stratégie de la séparation des préoccupations (*Separation of concerns strategy*). Ceci permet donc de créer séparément le modèle de l'architecture logique et le modèle d'application dont les composants seront également définis indépendamment pour faciliter leurs reconfiguration.

Par ailleurs, la spécification des niveaux intermédiaires s'avère un travail non aisé car il faut définir avec précision l'objectif de chacun d'entre eux. Ainsi, le passage d'un niveau de description à un autre est automatisé de manière à raffiner une description en la rapprochant du niveau de description de l'implémentation. Pour ces raisons, l'usage de l'IDM nous a permis d'*automatiser* notre flot de conception proposé. Tous les méta-modèles réalisés ainsi que les transformations de modèles spécifiées précédemment, peuvent être *réutilisés* pour les besoins d'une nouvelle implémentation de l'application sur une nouvelle plateforme. Il est à noter que notre approche est également basée sur un standard générique MARTE moyennant certaines extensions pour obtenir le profil RecoMARTE. Cet aspect de *généricité* nous laisse complètement indépendant de toute technologie. En effet, suivant notre flot de conception, le modèle d'application, d'architecture logique et de leur allocation restent des modèles indépendants de

la technologie (dans notre cas on parle de technologie Xilinx). Seul le modèle de l'architecture physique de l'FPGA décrit la plateforme choisie, dans notre exemple c'est Xilinx, mais ceci devrait être modifié dans un autre cas de figure.

Notre choix a ciblé les FPGAs Xilinx, non seulement parce que ce sont les plateformes disponibles dans le projet FAMOUS, mais aussi parce que Xilinx est le seul fournisseur à traiter tous les aspects de la RDP sur les FPGAs. Par conséquent, notre méthodologie permet de générer les fichiers qui serviront d'entrée pour l'environnement EDK de Xilinx. Dans le cas où le concepteur souhaite modifier ou ajouter un composant, il lui suffit juste d'effectuer cette modification dans le modèle du haut niveau de l'architecture. Grâce à l'automatisation notre flot, les détails de transformations seront transparents pour l'utilisateur. Ainsi, les fichiers concernés (MHS, MPD et UCF) seront modifiés et re-générés automatiquement sans l'intervention de l'utilisateur.

Enfin, notre méthodologie nous permet de traiter l'aspect reconfiguration dynamique très tôt dans le flot, contrairement à flot de Xilinx qui ne le propose que lors de la phase de placement des zones reconfigurables. En effet, dès la création du modèle d'application, les concepts dans RecoMARTE nous permettent de définir quelles sont les tâches de l'application qui sont reconfigurables. Cette information présentée dans le chapitre 5, est très utile car elle nous permet de connaître quel est le composant qui va exécuter cette tâche ; ce composant va être ensuite placé dans une zone reconfigurable du modèle physique de l'FPGA. Ce genre d'information suivant le flot permet, bien évidemment, de guider et d'orienter l'utilisateur/concepteur grâce aux différentes contraintes que nous avons précédemment spécifiées. Par conséquent, le concepteur se voit capable d'éviter ce type d'erreurs durant toutes les phases de notre flot.

6.8 Conclusion

Dans ce chapitre, une approche de conception basée sur l'IDM a été proposée. Elle vise à 1) rendre transparents aux concepteurs les détails d'implémentation de bas niveau, 2) automatiser notre flot de conception haut niveau vers la génération du code 3) permettre ainsi d'accélérer la phase de conception 4) éviter les erreurs dues à la manipulation directe des ces détails.

Tout le mécanisme de passage d'une modélisation à haut niveau jusqu'à la génération de fichiers, sera géré par les transformations de modèles et l'outil de génération de code, ce qui facilite et accélère la conception. Pour arriver jusqu'à la génération du code, une chaîne de transformation a été utilisée. Le modèle final en MARTE résultant du flot de conception proposé est donné comme entrée à cette chaîne. Nous passons ainsi d'un modèle MARTE vers une description intermédiaire en IP-XACT pour générer finalement des fichiers décrivant le système complet dans l'environnement XPS de Xilinx.

La figure 6.20 illustre les différents modèles en entrée et en sorties des différentes transformations. Comme nous l'avons évoqué, le modèle final résultat de notre flot de conception est donné en entrée à la première transformation UML vers MARTE moyennant les extensions de la RDP. Ensuite, la transformation RECOMARTE vers IP-XACT a traduit tous les concepts de

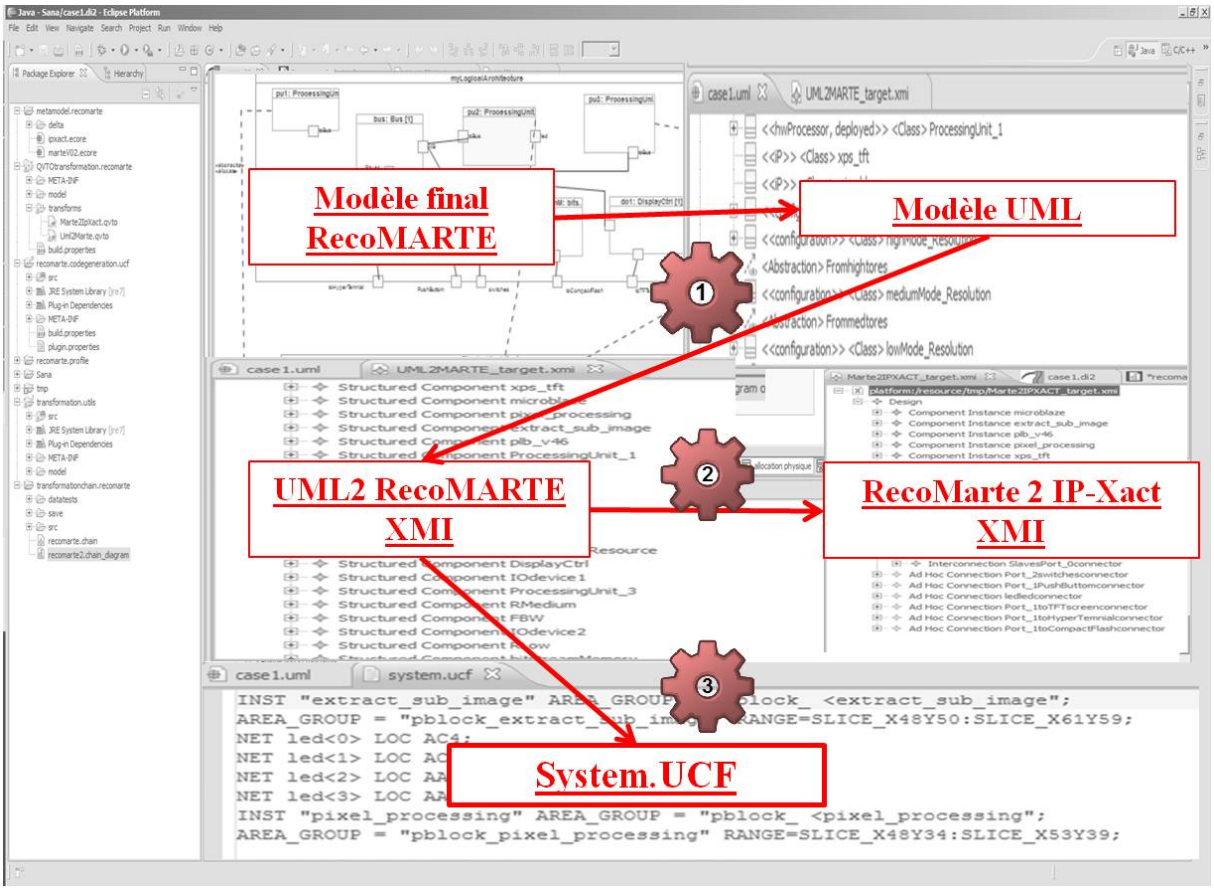


FIGURE 6.20: Automatisation de la chaîne via les transformations de modèles

RECOMARTE vers un modèle conforme au méta-modèle IP-XACT. Cette transformation permet de promouvoir la réutilisation d'IP dans notre méthodologie en reliant les composants du haut niveau à leurs équivalents dans IP-XACT. La troisième transformation nommée RECOMARTE 2 UCF est de type modèle vers texte. Elle a traduit certains concepts de la modélisation physique de l'FPGA vers des instructions textuelles saisies dans le fichier UCF. Ces transformations ont été développées moyennant le langage QVTo et sont intégrées dans l'outil MDWorkBench de notre partenaire industriel Sodius. Ce dernier s'est chargé de compléter et finaliser le flot FAMOUS en développant la dernière transformation de la chaîne présentée. Cette transformation permet le paramétrage des IPs afin de contrôler les interfaces, les ports et les bus et facilite ainsi le passage vers les fichiers de MHS et MPD.

Ce chapitre résume la première contribution de la partie "Validation et Automatisation du flot" illustré dans la figure 6.21.

Nous allons passer au prochain chapitre afin de mettre en œuvre un exemple de système reconfigurable, dont la conception va exploiter la méthodologie présentée dans cette étude.

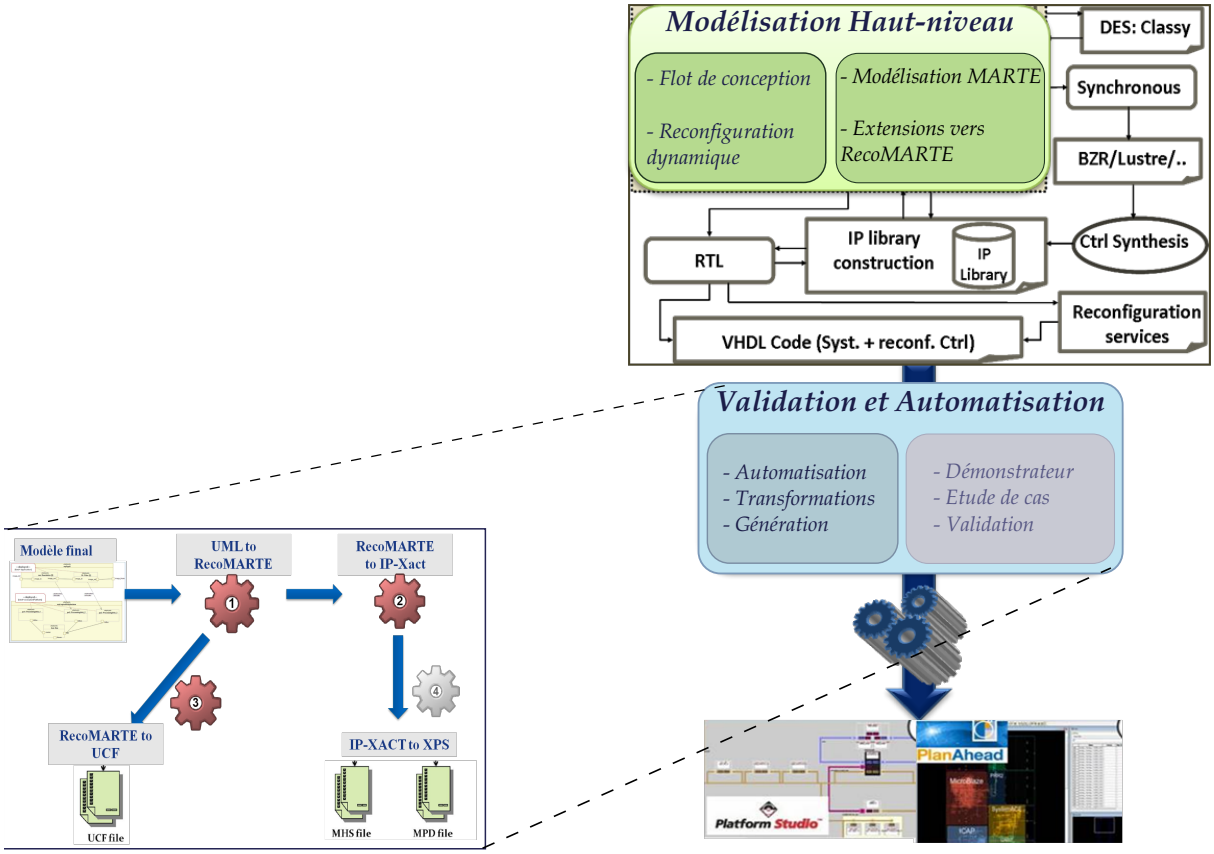


FIGURE 6.21: Contribution 3 : Transformation et Automatisation du flot

CHAPITRE 7

MODÉLISATION DU CAS D'ÉTUDE

7.1 Introduction

Une des tâches majeures dans le projet FAMOUS est de valider les théories développées par tous les acteurs du projet.

Dans ce chapitre, nous présentons une étude de cas qui nous permet de mettre en œuvre notre méthodologie et d'appliquer la modélisation en MARTE et RecoMARTE. La modélisation suivra le flot de conception proposé dans le chapitre 4 à savoir la modélisation à haut niveau de l'application, l'architecture et leur allocation déployée ainsi que la modélisation des différents

IPs de la bibliothèque. De plus, nous nous intéressons aux systèmes basés sur FPGA et nous présentons ainsi la modélisation de la plateforme physique des FPGAs. Étant donné que notre méthodologie prend en compte les FPGAs dynamiquement reconfigurables, nous présentons également la modélisation du contrôleur de la RDP. Ainsi, dans ce chapitre, nous présentons l'application de traitement d'image et nous détaillerons les différents niveaux de modélisation selon notre flot.

7.2 Présentation de l'étude de cas

Nos travaux décrits dans ce manuscrit viennent s'intégrer dans le flot de conception de FAMOUS ; un flot facilitant la *modularité*, la *réutilisabilité* et l'*automatisation*. Ce flot est modulaire car les différents acteurs du projet FAMOUS viendront greffer leurs modules dessus. De plus, les modèles présentés sont indépendants des détails d'implémentation et éventuellement réutilisables. Afin d'assurer l'automatisation qui est un point clé permettant d'améliorer la productivité des concepteurs, la méthodologie proposée est donc basée sur une approche d'Ingénierie Dirigée par les Modèles. Cette approche permet d'automatiser la génération du code à partir de modèles de haut-niveau d'abstraction et fait usage du profil standard MARTE permettant de rendre les détails techniques de bas niveau transparents aux concepteurs et d'automatiser la génération du code.

7.2.1 L'application

Dans cette étude de cas, le choix d'exploiter le flot d'acquisition vidéo se justifie par le fait que nous recherchons un moyen visuel et didactique de présenter les travaux. L'établissement d'un framework pour le déploiement de systèmes de traitement vidéo est donc légitime. Les modules dédiés à la capture et à l'affichage de la vidéo sont implémentés statiquement et fournis par le framework en tant que brique logicielle/matérielle pour le support de conception et du déploiement.

7.2.2 La plateforme d'exécution

La réalisation du démonstrateur repose sur le choix du matériel adéquat : un FPGA dynamiquement reconfigurable et des entrées/sorties vidéo. Ainsi, le choix s'est porté sur deux cartes de développement Xilinx : une XUP-V5, basée sur un FPGA Virtex 5, et une ML-605, basée sur un FPGA Virtex 6. La première carte a été initialement choisie par le fait qu'elle soit maîtrisée par l'ensemble des partenaires du projet FAMOUS. De plus, le démonstrateur a été testé avec succès sur cette carte, néanmoins la résolution du traitement vidéo a été limitée à 640x480.

Par la suite, le matériel a été modernisé et amélioré dans le but de montrer la portabilité du flot. La ML605 a été donc choisie, puis augmentée d'une carte de traitement vidéo rendant ainsi le tout capable de traiter un flux vidéo en haute définition. Il est important de noter que les éventuels conflits de portabilité concernent essentiellement la spécification matérielle. Cette question dépasse donc le cadre de cette étude, vue que tout est logiciel. L'implémentation de notre étude

de cas est présentée dans la figure 7.1 et illustre ainsi la carte ML605, les entrées/sorties vidéo ainsi que l'image traitée affichée à l'écran.

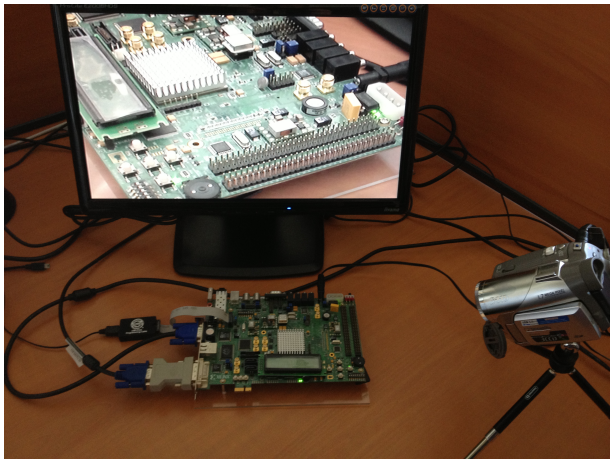


FIGURE 7.1: Implémentation de notre étude de cas avec une entrée vidéo

7.3 Modélisation à haut-niveau de l'étude de cas

Dans ce qui suit, nous présentons la modélisation notre application proposée pour valider notre méthodologie. Ces modèles suivront les différents niveaux de notre flot (double Y) proposé précédemment et feront usage des deux profils MARTE et RecoMARTE pour la modélisation des concepts de la RDP. Ainsi, nous partons de la modélisation de la première partie du modèle Y, à savoir le modèle d'application (structurel et comportemental), l'architecture logique et le niveau de l'allocation déployée de l'application sur les composants de l'architecture. Ensuite, les différents IPs de la bibliothèque sont modélisés dans des diagrammes de classe. Par la suite, la plateforme physique de l'FPGA sera modélisée ainsi que le contrôle de la reconfiguration dynamique.

7.3.1 Modélisation de l'application

Le modèle d'application dans MARTE vise à définir les fonctionnalités ainsi que les communications des tâches d'un système donné. Cependant, comme précédemment mentionné, la partie communication n'est pas prise en compte dans notre méthodologie puisqu'il s'agit d'un module traité par un autre acteur du projet FAMOUS.

La spécification du modèle d'application comporte la modélisation structurelle du graphe de tâches dédiées au traitement de données, ensuite d'une modélisation comportementale décrivant le comportement de ces tâches.

Notre application vise un traitement d'un flux de vidéo, comprenant deux tâches reconfigurables, exécutées en séquence de telle manière que la première communique le résultat de son traitement à la deuxième. Nous allons donc implémenter deux tâches, *Resolution* et *Filter*. La

première a pour rôle de modifier la taille de l'image du flux vidéo alors que la deuxième tâche permet d'adapter ses couleurs. La figure 7.2 illustre le modèle d'application en RecoMARTE.

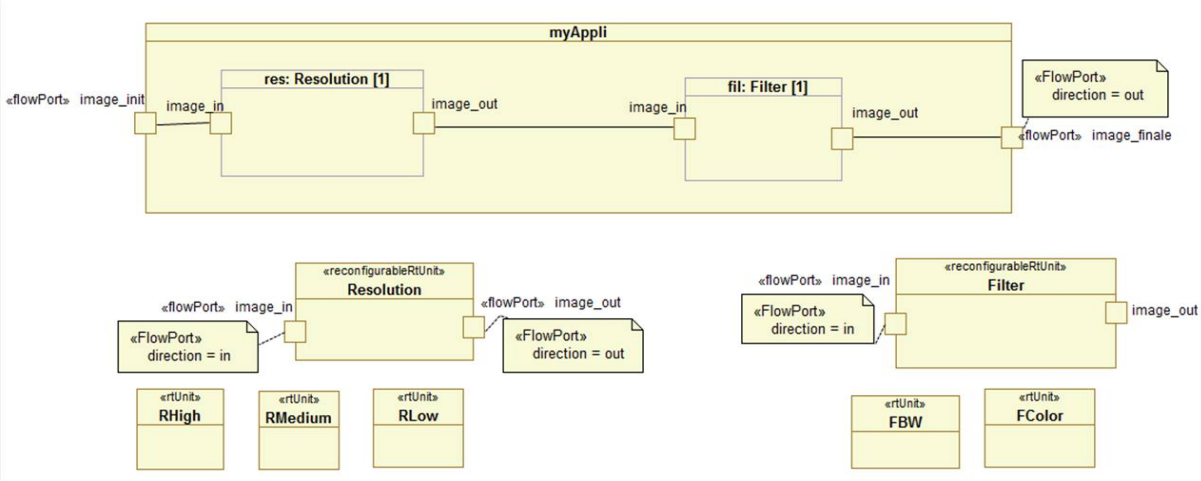


FIGURE 7.2: Modélisation de l'application

Les classes *Resolution* et *Filter* sont stéréotypées «*ReconfigurableRtUnit*». Ce stéréotype, précédemment défini dans le chapitre 5, permet de distinguer les tâches reconfigurables qui seront exécutées par des unités de traitement à définir dans le modèle d'architecture. Une des contraintes à définir est d'obliger le concepteur de placer (allouer)ces unités de traitement dans des zones reconfigurables et ce au niveau de la modélisation de l'allocation physique. De plus, les RtUnit reconfigurables possèdent deux types de ports stéréotypés «*FlowPort*» dont la direction est soit*in* soit *out*.

7.3.2 Modélisation du comportement

Une fois la structure de l'application est définie, nous nous intéressons dans ce qui suit, à définir son comportement. Le comportement des tâches est défini par des modes qui appartiennent à des ModesBehaviors. Ainsi, deux ModeBehaviors sont définis, *ResolutionModes* et *FilterModes* et dédiés respectivement aux comportements des tâches *Resolution* et *Filter*. Les transitions permettant de passer d'un mode à un autre sont typées «*ModeTransition*» :

- Pour le ModeBehavior *ResolutionModes*, les événements sont *goMedium*, *goHigh* et *goLow*.
- Pour *FilterModes*, les événements sont *goColor* et *goBW*.

Ces ModeBehaviors, illustrés par la figure 7.3, seront employés lors de la modélisation du contrôleur.

7.3.3 Les différentes configurations de l'application

La définition des Configurations dans MARTE permet d'identifier les implémentations possibles des tâches de l'application comme le montre la figure 7.4. Il s'agit d'un diagramme de configuration contrôlé par des machines d'états.

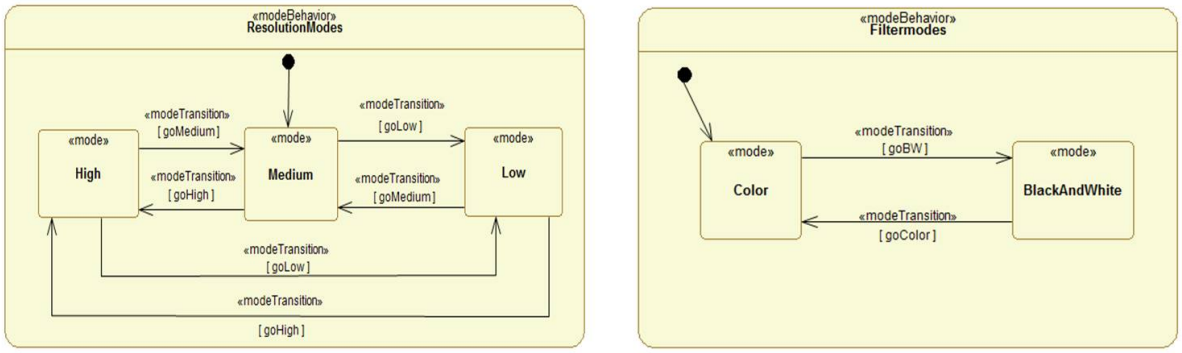


FIGURE 7.3: Modélisation du comportement des tâches via le concept *ModeBehavior* dans MARTE

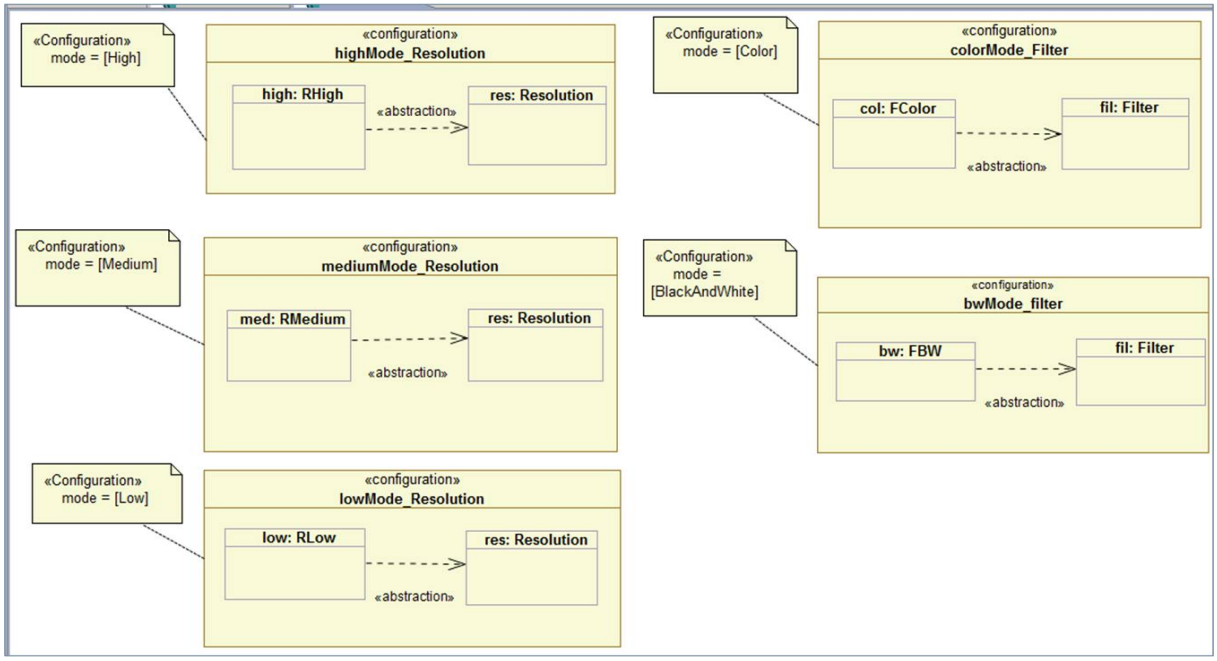


FIGURE 7.4: Modélisation des liaisons tâches/implémentations via la notion de *Configuration* dans MARTE

Ainsi, nous pouvons définir trois implémentations pour la tâche *Resolution* (High, Medium, Low) et deux pour la tâche *Filter* (Color et BW). De plus, chaque Configuration définie est rattachée à un mode. Ce mode est spécifié grâce à la propriété *mode* du stéréotype «Configuration» de MARTE. Ces implémentations sont considérées, au bas niveau, comme des bitstreams partiels obtenus à partir d'un ensemble de modules PRM (Partial Reconfigurable Modules) synthétisés. En d'autres termes, les tâches reconfigurables sont assimilables aux partitions reconfigurables qui seront définies dans le code HDL du top-level, lors de la phase de conception. La différence est que le concepteur dans UML MARTE ne s'intéresse pas aux détails d'implémentation du bas niveau.

7.3.4 Modélisation de l'architecture logique

L'architecture logique permettant la construction du système doit être définie. Ceci est possible via la construction d'un diagramme de composite illustré par la figure 7.5. Dans cette figure, les composants sont modélisés dans le même diagramme que leurs instances et sont stéréotypés avec les concepts du sous-profil HRM logique. Le bus possède trois ports différents : master, slave et MasterSlave (MS). Ce dernier le relie aux PU2 et PU3 dédiés à l'exécution des tâches reconfigurables définies dans le modèle d'application. Ce niveau de modélisation représente une des premières étapes de la méthodologie du co-design (modèle Y) qui précède la phase d'association et de déploiement. Par conséquent, les blocs modélisés à ce niveau ne représentent pas des blocs d'IPs spécifiques ou une technologie particulière, mais plutôt des composants élémentaires tels que les processeurs, les mémoires et les périphériques d'entrée-sortie pour les communications. Dans le cas de notre étude, il s'agit d'une entrée vidéo et des périphériques de sortie.

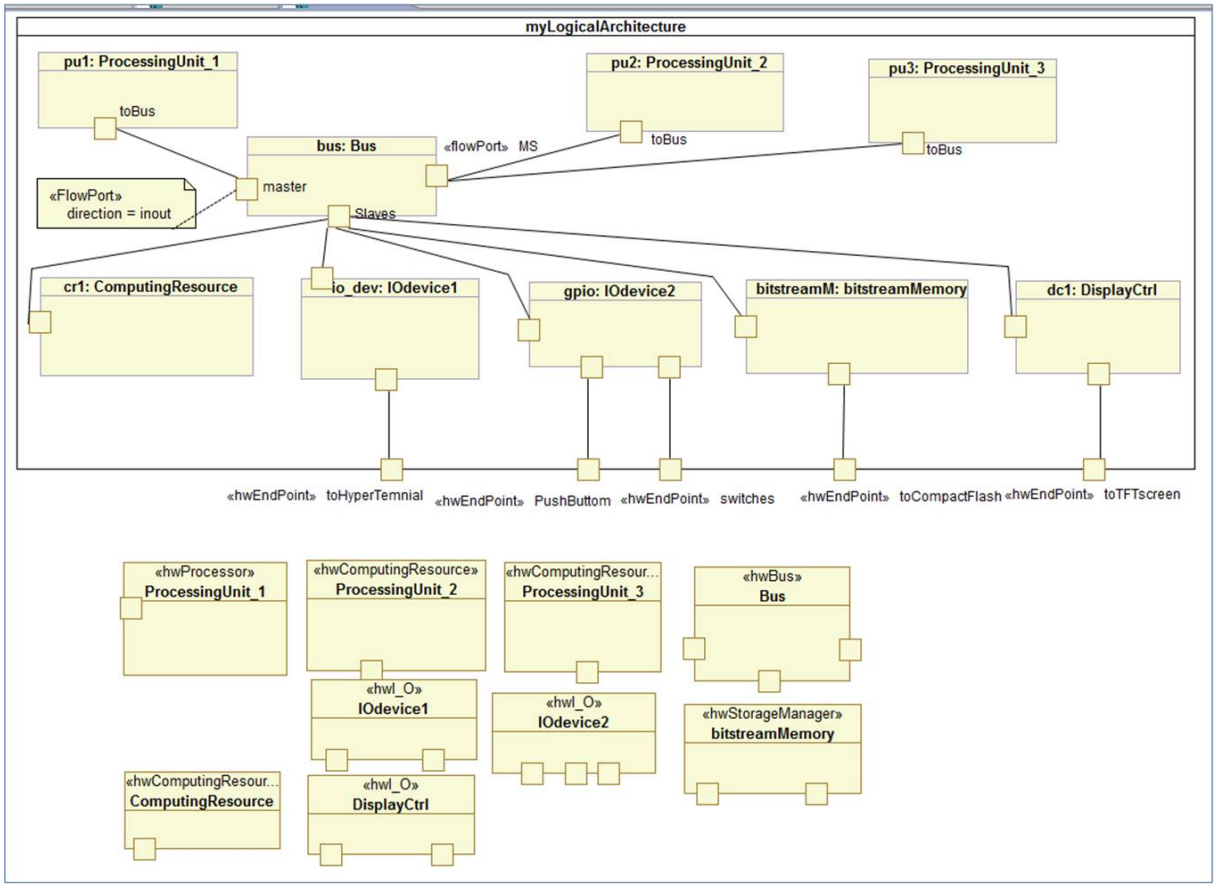


FIGURE 7.5: Modélisation de l'architecture logique

Ces derniers sont stéréotypés «*HwEndPoint*». Il s'agit d'un concept générique qui symbolise les points d'extrémité d'une *HW_Resource*.Il sert ainsi, d'interface pour communiquer avec les autres *HW_Resources* par le biais d'un *HW_Medias*.

Les composants qui sont reliés aux différents HwEndPoint sont les suivants :

- Un premier IOdevice typé «*HwI_O*» qui est relié à un hyper Terminal.
- Un deuxième IOdevice dont le rôle est celui d'un GPIO (General Purpose Input/Output) connecté à des switches et boutons afin que l'utilisateur puisse produire des événements manuellement.
- Un gestionnaire de mémoire typé «*HwStorageManager*» qui permet de gérer l'accès et/ou le contenu de certaines mémoires contrôlées ; dans notre exemple, il s'agit d'une Compact Flash.
- Le dernier périphérique d'entrée/sortie est nommé *Display Controller* qui sera défini par la suite, en tant que UART (Universal Asynchronous Receiver Transmitter) afin d'être relié à un écran TFT.

7.3.5 Modélisation de l'allocation déployée

Le niveau de l'allocation déployée contient un ensemble de vues pour la génération de modèles exécutables. Nous nous intéressons à la génération de la Netlist de l'architecture logique du top-level pour la spécification d'une plateforme donnée. De plus, nous donnons également les descriptions des IPs de cette plateforme : les IPs statiques et dynamiquement reconfigurables.

Comme précédemment mentionné, notre objectif est de générer des informations de type fonctionnel et structurel qui seront utilisées par la suite comme entrée dans le flot de conception de la RDP. Ainsi, le diagramme illustré par la figure 7.6 présente l'allocation déployée de l'application déployée sur l'architecture logique déployée (la figure ici présente un extrait de l'architecture logique afin d'éviter l'encombrement). En d'autres termes, les tâches de l'application ainsi que les composants de l'architecture sont stéréotypés «*deployed*» ; un stéréotype appartenant au paquetage Déploiement du profil RecoMARTE et présenté dans le chapitre 5. Ce stéréotype indique que chaque composant élémentaire est associé à un IP appartenant à la bibliothèque. Les composants UML MARTE sont classifiés en deux groupes en utilisant l'attribut *deployedEndKind* du stéréotype «*deployed*». Ainsi, dans le diagramme d'allocation déployée (Cf. figure 7.6), les composants élémentaires de l'application sont stéréotypés «*deployed*» dont la valeur de l'attribut {*kind* = *application* } alors que les composants de l'architecture logique sont identifiés avec la valeur {*kind* = *executionPlateform* }. Cette classification induit d'importantes conséquences au niveau de la génération de la chaîne FAMOUS, étant donné qu'elles sont traitées différemment afin de produire les Netlists des modules reconfigurables ainsi que la description top-level du système.

Ce niveau de modélisation, discuté précédemment, permet de lier les modèles MARTE aux composants IP-XACT qui contiennent plus d'information décrivant les IPs (paramétrage, réutilisation et les interconnexions).

La figure 7.7 illustre la modélisation des choix de configuration de l'allocation des tâches sur les *PUs* (dans cet exemple : processing units pu2 et pu3). Ces *PUs* peuvent être des co-processeurs ou des accélérateurs matériels ; le choix de type d'implémentation se fait également

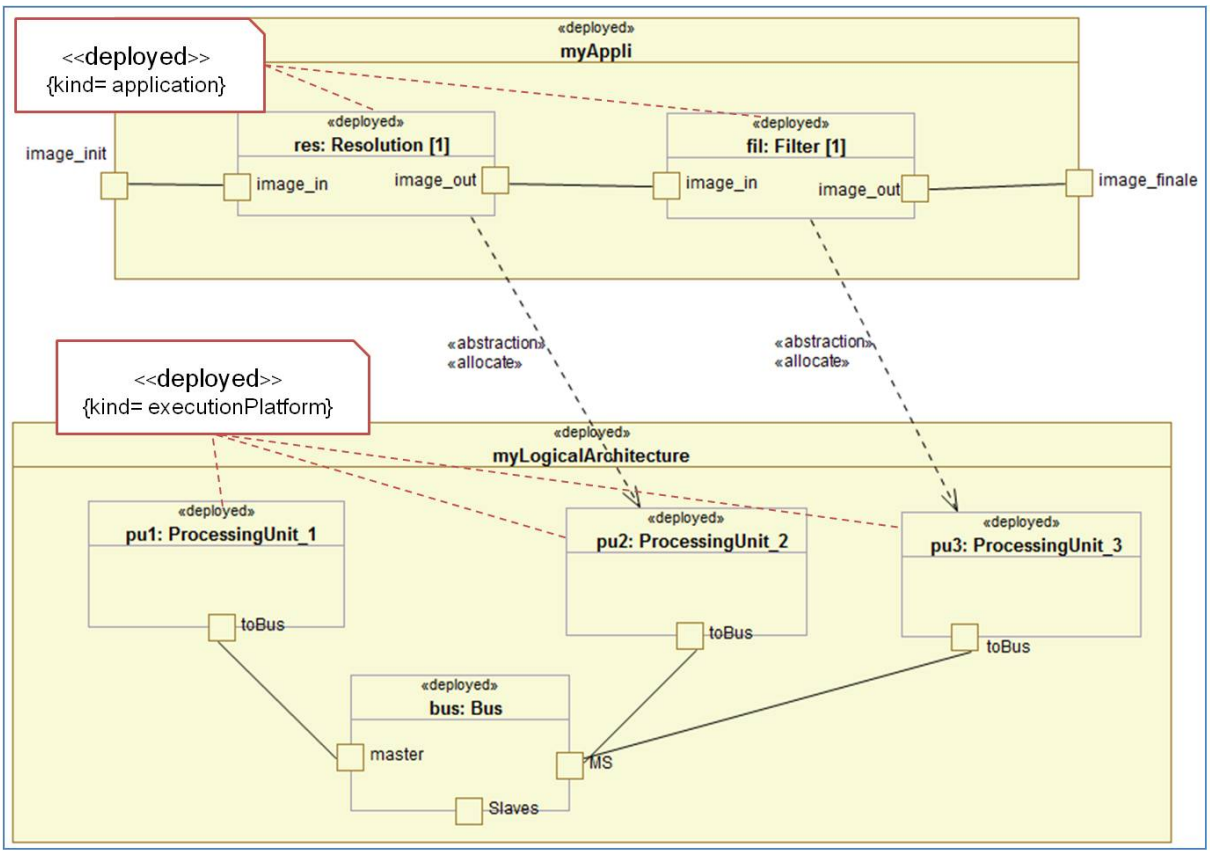


FIGURE 7.6: Modélisation de l'allocation déployée des tâches déployées sur l'architecture déployée

à ce niveau de modélisation.

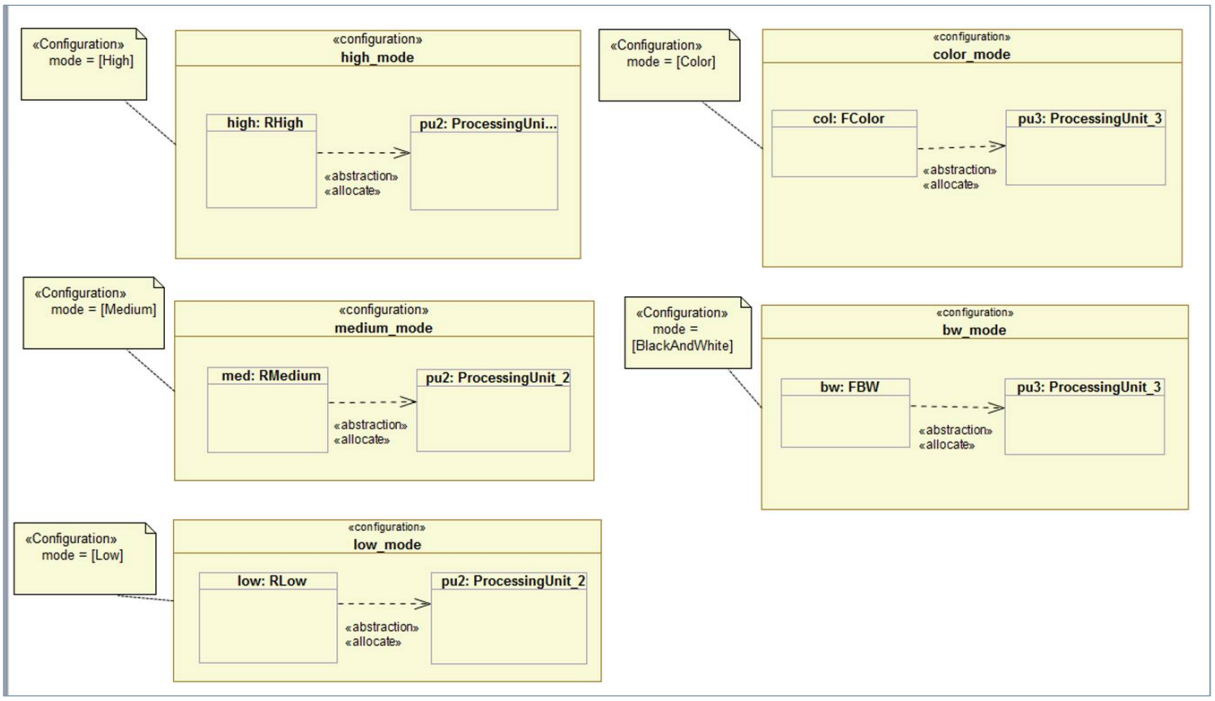


FIGURE 7.7: Modélisation des choix de configuration de l'allocation des tâches sur les *PUs*

Un PU peut contenir plusieurs configurations des IPs afin de traiter une partie de l'image reçue en entrée. Pour ce faire, la tâche choisie doit être déployée et allouée sur un PU en utilisant une «*Configuration*» à part. Dans notre exemple, l'unité de traitement PU2 peut configurer la taille de l'image reçue (effectuer la tâche Resolution) en supportant l'implémentation High, Medium et Low qui correspondent respectivement aux modes *High*, *medium* et *low* définis pour chaque Configuration. Le tableau 7.1 traduit le contenu de la figure 7.7 afin d'exprimer l'allocation des différentes implémentations des tâches sur les PUs, selon les configurations définies.

TABLE 7.1: Allocation des différentes implémentations sur les PUs selon les configurations définies

	Type d'implémentation à allouer		
Unité de traitement	**Configuration_1**	**Configuration_2**	**Configuration_3**
PU2	High_Resolution	Medium_Resolution	Low_Resolution
PU3	Color _Filter	BW_Filter	—

7.3.6 Autres diagrammes pour la modélisation de l'architecture déployée

Dans le but de modéliser une plateforme d'un système SoC complet basé sur FPGA (et par conséquent, les bas niveaux du flot de génération), les instances des composants IPs doivent être représentées dans ce modèle. Cependant, afin d'éviter l'encombrement des informations, nous avons décidé de séparer les diagrammes en plusieurs vues. Nous présentons dans ce qui suit quatre diagrammes modélisés en MARTE afin de faciliter les transformations de modèles.

7.3.6.1 Modélisation des interfaces

La figure 7.8 présente le modèle d'architecture déployée contenant les blocs d'IPs ainsi que leurs interconnexions via un sous-système de bus. Ainsi, dans ce diagramme, seules les interfaces de bus sont représentées. Les ports sont différenciés grâce au stéréotype étendu «*ExtendedFlowPort*» de RecoMARTE qui permet de classifier les ports en trois groupes : interfaces basées sur les bus (BusInterface), port simple (singlePorts) et les ports d'entrée/sortie (IO_interface). Le premier et le dernier types sont représentés dans ce diagramme : Les interfaces basées sur les bus associées à des protocoles de bus prédéfinis, tandis que les IP_interfaces représentent les groupes de signaux non standards associés à des interfaces d'entrée/sortie, externes à l'FPGA.

7.3.6.2 Modélisation des ports Reset et d'interruption

Ce diagramme (Cf. Figure 7.9) représente le même modèle d'architecture déployée en soulignant un autre type de ports : les ports Reset et les ports d'interruption. Ces ports sont reliés, via des interconnexions point à point, aux instances des composants qui leurs correspondent.

Ce type d'informations existe dans le fichier de configuration MHS. L'objectif de les faire ressortir et de les exposer dans ce modèle, est de faciliter l'éventuelle phase de transformation de IP-XACT vers MHS.

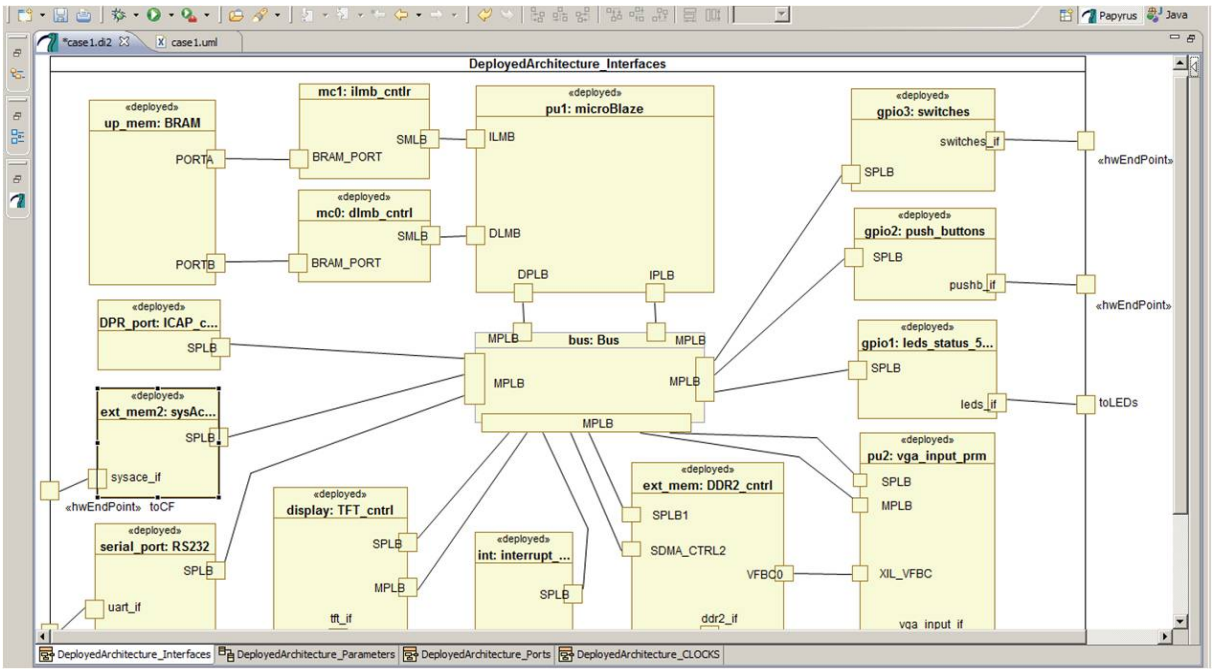


FIGURE 7.8: Modélisation de l'architecture déployée avec les interfaces

7.3.6.3 *Modélisation des ports d'horloge*

Les signaux d'horloge ont été modélisés dans un autre diagramme (Cf. Figure 7.10), mais toujours relié aux composants de l'architecture logique. Il est important de noter que le concepteur, à un haut niveau d'abstraction, n'a pas besoin de traiter ce genre de détails de bas niveau. Cependant, ces diagrammes peuvent être confiés à un expert dans la conception des FPGAs. Autrement, ils seront obtenus suite à une transformation de modèles, à partir d'une description précédemment créée d'un design IP-XACT réutilisable, favorisant ainsi la réutilisabilité de la conception et l'abstraction des détails du bas niveau.

Toutefois, la présence de cette information est nécessaire dans les modèles MARTE afin de pouvoir générer une description complète de la plateforme et éviter d'éventuels problèmes de synchronisation.

7.3.7 Modélisation des classes d'IPs

Il s'agit d'une représentation (Parametrization View) des classes d'IPs afin de les configurer et les paramétrer, en particulier ceux qui vont exécuter les tâches de l'application (autrement dits les modules reconfigurables). Le concepteur a la possibilité de paramétrer ces IPs, autrement, il peut faire usage d'une procédure d'Exploration de l'espace de conception (DSE). Cette vue représente la relation entre les blocs d'IPs avec leurs identifiants <*vlnv*> de IP-XACT.

La figure 7.11 présente un diagramme de classe illustrant tous les IPs utilisés dans notre étude de cas. Les détails des paramètres sont invisibles dans ce diagramme afin de rendre la figure plus lisible. Ce diagramme contient tous les IPs correspondant aux composants de l'architecture ainsi que les tâches de l'application (les différentes implémentations des tâches

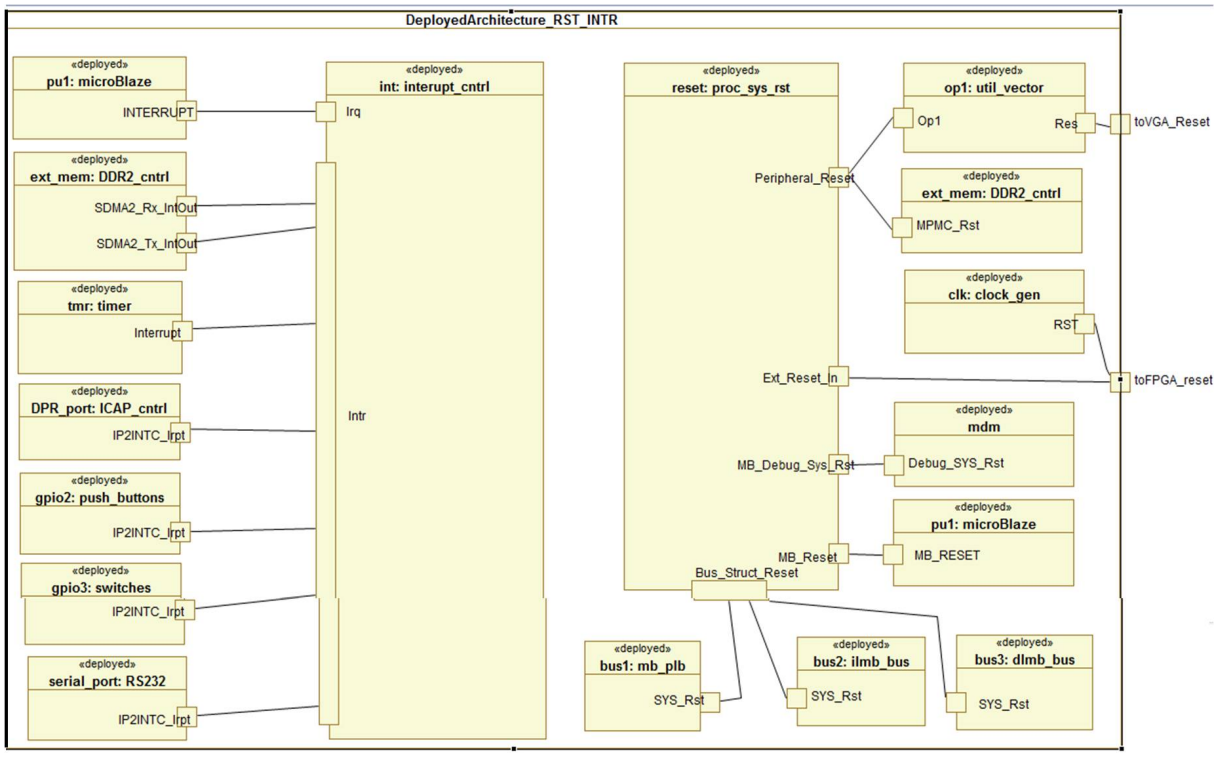


FIGURE 7.9: Modélisation de l'architecture déployée avec les ports Reset et les ports d'interruption

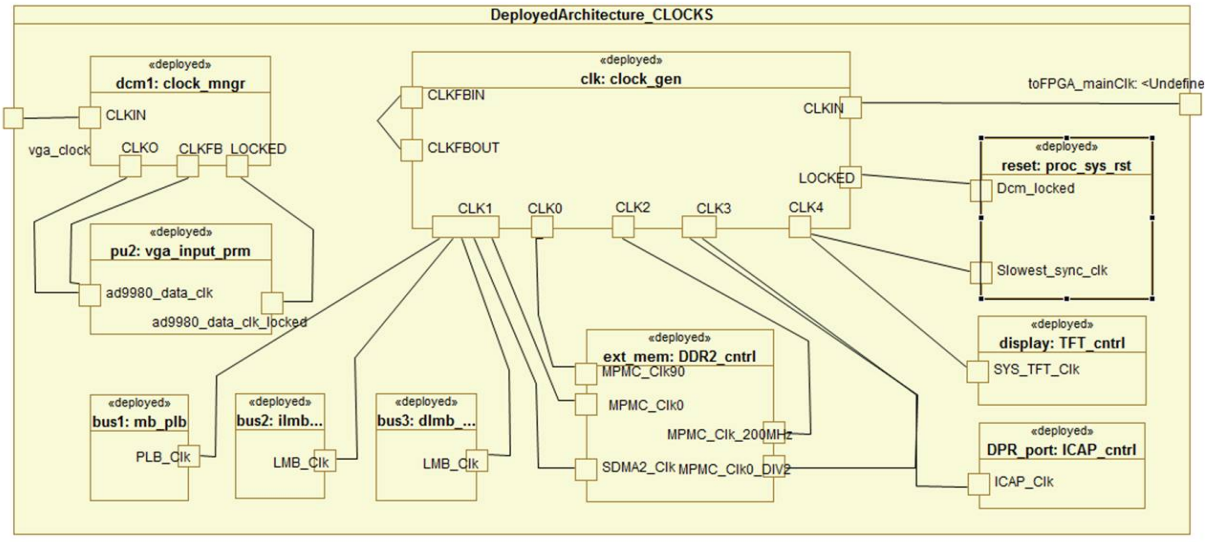


FIGURE 7.10: Modélisation de l'architecture déployée avec les ports d'horloge

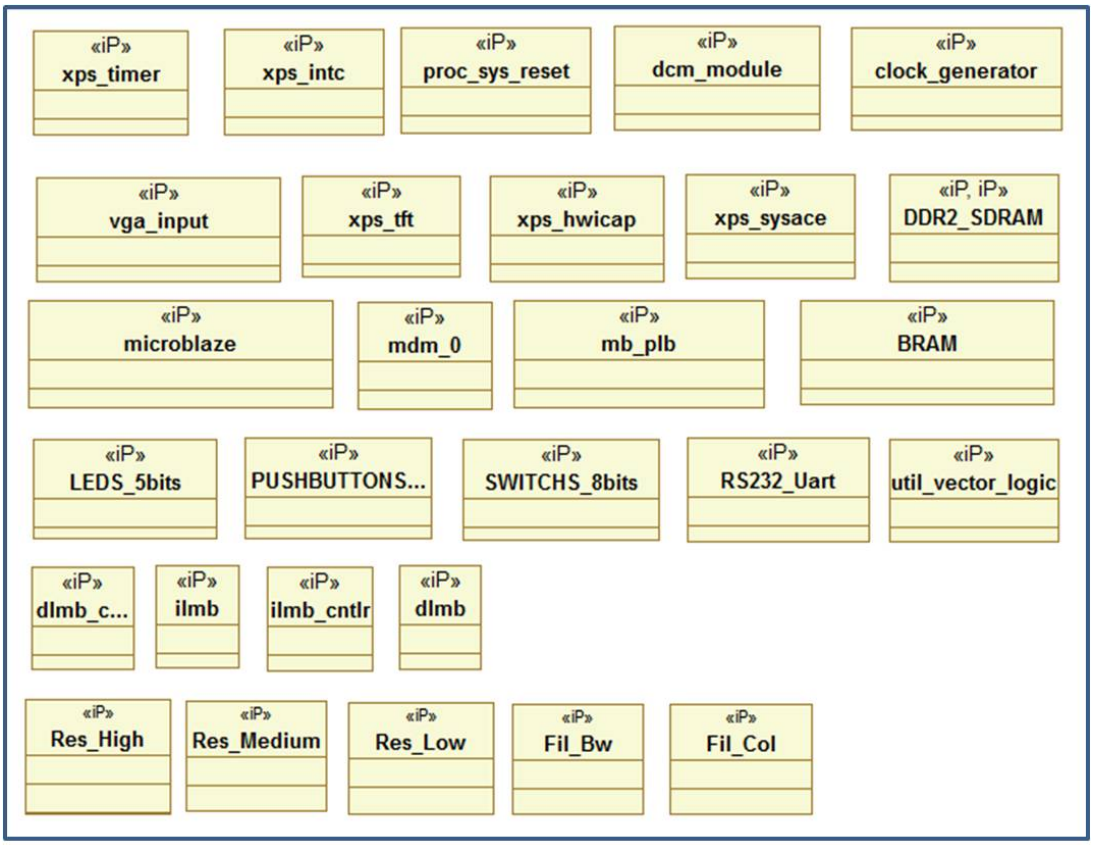


FIGURE 7.11: Modélisation des classes d'IPs paramétrés

reconfigurables).

Un extrait d'un IP paramétré est exposé dans la figure 7.12. L'IP présenté en exemple est un UART dont les propriétés sont modélisées avec l'outil Papyrus. Une autre représentation des IPs est illustrée par la figure 7.13. Ici, les propriétés (paramètres) des IPs sont visibles en forme de commentaire. De plus, dans cet exemple, nous montrons la relation entre le composant déployé et son IP qui lui est associé. Ces informations sont utilisées dans la génération de la description de conception IP-XACT dans laquelle elles sont transformées sous forme de <*ConfigurableElements*> sous chaque <*ComponentInstance*>, comme précédemment expliqué dans le chapitre des transformations RecoMARTE vers IP-XACT.

L'un des rôles les plus importants de cette vue de paramétrage, est de contrôler les attributs de configuration qui permettent l'intégration de paramètres dépendants, des ports et des interfaces dans le fichier MHS sous-jacent ; et par conséquent, l'intégration dans la description HDL du haut niveau de la plateforme statique.

Par ailleurs, certains composants de la plateforme statique doivent être configurés (paramétrés) tandis que d'autres sont considérés comme des boites noires matérielles telles que le composant ICAP. De plus, chaque IP est associé à un CodeFile qui est de type Artifact. Un extrait des artifacts est illustré par la figure 7.14. Chaque codeFile doit avoir un chemin unique qui l'identifie grâce à la valeur de l'attribut <*filePath*>, par contre l'attribut <*entityName*> doit

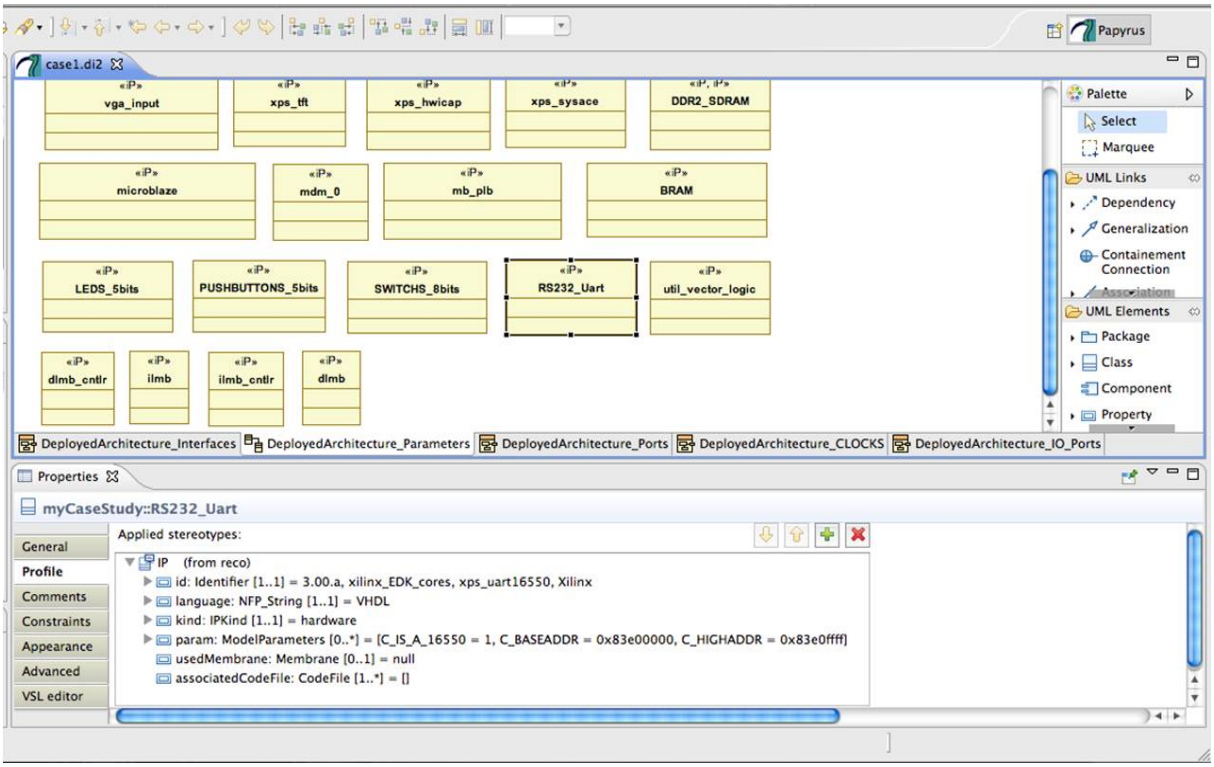
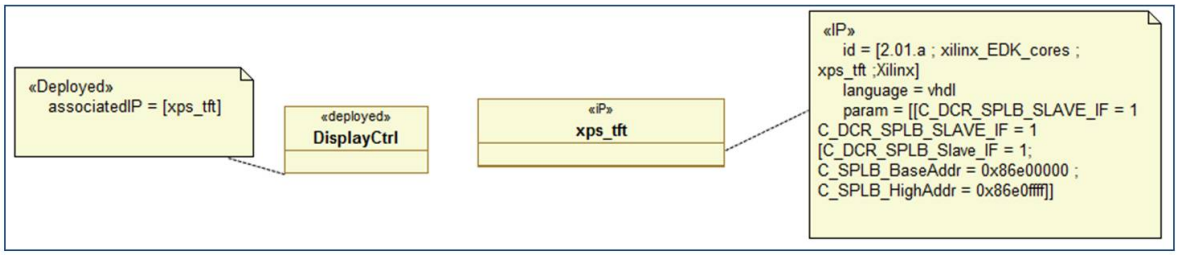
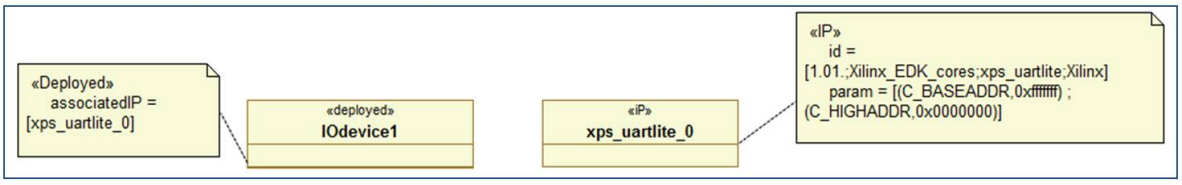


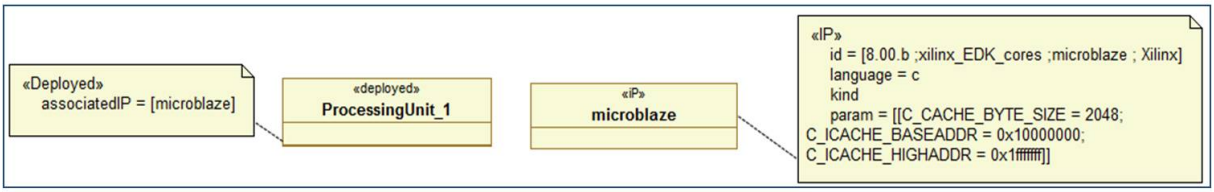
FIGURE 7.12: Exemple de paramètres d'un IP avec Papyrus



a) Configuration de l'IP XPS_TFT



b) Configuration de l'IP XPS_UART



c) Configuration de l'IP du MicroBlaze

FIGURE 7.13: Extrait de la modélisation de la relation composant déployé/Classe d'IP paramétrée

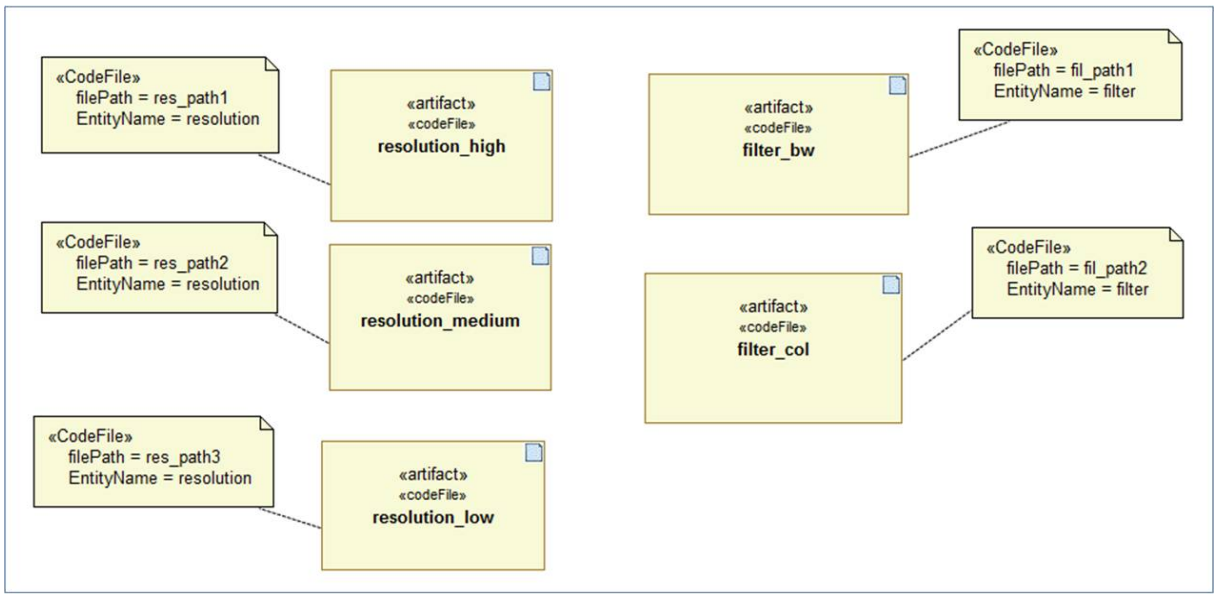


FIGURE 7.14: Extrait de la modélisation des CodeFiles correspondant aux IPs de l'application

être le même pour les différentes implémentations de la même tâche. Dans l'exemple, la tâche *Resolution* est une tâche implémentée par un IP matériel écrit en VHDL. Il faut donc :

1. définir les 3 Codefiles correspondant au nombre d'implémentations à savoir *resolution_High*, *resolution_Medium* et *resolution_Low* ;
2. Tous ces codeFiles possèdent la même valeur de l'attribut {<*entityName*> = *resolution*} puisqu'il s'agit du même IP, en d'autres termes la même boite noire avec la même interface
3. Chaque Codefile est identifié par un chemin unique spécifié par l'attribut *filePath*.

Ainsi, la figure 7.13 montre trois exemples d'IPs qui correspondent respectivement aux composants appartement à l'architecture logique de notre étude de cas de traitement d'image. Nos retrouvons le composant TFT (figure 7.13.a) qui correspond au composant déployé DisplayCtrl de l'architecture logique, le UART (figure 7.13.b) associé au périphérique d'entrée/sortie et enfin le processeur softcore Microblaze (figure 7.13.c) associé à l'unité de traitement PU1.

Par ailleurs, les tâches de l'application *Resolution* et *filter* sont implémentées par des IPs matériels. Au niveau du déploiement, ces IPs sont associés aux unités de calcul (les PUs) déployées. Notons que nous avons défini précédemment dans la figure 7.7, les différentes configurations de l'allocation des tâches sur les PUs. Il est maintenant nécessaire d'associer, pour chaque configuration, qui (PU) va exécuter quoi (IP). La figure 7.15 nous montre un extrait des différentes associations PUs/IPs pour les configurations possibles. Par exemple, dans la configuration en {mode=high}, la tâche Resolution_high a été allouée précédemment au composant PU2, au niveau de l'allocation. En d'autres termes, le PU2 va exécuter l'implémentation *high* de la tâche résolution et devrait donc être associé à l'IP *Res_high*. Ceci définit une première configuration pour la première implémentation. Idem pour les implémntations *medium* et *low*, ainsi que pour la tâche *Filter*.

Enfin, dans ce diagramme, nous associons aussi les CodeFiles de chaque IP. Ces CodeFiles

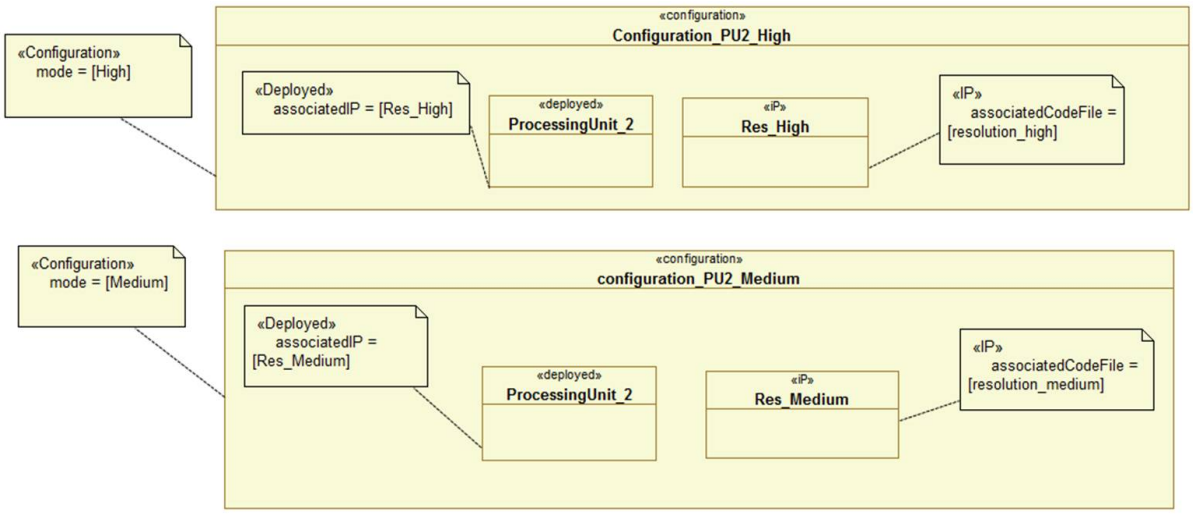


FIGURE 7.15: Extrait de la modélisation des associations PUs/IP

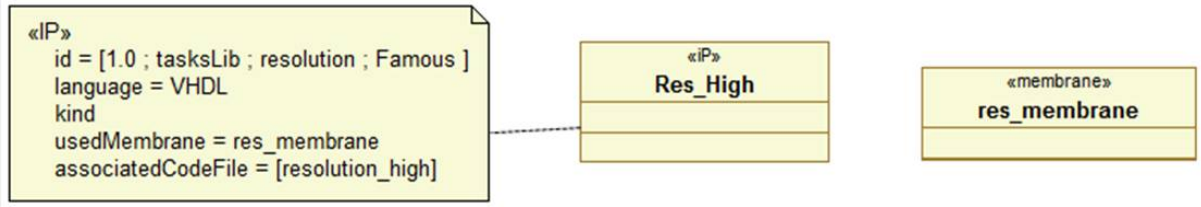


FIGURE 7.16: Exemple d'association d'un IP à sa membrane

ont été précédemment définis dans la figure 7.14.

Notons que la séparation des différents niveaux d'abstraction permet de rendre les représentations plus claires et les diagrammes nettement plus lisibles et compréhensibles par le concepteur. Ceci facilite bien évidement sa tâche est améliore sa productivité.

7.3.8 Modélisation des membranes

Le module du mécanisme de la diffusion de reconfiguration prévu dans FAMOUS n'est encore pas mis en œuvre lors de la mis en place du démonstrateur. Cependant, nous présentons ici un exemple de modélisation de l'association d'un IP à une membrane suivant les concepts définis dans le profil RecoMARTE. Dans cet exemple (Cf. Figure 7.16), l'IP *Res_High* est associé au PU2 qui sera placé sur une région reconfigurable. Il va donc utiliser une membrane qui va sauvegarder et gérer son contexte dans le cas d'une reconfiguration. Après avoir instancié la membrane, il faut donc saisir la valeur de la relation "*usedMembrane*" en spécifiant le nom de la membrane correspondante.

7.3.9 Le modèle physique

7.3.9.1 Modélisation de l'architecture physique de l'FPGA

Les concepts permettant de modéliser la plateforme physique de l'FPGA ont été étendues et présentées dans le chapitre 5. Ces concepts offrent au concepteur de définir les zones reconfi-

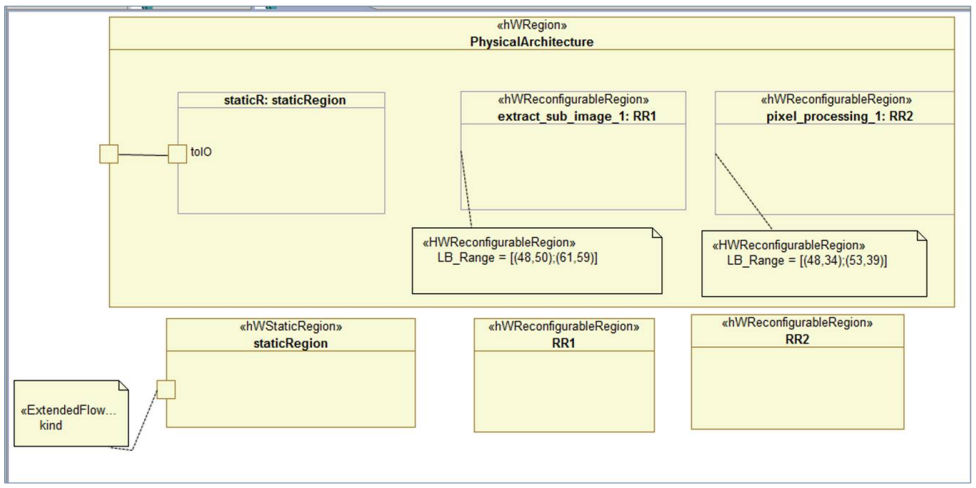


FIGURE 7.17: Modélisation de l'architecture physique de la plateforme

gurables qui vont contenir les unités de traitement (co-processeurs ou accélérateurs) exécutant les tâches reconfigurables de l'application. Dans notre étude de cas, nous disposons de deux tâches reconfigurables *Resolution* et *Filter*, qui vont être exécutées respectivement par les pu2 et pu3. Par conséquent, ces deux PUs devront être placés (alloués) sur deux zones reconfigurables différentes.

Ainsi, le modèle de l'architecture physique (Cf. figure 7.17) contiendra une région statique et deux régions reconfigurables. La région statique est stéréotypée «*HwStaticRegion*», elle contiendra tous les composants statiques de l'architecture (dans notre exemple : microblaze, les périphériques d'entrée/sortie, le bus, l'ICAP, etc.). Tandis que les zones reconfigurables sont stéréotypées «*HwReconfigurableRegion*» et leurs coordonnées sont données dans notre exemple. De la même manière que le concepteur définisse les composants qu'il placera dans la région statique, il identifie également les caractéristiques des régions reconfigurables. Rappelons que ces coordonnées sont définies en terme de certaines ressources physiques : Les blocs logiques (LB), les blocs mémoires (BRAM) et enfin les DSPs. Ces blocs sont agencés sur la plateforme de l'FPGA sous forme de colonnes ; leur nombre et leur disposition varie selon la technologie de l'FPGA choisie.

Dans notre cas d'étude, l'application de traitement d'image a été réalisée sur la carte ML605. Le modèle de la plateforme physique est présenté par la figure 7.17. Nous définissons ainsi deux régions reconfigurables dont les coordonnées sont définies selon des <Ranges> de LBs.

Il est à noter que la région statique est connectée à l'extérieur de l'FPGA, via des ports stéréotypés «*ExtendedFlowPort*» dont la valeur de l'attribut kind= IO_Interface.

7.3.9.2 Allocation physique de l'architecture logique déployée sur l'architecture physique

Pour résumer (Cf. Figure 7.18), tous les composants statiques du modèle déployé de l'architecture logique sont alloués (placés) à la région statique de la plateforme physique. Tandis que les composants (dans notre cas de figure : pu2 et pu3 qui exécutent les tâches *Resolution* et *filter*), sont implémentés en tant qu'accélérateurs matériels (des modules VHDL) et sont placés

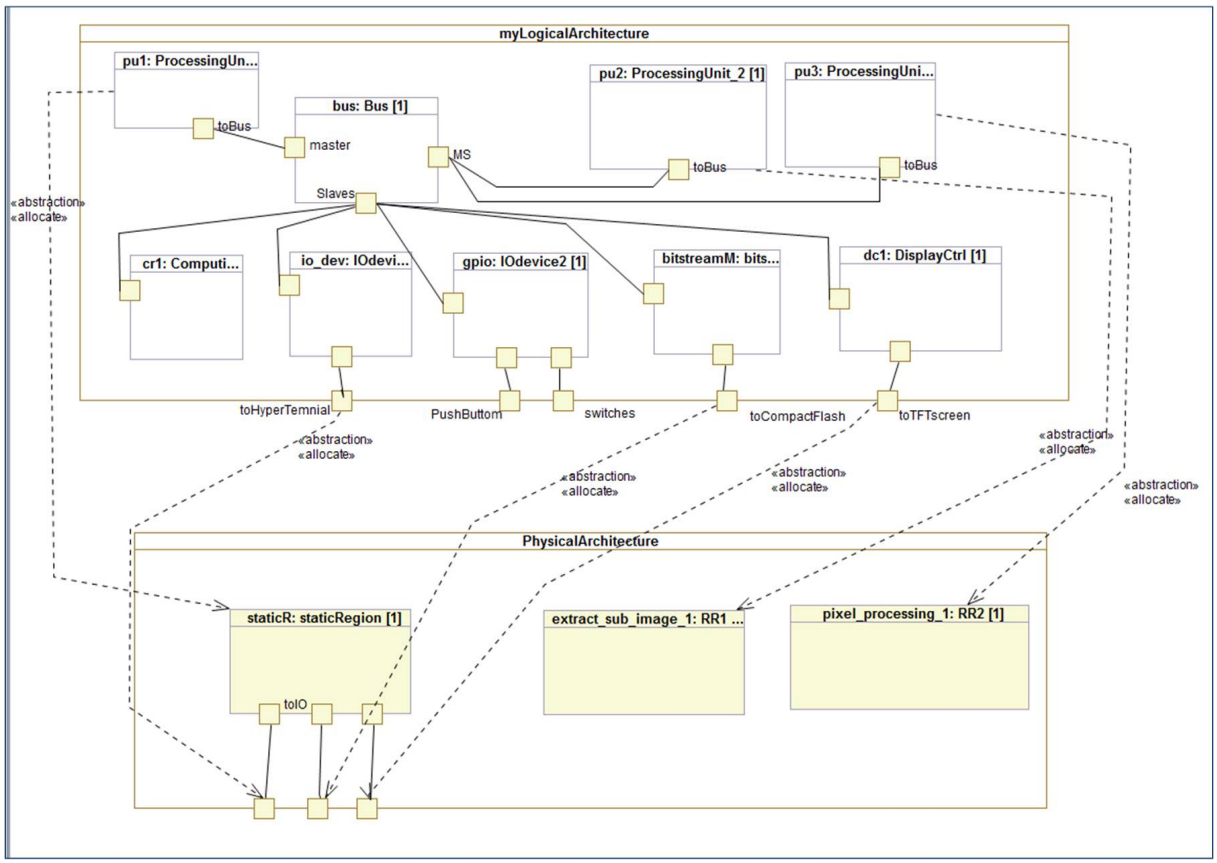


FIGURE 7.18: Modélisation de l'allocation physique

respectivement sur les régions reconfigurables *extract_sub_image* (RR1) et *pixel_processing* (RR2).

De plus, les ports, précédemment définis dans le diagramme de l'architecture logique, stéréotypés «*HwEndPoint*» assurent l'interconnexion *logique* de certains composants avec les périphériques d'entrée/sortie (tels que le GPIO ou UART). Ces ports sont donc alloués aux ports physiques du modèle de l'architecture physique de l'FPGA, appelés IOB (Input/Output Blocks dans le jargon Xilinx).

7.3.9.3 Le modèle final

Le modèle final est obtenu suite à la phase de l'allocation et représente ainsi le résultat de notre flot de conception en double Y, précédemment défini dans le chapitre 4. Le tableau 7.2 traduit le modèle final des deux types d'allocation :

- Allocation *logique* du modèle d'application déployé sur le modèle d'architecture déployé : il s'agit d'associer (allouer) les différentes implémentations des tâches reconfigurables aux unités de traitement (appelés Processing Unit PU) qui se chargeront de les exécuter. Cette allocation respecte, bien évidemment, un certain nombre de configurations valables et définies par le concepteur. Ainsi, ce niveau de modélisation présente le résultat de la première partie de notre flot en double Y.
- Allocation *physique* de l'architecture logique déployée sur le modèle physique de la plate-forme d'FPGA : il s'agit du résultat de la deuxième partie de notre flot.

TABLE 7.2: Illustration du modèle final d'Allocation des différentes implémentations sur les PUs placés sur les régions reconfigurables

Régions	Unité de traitement	Type d'implémentation		
		Configuration1	Configuration2	Configuration3
extract_sub_image (RR1)	PU2	High_Resolution	Medium_Resolution	Low_Resolution
pixel_processing (RR2)	PU3	Color _Filter	BW_Filter	—

7.3.10 Modélisation du contrôle de la reconfiguration

La reconfiguration dynamique est la caractéristique clé dans la conception et l'implémentation des FPGAs, qui doivent être en mesure de faire face à l'environnement de l'utilisateur final ainsi que ses exigences.

Dans un système, la notion de configuration représente une combinaison de modes actifs de ce système à un instant donné. Ainsi, intervient le contrôleur qui choisit quelle configuration charger, en leur attribuant des priorités. Il décide également de la suspension ou la reprise des tâches en cours d'exécution.

Étant donné que notre application dispose de deux tâches reconfigurables, il est donc nécessaire de définir un contrôleur qui va gérer le comportement de ces tâches. Ainsi, nous définissons le contrôleur, nommé dans cet exemple *myController* et stéréotypé «*Controller*», illustré par la figure 7.19. nous rappelons que les concepts de contrôle de la reconfiguration ont été étendus dans le profil RecoMARTE et présentés dans la section 5.6 du chapitre 5. MyController dispose d'un tableau de type *ModeBehavior* afin de contrôler les ModeBehaviors respectifs *ResolutionModes* et *FilterModes*.

Il est à noter que le rôle principal du contrôleur est de garantir que la configuration globale du système respecte un ensemble de contraintes globales. Ainsi, nous définissons un fichier de contraintes grâce l'attribut *contractFile*. Les contraintes à respecter sont décrites sous la forme de contrainte «*nfpConstraint*».

Nous rappelons également qu'un acteur du projet FAMOUS a traité avec plus de détails l'aspect du contrôle de la reconfiguration dynamique [35], en faisant usage à la technique de synthèse du contrôleur. Ces travaux ont été précédemment présentés dans le chapitre 5. Le contrôleur modélisé est généré en langage C et sera placé sur le processeur MicroBlaze de la plateforme du démonstrateur. Par conséquent, ce placement n'est pas effectué au niveau des transformations, mais c'est l'outil de génération de code qui s'en charge.

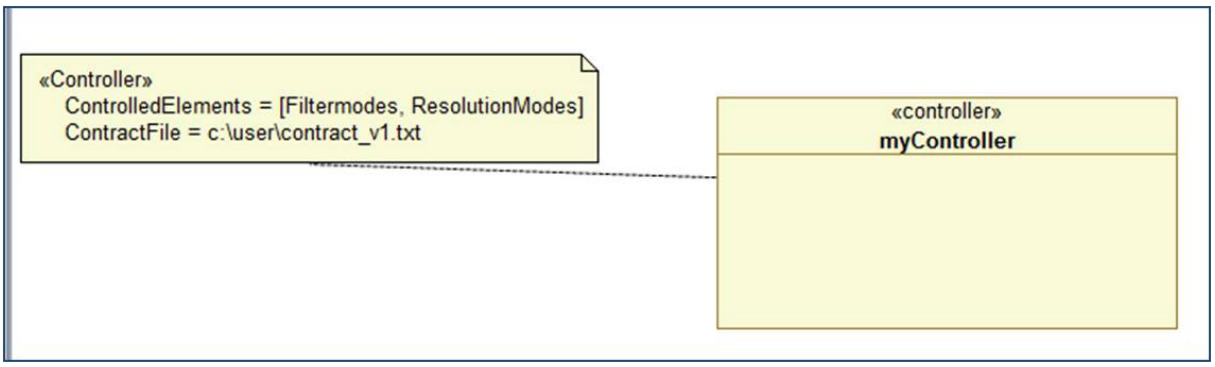


FIGURE 7.19: Modélisation du contrôleur de la reconfiguration

7.4 Implémentation et génération

Les transformations de modèles ont été présentées dans le chapitre 6. Ces transformations nous ont permis de passer d'un haut niveau de modélisation en MARTE et RecoMARTE vers une description intermédiaire basée sur le standard IP-XACT. Ensuite, à partir d'une transformation du modèle IP-XACT vers une spécification de la plateforme matérielle Xilinx XPS, le fichier MHS ainsi obtenu est exécuté et utilisé par l'outil MDWorkbench. Suivi d'un ensemble de scripts à exécuter, ceci a permis de générer un système pouvant être utilisé par l'environnement EDK de Xilinx comme le montre la figure 7.20. Cette figure illustre une capture d'écran de la description du système XPS obtenue (il s'agit du *block diagram* de l'architecture utilisée pour notre cas d'étude).

Suivant toute autre approche de conception, cette plateforme matérielle peut être utilisée afin de tester des fonctionnalités d'un système qui n'est pas reconfigurable dynamiquement. Ainsi, les fonctionnalités destinées à la RDP peuvent être créées en premier, en ajoutant ensuite des *wrappers* RDP qui permettront de générer le système avec des partitions reconfigurables. Ces dernières peuvent être synthétisées par la suite, sans être ni placées et ni routés, selon le flot classique de conception d'FPGA.

Enfin, une fois le système statique (c-à-d non reconfigurable dynamiquement) ait été validé, l'étape de la synthèse peut avoir lieu afin de générer les fichiers Netlists. Il s'agit des fichiers qui seront donnés en entrée à l'outil PlanAhead pour l'implémentation du système RDP. La totalité des fichiers bitstreams partiels est par la suite obtenue afin de pouvoir configurer l'FPGA.

La figure 7.21 est une capture d'écran du projet du démonstrateur via l'outil PlanAhead. Elle illustre l'étape de placement physique (*flooplanning*) du système.

Les tableaux 7.3 et 7.4 résument respectivement les langages et les outils utilisés lors du développement du démonstrateur du projet FAMOUS. Certains outils et langages ont été utilisés au niveau de la modélisation haut-niveau, d'autres au niveau implémentation.

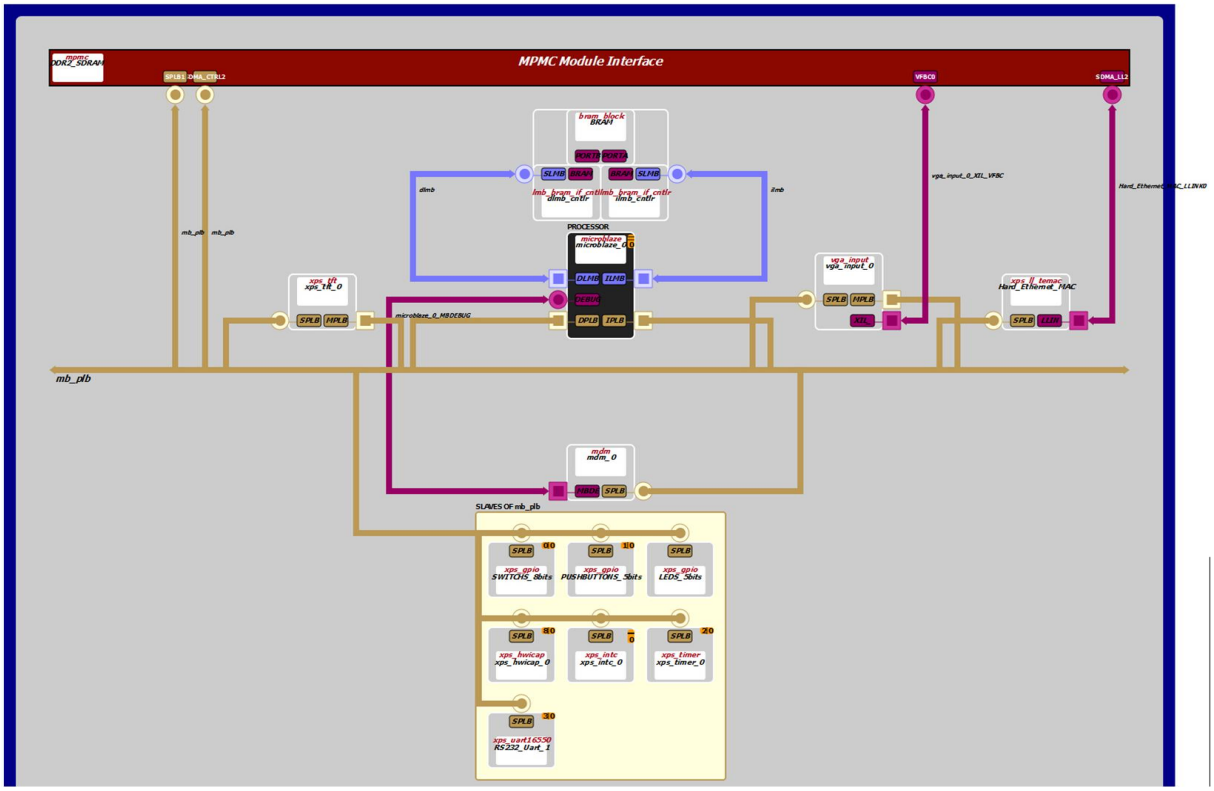


FIGURE 7.20: Extrait de la description obtenue dans l'outil Xilinx XPS

TABLE 7.3: Langages utilisés dans le démonstrateur FAMOUS

Langages utilisés	Utilité et objectifs
MQL / QVTO	Définition des règles de transformations
IP-XACT	Modèle intermédiaire
VHDL	Implémentation des IPs

TABLE 7.4: outils utilisés pour le démonstrateur FAMOUS

Outils utilisés	Utilité et objectifs
Papyrus	Modélisation haut-niveau
Rhapsody	Méta-modèles EDK
MDWorkbench	Transformations de modèles
Xilinx EDK	Implémentation de la RDP dans les SoCs
Xilinx ISE	Développement des IPs

7.5 Conclusion

Ce chapitre vient clore la partie "validation" de cette étude en montrant l'utilisation de notre méthodologie basée sur un flot de conception dans le cadre d'une application de traitement d'images, développée dans le projet FAMOUS. Cette étude de cas nous a permis de mettre en œuvre la modélisation haut-niveau en utilisant les profils MARTE et RecoMARTE pour la

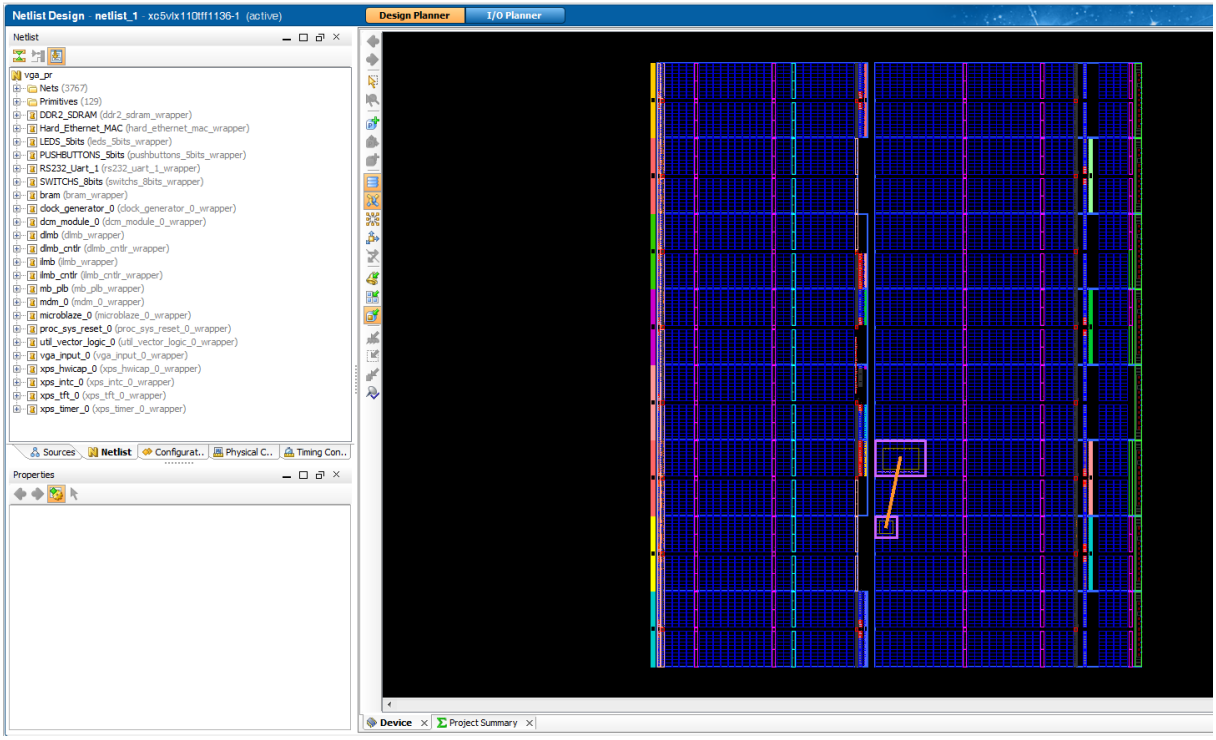


FIGURE 7.21: La phase du flooplanning (placement) de notre étude de cas

spécification des concepts de la reconfiguration dynamique. Nous avons également présenté les différents niveaux de modélisation suivant le flot jusqu'à la phase d'implémentation et génération. L'automatisation du flot grâce à la spécification des règles de transformations a été présentée dans le chapitre 6 ce qui nous a permis de passer d'une modélisation à haut niveau à une description d'implémentation. La figure 7.22 illustre la dernière contribution de la partie "Validation et Automatisation" à savoir la modélisation de l'étude de cas et du démonstrateur depuis le haut niveau jusqu'à la génération du système.

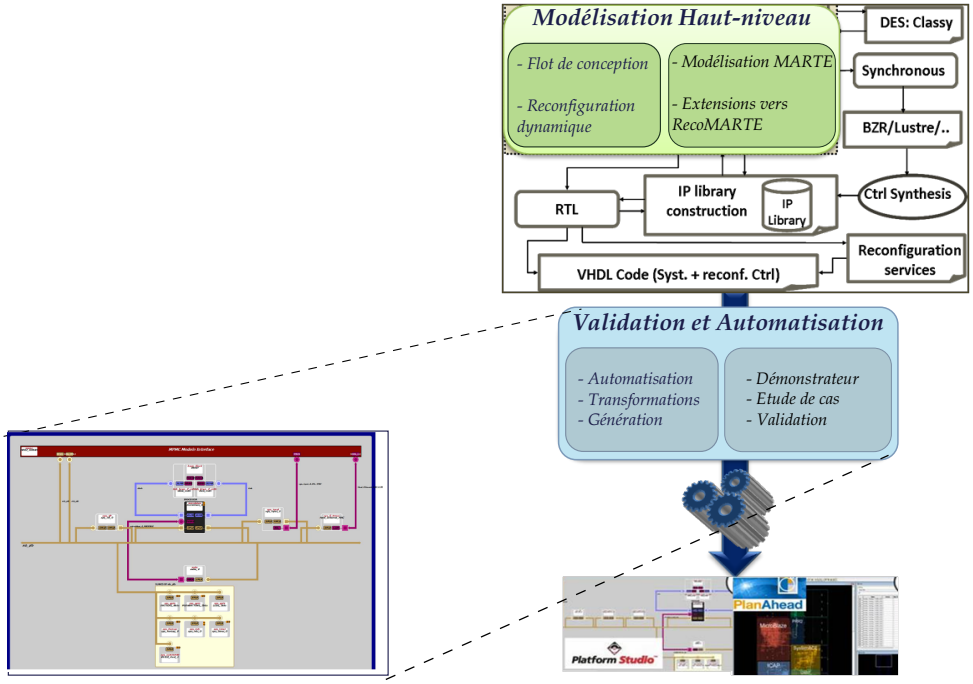


FIGURE 7.22: Contribution 4 : Etude de cas et validation du démonstrateur

CHAPITRE 8

BILAN ET PERSPECTIVES

8.1 Bilan

Les travaux présentés dans ce manuscrit visent à proposer une méthodologie de conception conjointe des systèmes dynamiquement reconfigurables basés sur FPGA. Notre méthodologie s'appuie sur des standards OMG : l'Ingénierie Dirigée par les Modèles (IDM) et le profil UML MARTE. L'objectif de nos travaux visait à garantir la flexibilité, la réutilisabilité et l'automatisation afin de faciliter le travail du concepteur et d'améliorer sa productivité.

Cette thèse s'inscrit dans le cadre du projet ANR FAMOUS. Il s'agit ici d'apporter essentiellement une contribution au premier axe de recherche du projet concernant la modélisation haut-niveau de la reconfiguration dynamique des FPGAs suivant un flot de conception. De nos travaux de thèse, il résulte la proposition d'une chaîne de conception automatisée et entièrement dirigées par les modèles. Cette chaîne permet la génération d'une description complète du système conçu utilisée comme entrée au flot de conception de la RDP (Xilinx) et ce, à partir d'une modélisation à haut niveau (MARTE).

La figure 8.1 illustre nos contributions (les cadres en vert et en bleu) qui sont intégrées dans le flot global du projet FAMOUS. Dans ce qui suit, nous passons en revue les principaux points qui les résument.

8.1.1 Flot de conception haut niveau

Avec la croissante complexité des systèmes sur puce (en particulier ceux implantés sur les FPGAs), il est devenu impossible de les concevoir dans un bas niveau (RTL : *Register Transfert Level*) où il faut préciser chaque détail du comportement des composants. Afin de dépasser ce défi, nous avons mis en œuvre une méthodologie de conception basée sur les modèles pour modéliser les systèmes dynamiquement reconfigurables à un haut niveau d'abstraction, permettant ainsi de cacher un grand nombre de détails d'implémentation.

Cette méthodologie suit un flot de conception conjointe basé sur l'IDM afin d'assurer l'automatisation de la génération de code. Ce flot, comme précédemment détaillé au cours de ce

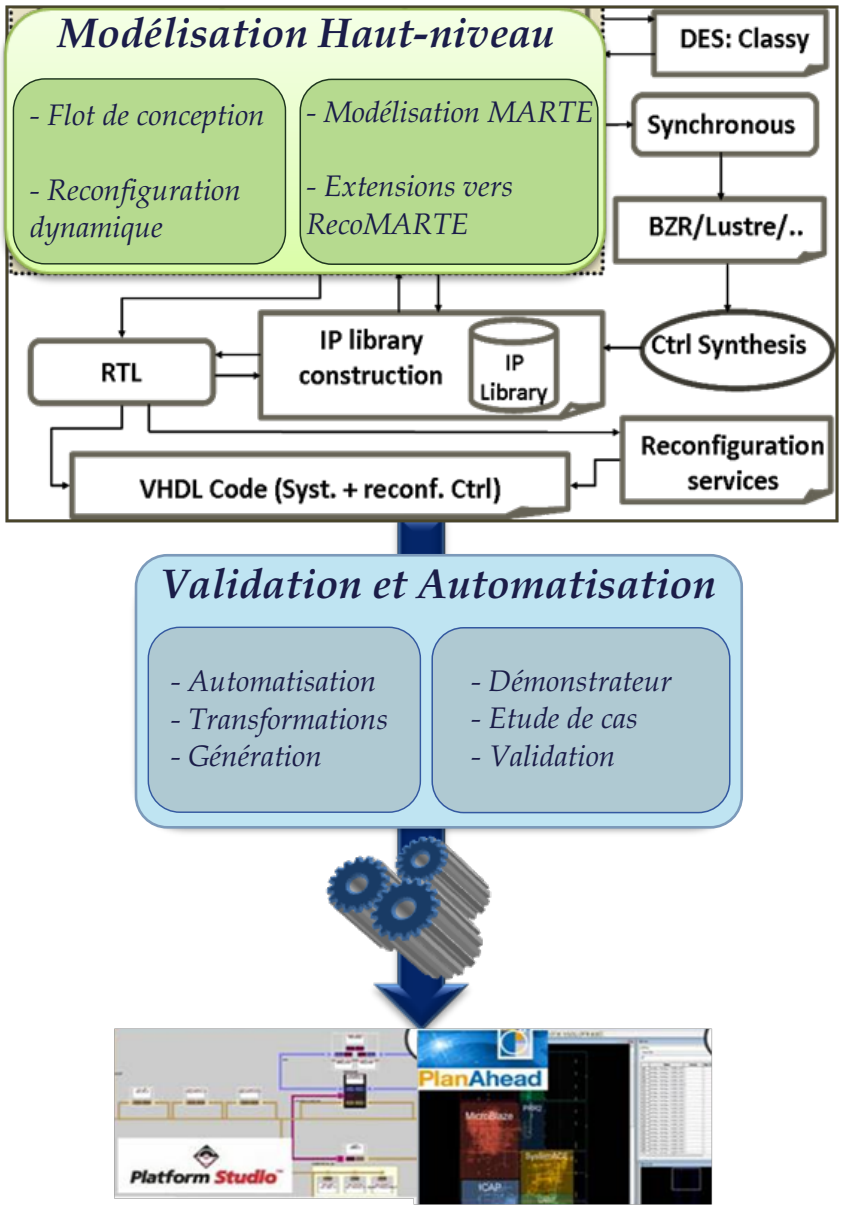


FIGURE 8.1: Illustration des différentes contributions de la thèse

présent document, est partiellement basé sur le modèle en Y (Y-chart). En effet, il ressemble à un *double Y* dont les branches correspondent aux différents niveaux de modélisation. En effet, les premières branches du Y représentent la modélisation conjointe de l'application et de l'architecture du système conçu ainsi que leur association que nous avons nommée <*allocation déployée*>. Nous rappelons que ce niveau fusionne l'allocation simple de MARTE et le niveau de déploiement de UML afin de faciliter d'une part, le paramétrage et la configuration des IPs, et d'autre part leur ré-utilisation.

Les branches du deuxième Y concernent la modélisation ciblant les plateformes FPGAs. En effet, le modèle déployé est mappé sur une plateforme physique de l'FPGA. Ainsi, la modélisation de l'architecture physique est nécessaire à ce stade afin de se rapprocher au mieux des technologies.

Étant donné que nous ciblons des FPGAs dynamiquement reconfigurables, il est inéluctable

de définir le mécanisme du contrôle de la RD. En effet, ces modèles identifient l'exécution des tâches concurrentes de l'application, disposant de plusieurs implémentations. Ces implémentations sont allouées par la suite soit, en tant que tâches matérielles (accélérateur matériel) sur un FPGA, soit en tant que tâches logicielles sur un processeur.

8.1.2 Extension du profil/méta-modèle MARTE : vers RecoMARTE

Suivant le flot de conception précité, plusieurs modèles sont présentés, moyennant principalement les concepts du le profil UML MARTE. Cependant, la modélisation de certains concepts de la reconfiguration dynamique des FPGAs a nécessité des extensions dans MARTE. Nous avons donc identifié les concepts manquants et les avons intégrés dans un nouveau profil qui étend MARTE et baptisé RecoMARTE (Reconfigurable MARTE). Il vient répondre également aux besoins des différents acteurs du projet FAMOUS, en intégrant les concepts permettant 1) la modélisation d'une application à caractère reconfigurable en prenant en compte son comportement 2) la modélisation du contrôle de la reconfiguration dynamique qui permet de gérer ces tâches 3) la modélisation d'un niveau de déploiement, intermédiaire entre une description haut niveau MARTE et les outils de la RDP de Xilinx 4) la modélisation de membrane qui permet la sauvegarde de contexte des modules reconfigurables et enfin 5) la modélisation de la plateforme physique de l'FPGA qui permet de se rapprocher le plus des détails d'implémentation.

Par conséquent, le profil a été étendu ainsi que le méta-modèle de MARTE afin d'aider les concepteurs à représenter les propositions mentionnées précédemment via des outils de modélisation graphique UML tels que Papyrus.

8.1.3 Automatisation de la chaîne : Spécification des transformations de modèles

Afin d'intégrer notre flot de conception et d'assurer son automatisation vers la génération du code, une chaîne de transformation a été utilisée. Le modèle final en MARTE résultant du flot de conception proposé est donné comme entrée à cette chaîne.

Nous passons ainsi d'un modèle MARTE (RecoMARTE) vers une description intermédiaire selon le standard IP-XACT afin de générer finalement des fichiers décrivant le système complet dans l'environnement XPS de Xilinx. Ces fichiers sont MHS, MPD et UCF.

Le choix d'utiliser la description IP-XACT, notamment dans nos travaux, est justifié par le fait que IP-XACT fournit une représentation intermédiaire à partir de nos modèles haut niveau MARTE. Il s'agit d'un méta-modèle sur lequel plusieurs outils peuvent être développés, d'où l'intérêt d'unifier MARTE et IP-XACT. Ceci permet à la fin de visualiser des IPs dans les modèles MARTE afin de composer un diagramme structurel qui se transformera par la suite en un design IP-XACT.

L'utilisation de l'approche IDM nous a permis de rendre tous les les détails d'implémentation de bas niveau transparents aux concepteurs et d'automatiser ainsi notre flot de conception haut niveau vers la génération du code. De plus, cette automatisation va permettre d'accélérer la phase de conception et éviter les erreurs dues à la manipulation directe des ces détails.

8.1.4 Validation expérimentale

L'objectif majeur de cette étude est de pouvoir mettre en œuvre un exemple de système reconfigurable, dont la conception va exploiter la méthodologie présentée dans ce manuscrit. Pour ce faire, un démonstrateur a été mis en place afin de permettre aux membres du projet de valider l'ensemble des contributions apportées. L'application développée est une application de traitement d'images ayant des tâches reconfigurables et a été réalisée sur une carte ML-605, basée sur un FPGA Virtex 6.

Ainsi, nous avons présenté la modélisation de l'application proposée afin de valider notre méthodologie. Ces modèles ont suivi les différents niveaux de notre flot (double Y) proposé précédemment et ont fait usage des deux profils MARTE et RecoMARTE pour la modélisation des concepts de la RDP.

8.2 Perspectives

La méthodologie proposée aborde plusieurs aspects de la modélisation des systèmes dynamiquement reconfigurables basés sur FPGAs. Ainsi, pour chacun d'eux, de nouvelles questions ont été soulevées au cours du développement de cette thèse. Nous pouvons en énumérer quelques unes :

8.2.1 Modélisation du parallélisme de données

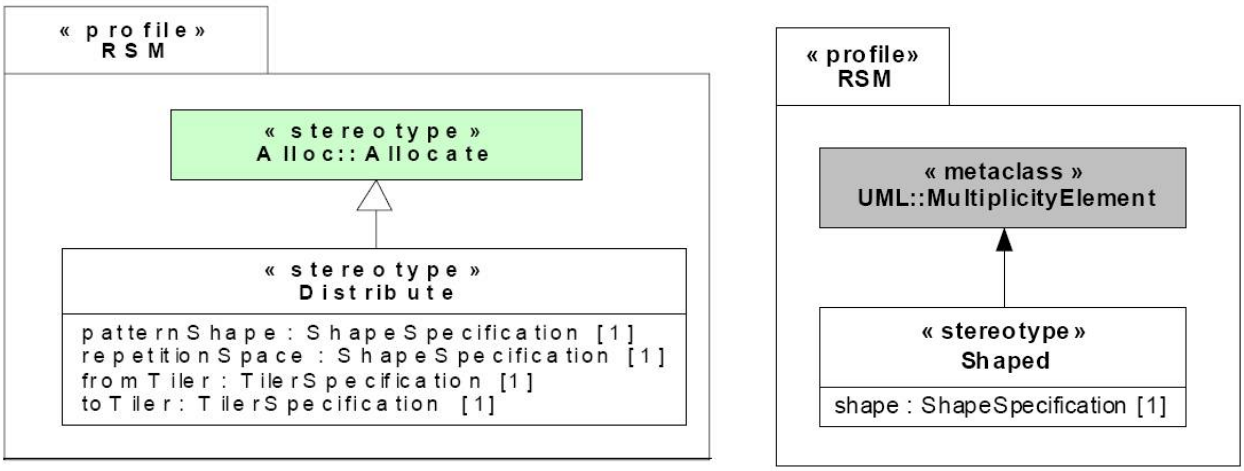
Les domaines d'application tels que le traitement de signal ou traitement d'image exigent habituellement des calculs intensifs de données à effectuer, éventuellement de façon parallèle, et avec l'aide de plusieurs unités de calcul. Dans le domaine des systèmes embarqués, nous appelons ce genre de traitement "*calcul intensif*". L'annexe RSM (*Repetitive Structure Modeling*) du standard MARTE [56] a pour objectif de proposer des concepts de modélisation de haut niveau qui permettent de prendre en compte ce type de systèmes. Plus précisément, il décrit de manière compacte la régularité de la structure ou de la topologie d'un tel système. Les structures considérées sont composées de répétitions d'éléments structurels, reliés entr'eux via un motif (*pattern*) de connexion. Nous appelons ce type de structures "des structures répétitives". Les mécanismes permettant la spécification de ces répétitions sont utilisés afin de :

- Modéliser la plateforme d'exécution matérielle : dans le but d'exprimer de manière précise, tout le parallélisme valable de la plateforme.
- Modéliser l'application : dans le but d'exprimer le parallélisme potentiel (tâches et parallélisme des données) de l'application.
- Allocation : décrire l'allocation spatiale et temporelle de l'application sur le matériels de la plateforme d'exécution.

Dans le cas d'un parallélisme de données, une tâche T1 peut être allouée sur plusieurs unités de calculs (processeur ou accélérateurs matériels), par exemple 4 processeurs [P1..P4], il s'agit dans ce cas d'une allocation factorisée et parallèle, nous présentons dans ce qui suit une manière de modéliser ce cas de figure, selon notre méthodologie de conception.

Tout d'abord, notons que le paquetage d'allocation, présenté dans le chapitre 5, permet d'allouer les tâches du modèle d'application sur les composants du modèle d'architecture. Avec la combinaison du concept *distribute* (Cf.8.2(a)), introduit dans l'annexe RSM, l'allocation des tâches multiples sur un ou multiple unités de calcul (processeur ou accélérateur) peut être réalisée.

La figure 8.2(b) illustre un extrait du profil permettant de modéliser le concept «*shaped*». Ce stéréotype fournit une vue multidimentionnelle d'une collection d'éléments. Il permet au concepteur de spécifier une valeur uniquement pour l'étiquette du shape de l'élément répété. En effet, il peut saisir la valeur du shape à la place de la multiplicité de l'élément.



(a) Le stéréotype «distribute» dans le profil RSM

(b) Extrait du profil RSM définissant le stéréotype «shaped»

FIGURE 8.2: Extrait du profil RSM pour la modélisation du parallélisme de données

Selon notre flot de conception, nous définissons deux niveaux d'allocation : 1) l'allocation factorisée de la tâche répétée T1 sur les quatre processeurs, et 2) l'allocation physique qui permet de placer les différents processeurs sur les quatres régions reconfigurables différentes. Ainsi, la tâche répétée T1 ainsi que les quatre processeurs sont stéréotypés «*shaped*». L'allocation des deux modèles est de type «*distribute*» qui hérite de «*allocate*». De plus, le deuxième type d'allocation est exprimé par un *Tiler* qui permet de spécifier l'origine de la région. La figure 8.3 montre notre proposition de modélisation selon notre méthodologie proposée et suivant les concepts existants de MARTE.

8.2.2 Ordre de reconfiguration factorisée et parallèle

Dans cette section, nous présentons notre vision quant à l'implémentation d'une reconfiguration factorisée et parallèle. Ici, nous gardons le même scénario de la section précédente mais nous nous intéressons plutôt à la gestion des ordres de reconfiguration. Il s'agit de ce que nous avons évoqué précédemment *membrane*. Nous avons défini les concepts permettant la modélisation de la membrane dans le profil RecoMARTE mais nous présentons ici une proposition

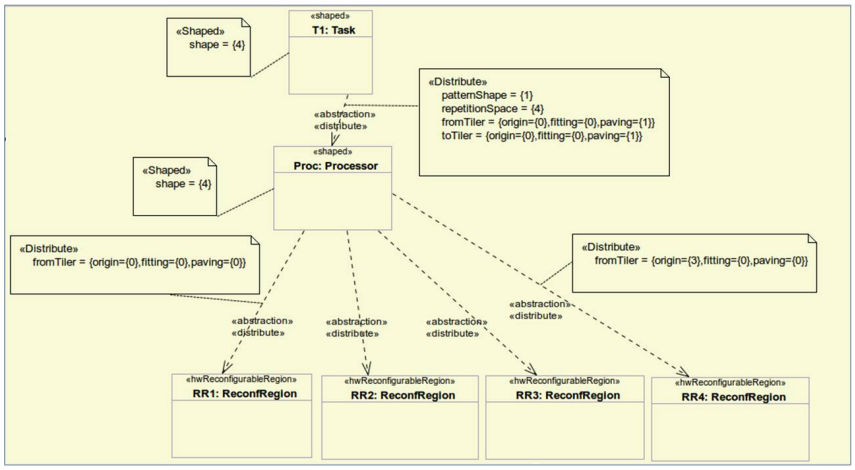


FIGURE 8.3: Modélisation du parallélisme de données selon notre approche

d'implémentation possible. Le mécanisme de membrane permet de configurer physiquement le système en conformité avec un ordre de reconfiguration reçu par un contrôleur. Par ailleurs, il s'agit ici de gérer : 1) la diffusion de l'ordre de reconfiguration suivant certaines contraintes imposées par un modèle de calcul, 2) l'interruption et la reprise d'une tâche et 3) la sauvegarde et la reprise de contexte.

Dans cet exemple, le contrôleur de la reconfiguration envoie un ordre pour exécuter la tâche T1 sur les 4 processeurs. Il y a ainsi une *membrane locale* (LocalMembrane LM) rattachée à chaque région, en d'autres termes quatre membranes dont chacune gère la sauvegarde de contexte d'un IP. Cet IP sera placé dans une région reconfigurable. Nous définissons aussi une <*super-membrane*> (SM) qui permet de communiquer avec les quatre LMs. Elle reçoit l'ordre de reconfiguration factorisée des 4 processeurs, de la part du contrôleur et le diffuse aux autres LMs. Chaque LM applique la sauvegarde de contexte avant la réception du nouveau bitstream. Par la suite, la super-membrane envoie une requête à l'ICAP afin de charger le nouveau bitstream et demande aux LMs de récupérer son nouveau contexte.

8.2.3 Automatisation de l'exploration

L'exploration architecturale est réalisée, actuellement, soit à un niveau très abstrait avec un langage de haut niveau (tel que UML, SysML), soit à un niveau d'abstraction plus bas (niveau transactionnel TLM ou CABA). Le résultat de l'exploration est obtenu à l'aide de techniques de simulation, de vérifications formelles ou de prototypage [17]. De plus, l'exploration vise l'adéquation de l'architecture avec l'application développée. Elle peut donc intervenir à tous les niveaux d'abstraction du flot de conception. En effet, une autre perspective à nos travaux et d'intégrer la phase d'exploration que nous avons évoquée précédemment durant ce manuscrit. Sur notre flot de conception, il peut y avoir plusieurs itérations dont nous citons quelques unes :

- *Niveau déploiement* de l'application sur l'architecture : Il peut y avoir un retour à ce niveau afin de modifier le choix du partitionnement par exemple ;
- *Niveau Application* : il est possible de faire un retour sur le choix du déroulage de boucle

ou le choix d'un ordonnancement ;

- *Niveau architecture :* dans le cas d'un éventuel changement de certaines caractéristiques d'un composant déjà utilisé lors de la modélisation de haut niveau, une modification dans le modèle de départ est donc inéluctable.
- *Niveau modélisation physique de l'FPGA* : à ce niveau, le concepteur a besoin de saisir les coordonnées des régions reconfigurables. Pour ce faire, il a besoin d'effectuer une exploration de l'espace de conception dirigée par les modèles afin de récupérer ces informations à partir d'un fichier UCF déjà synthétisé et simulé dans un projet avec les outils Xilinx.

D'autres objectifs peuvent être réalisés grâce à l'intégration de la DSE dans notre flot. En effet, suivant l'architecture choisie, ainsi que le déploiement de l'application sur celle-ci, il est possible que des ressources soient sur ou sous utilisées dans le temps d'exécution de l'application. Nous avons déjà défini dans RecoMARTE la relation entre les IPs et les ressources utilisées en termes de blocs logiques ou temps d'exécution. En exploitant cet espace, nous pouvons donc trouver le compromis adéquat.

Enfin, ces itérations peuvent faire partie d'un processus automatisé ce qui s'avère une solution intéressante qui permet d'accélérer le processus de conception. Pour ce faire, il faut définir l'ensemble des paramètres qui peuvent être modifiés dans un SoC.

Certains outils existants dans la littérature ont été développés afin de répondre à certains objectifs tels que la puissance [23], la consommation de surface et la vitesse [30], visant plusieurs architectures d'FPGA ainsi que le parallélisme des applications.

BIBLIOGRAPHIE

[1] ANDRES project. http://andres.offis.de/. 39

[2] HELP (High Level Models for Low Power Systems), 2010. http://www-verimag.imag.fr/PROJECTS/SYNCHRONE/HELP/?lang=en. 46

[3] COMPLEX Project : COdesign and power Management in PLatform-based design space EXploration, 2011. http://complex.offis.de/. 51

[4] Sodius Corporation MDWorkbench, 2011. http://www.mdworkbench.com/. 113

[5] AADL. The architecture analysis and design language (aadl) : An introduction, 2006. http://www.sei.cmu.edu/publications/documents/06.reports/06tn011.html. 63

[6] Acceleo. Acceleo - transforming models into code. http://www.eclipse.org/acceleo/. 114

[7] Altera 2010. Increasing design functionality with partial and dynamic reconfiguration in 28-nm fpgas , Technical Report WP-01137-1.0, 2010. 14

[8] X. An, S. Boumedien, A. Gamatié, and E. Rutten. CLASSY : a clock analysis system for rapid prototyping of embedded applications on MPSoCs. In *Proceedings of the 15th International Workshop on Software and Compilers for Embedded Systems*, SCOPES '12, pages 3–12. ACM, 2012. 5

[9] X. An, E. Rutten, J.-P. Diguet, N. Le Griguer, and A. Gamatie. Autonomic Management of Reconfigurable Embedded Systems using Discrete Control : Application to FPGA. Technical Report RR-8308, May 2013. 5

[10] C. André, F. Mallet, A. Mehmood Kahn, and R. de Simone. Modeling SPIRIT IP-XACT in UML MARTE. pages 35–40, March 2008. VI, 46, 47, 118

[11] T. Arpinen, T. Koskinen, E. Salminen, T. Hamalainen, and M. Hannikainen. Evaluating UML2 modeling of IP-XACT objects for automatic MP-SoC integration onto FPGA. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, 2009. 49, 57

[12] R. B. Atitallah. *Modèles et simulation des systèmes sur puce multiprocesseurs- Estimation des performances et de la consommation d'énergie*. PhD thesis, Université des Sciences et Technologies de Lille, Mars 2008. 45

[13] T. Basten, M. Hendriks, L. Somers, and N. Trcka. *UML for Electronic Systems Design : A Comprehensive overview*, volume 22. Springer New York, 2013. VII, 91

[14] S. Bayar and A. Yurdakul. Self-reconfiguration on Spartan-III FPGAs with compressed partial bitstreams via a parallel configuration access port (cPCAP) core. In *Research in Microelectronics and Electronics, 2008. PRIME 2008. Ph.D.*, pages 137–140, 2008. 15

[15] B. Blodget, S. McMillan, and P. Lysaght. A lightweight approach for embedded reconfiguration of fpgas. In *In Design, Automation and Test in Europe, DATE'03*, 2003. 15

[16] L. Bondé. *Transformations de Modèles et Interopérabilité dans la Conception de Systèmes Hétérogènes sur Puce à Base d'IP*. PhD thesis, Université des Sciences et Technologies de Lille, Décembre 2006. 18

[17] L. Bossuet. *Exploration de l'Espace de Conception des Architectures Reconfigurables*. PhD thesis, Université de Bretagne Sud, 2004. 170

[18] P. Boulet, C. Dumoulin, and A. Honoré. *Model Driven Engineering for distributed Real-Time Systems :From MDD Concepts to Experiments and Illustrations*, chapter Model Driven Engineering for System-on-Chip Design. CRC Press, 2006. 19

[19] J. Bézivin. In search of a basic principle for model driven engineering. In *The European Journal for the Informatics Professional*, 2004. 20

[20] J.-P. Calvez. Spécification et conception conjointe des systèmes matériel/logiciel : Méthodologie et outil, 2003. 33

[21] S. Cherif, I. Quadri, S. Meftali, and J.-L. Dekeyser. Modeling reconfigurable Systems-on-Chips with UML MARTE profile : an exploratory analysis. In *13th Euromicro Conference on Digital System Design (DSD 2010)*, 2010. 6

[22] S. Cherif, C. Trabelsi, S. Meftali, and J.-L. Dekeyser. High Level Design of adaptive distributed controller for Partial Dynamic reconfiguration in FPGA. In *Conference on Design and Architectures for Signal and Image Processing (DASIP 2011)*, 2011. 43

[23] S. Choi, J.-w. Jang, S. Mohanty, and V. K. Prasanna. Domain-Specific Modeling for Rapid Energy Estimation of Reconfigurable Architectures. *J. Supercomput.*, 26(3) :259–281, 2003. 171

[24] T. A. Claasen. System on a chip : changing IC design today and in the future. *IEEE Micro*, 23(3) :20–26, 2003. VI, 62

[25] T. A. C. M. Claasen. System on a chip : changing IC design today and in the future. *Micro, IEEE*, 23(3) :20–26, 2003. 17

[26] DaRT team. GASPARD SoC Framework, 2009. http://www.gaspard2.org/. 34, 45, 57, 71, 84

[27] I. David Mallis, Si2. Si2 OpenAccess API Tutorial 11th Edition, 2010. 118

[28] J.-L. Dekeyser, P. Boulet, P. Marquet, and S. Meftali. Model Driven Engineering for SoC Co-Design. In *NEWCAS'05*. IEEE, 2005. 39

[29] Eclipse. Papyrus. http://www.eclipse.org/modeling/mdt/papyrus/. 114

[30] R. Enzler, T. Jeger, D. Cottet, and G. Tröster. High-Level Area and Performance Estimation of Hardware Building Blocks on FPGAs. In *Field-Programmable Logic and Applications : The Roadmap to Reconfigurable Computing*, volume 1896, pages 525–534. Springer Berlin Heidelberg, 2000. 171

[31] H. Espinoza, D. Cancila, B. Selic, and S. Gérard. Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems. In *Model Driven Architecture - Foundations and Applications*, volume 5562, pages 98–113. Springer Berlin Heidelberg, 2009. 32

[32] M. Eva. SSADM Version4 : A User's Guide . McGraw-Hill Publishing Co, 1994. 19

[33] A. Gamatié, S. L. Beux, E. Piel, A. Etien, R. B. Atitallah, P. Marquet, and J.-L. Dekeyser. A model driven design framework for high performance embedded systems. research report. Technical report, 2008. http://hal.inria.fr/inria-00311115/en. 34

[34] C. Glitia, P. Dumont, and P. Boulet. Array-OL with delays, a domain specific specification language for multidimensional intensive signal processing. *Multidimensional Syst. Signal Process*, 21, 2010. 78

[35] S. Guillet. *Modélisation et contrôle formel de la reconfiguration : Application aux systèmes embarqués dynamiquement reconfigurables*. PhD thesis, Université de Bretagne Sud, 2012. VII, 4, 105, 106, 160

[36] T. Henzinger and J. Sifakis. The Discipline of Embedded Systems Design. *IEEE Society Computer*, 40(10) :32–40, Octobre 2007. 17

[37] F. Herrera, H. Posadas, E. Villar, and D. Calvo. Enhanced IP-XACT Platform Descriptions for Automatic Generation from UML/MARTE of Fast Performance Models for DSE. In *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pages 692–699, 2012. VI, 52, 57

[38] F. Herrera and E. Villar. A framework for the generation from UML/MARTE models of IPXACT HW platform descriptions for multi-level performance estimation. In *Specification and Design Languages (FDL), 2011 Forum on*, pages 1–8, 2011. 52, 53

[39] A. Herrholz, E. Oppenheimer, P. Hartmann, A. Schallenberg, W. Nebel, C. Grimm, M. Damm, J. Haase, E. Brame, F. Herrera, E. Villar, I. Sander, A. Jantsch, A.-M. Fouilliart, and M. Martinez. The Andres Project : Analysis and Design of Run-Time Reconfigurable, Heterogeneous Systems. In *International Conference on Field Programmable Logic and Applications, 2007. FPL 2007.*, pages 396–401, 2007. V, 39, 40, 57

[40] E. L. Horta and J. W. Lockwood. Parbit : A tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (fpgas). Technical report, 2001. 15

[41] V. Jorgiano. *Dynamic and partial reconfigurable embedded systems design with UML*. PhD thesis, Université de Bretagne Sud Lorient, 2010. VI, 43, 44, 80

[42] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, J. Riihimäki, and K. Kuusilinna. UML-based multiprocessor SoC design framework. *ACM Trans. Embed. Comput. Syst.*, 5(2) :281–320, 2006. 49

[43] A. Koudri, D. Vojtsiek, P. Soulard, C. Moy, J. Champeau, J. Vidal, and J.-C. L. Lann. Using MARTE in the MOPCOM SoC/SoPC Co-Methodology. In *Workshop MARTE, Colocated with DATE*, 2008. 44, 71

[44] P. Kukkala, J. Riihimäki, M. Hännikäinen, T. D. Hämäläinen, and K. Kronlöf. UML2.0 Profile for Embedded System Design. In *In Proceedings of the Design, Automation and Test in Europe Conference end Exhibition (DATE'05)*, pages 710–715, 2005. IX, 49, 51

[45] P. Kukkala, M. Setälä, T. Arpinen, E. Salminen, M. Hännikäinen, and T. D. Hämäläinen. Implementing a WLAN video terminal using UML and fully automated design flow. *EURASIP J. Embedded Syst.*, (1) :20–20, 2007. 49

[46] O. Labbani. *Modélisation à haut niveau du contrôle dans des applications de traitement systéma-tique à parallélisme massif*. PhD thesis, Université des Sciences et Technologies de Lille, 2006. V, 41, 42

[47] S. Le Beux. *Un flot de conception pour applications de traitement du signal systématique implémentées sur FPGA à base d'Ingénierie Dirigée par les Modèles*. PhD thesis, Université des Sciences et Technologies de Lille (USTL), 2007. 20, 22

[48] J.-F. Le Tallec and R. De Simone. SCIPX : a SystemC to IP-XACT extraction tool. In *ESLsyn : Electronic System Level Synthesis Conference*, 2011. VI, 47

[49] J.-F. Le Tallec, J. Deantoni, R. De Simone, B. Ferrero, F. Mallet, and L. Maillet-Contoz. Combining SystemC, IP-XACT and UML/MARTE in model-based SoC design. In *Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2011)*, 2011. VI, 46, 57

[50] S. Lecomte. *Méthodologie de conception de haut niveau orientée modèles pour les équipements de radio logicielle*. PhD thesis, Université Rennes 1, Novembre 2011. V, 2, 3, 23

[51] S. Lecomte, S. Guillouard, C. Moy, P. Leray, and P. Soulard. A co-design methodology based on model driven architecture for real time embedded systems. *Mathematical and Computer Modelling : An International Journal*, pages 53(3–4), February 2011. 62

[52] MARTES. The MARTES Project, 2009. http://www.martes-itea.org/public/info.php. 62

[53] G. Martin. UML for Embedded Systems Specification and Design : Motivation and Overview. In *DATE'02 Proceedings of the Conference on Design, Automation and Test in Europe*, Mars 2002. 22

[54] S. McMillan and S. Guccione. Partial Run-Time Reconfiguration Using JRTR. In *Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications*, pages 352–360, 2000. 14

[55] T. Mens and P. V. Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152 :125 – 142, 2006. 21

[56] Object Management Group Inc. Modeling and Analysis of Real-time and Embedded systems (MARTE). http://www.omgmarte.org/. 23, 36, 54, 64, 168

[57] Object Management Group Inc. The unified modeling language (UML). http://www.omg.org/uml. 19, 22

[58] Object Management Group Inc. UML Profile for Schedulability, Performance and Time (SPT). Technical report, OMG, September 2003. http://www.omg.org/cgi-bin/apps/doc?formal/03-09-01.pdf. 23, 24, 63, 84

[59] Object Management Group Inc. Final adopted OMG SysMl specification, 2006. http://www.omg.org/cgibin/doc?ptc/06-0504. 23, 31, 36

[60] Object Management Group Inc. UML profile for System on Chip (SoC), Version 1.0.1, 2006. IX, 23, 29, 32, 63

[61] Object Management Group Inc. UML Profil for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms specification. formal/2008-04-05, 2008. 23

[62] Object Management Group Inc. Meta object facility (mof) 2.0 query/view/transformation specification, version 1.1, 2011. http://www.omg.org/spec/QVT/1.1/. 112

[63] G. Ochoa-Ruiz, S. Cherif, O. Labbani-Narsis, E. Bourennane, S. Meftali, and J.-L. Dekeyser. Enabling partially reconfigurable IP cores parameterization and integration using IP-XACT and MARTE. 45

[64] G. Ochoa-Ruiz, S. Cherif, O. Labbani-Narsis, E. Bourennane, S. Meftali, and J.-L. Dekeyser. Facilitating IP deployment in a MARTE-based MDE methodology using IP-XACT : a XILINX EDK case study. In *International Conference on Reconfigurable Computing and FPGAs (Reconfig 2012)*, page 8, 2012. 45

[65] G. Ochoa-Ruiz, O. Labbani, E.-B. Bourennane, P. Soulard, and S. Cherif. A high-level methodology for automatically generating dynamic partially reconfigurable systems using IP-XACT and the UML MARTE profile. *Design Automation for Embedded Systems*, pages 1–36, 2012. VII, VIII, 5, 72, 114, 120, 121, 122, 133, 134, 136, 137

[66] K. Paulsson, M. Hübner, and J. Becker. On-line optimization of fpga power-dissipation by exploiting run-time adaption of communication primitives. In *In Proceedings of the 19th annual symposium on Integrated circuits and systems design, SBCCI '06, NewYork, NY, USA*. ACM, 2006. 14

[67] K. Paulsson, M. Hübner, and J. Becker. Dynamic power optimization by exploiting self-reconfiguration in xilinx spartan 3-based systems. *Microprocess. Microsyst*, 33(1) :46–52, 2009. 14

[68] S. Pillement and D. Chillet. High-level model of dynamically reconfigurable architectures. page 17, 2000. 40

[69] C. Plessl and M. Platzner. Virtualization of hardware -introduction and survey. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'04)*, 2004. 12

[70] I. R. Quadri. *MARTE based model driven design methodology for targeting dynamically reconfigurable FPGA based SoCs*. PhD thesis, Université des Sciences et Technologies de Lille, 2011. 41, 45, 100

[71] I. R. Quadri, S. Meftali, and J.-L. Dekeyser. Designing dynamically reconfigurable SoCs : From UML MARTE models to automatic code generation. In *Conference on Design and Architectures for Signal and Image Processing (DASIP 2010)*, 2010. 45

[72] I. R. Quadri, A. Muller, S. Meftali, and J.-L. Dekeyser. MARTE based design flow for Partially Reconfigurable Systems-on-Chips. In *17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC 09)*, 2009. 45

[73] QVTO. Eclipse Model To Model (M2M) Project. `http://m2m.eclipse.org/`. 112, 114

[74] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A SOC Design Methodology Involving a UML2.0 Profile for SystemC. In *In Proceedings of the Design, Automation and Test in Europe Conference end Exhibition (DATE'05)*, 2005. 29, 32

[75] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A model-driven design environment for embedded systems. In *Design Automation Conference (DAC 2006)*, pages 915–918, 2006. 63

[76] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. Designing a Unified Process for Embedded Systems. In *Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES '07.*, pages 77–90, 2007. 36, 39

[77] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. UML and MDA for Transactional Level Transactional Level Modeling. In *In UML-SoC, 2007*, 2007. 36, 39, 57

[78] M. Rieder, R. Steiner, C. Berthouzoz, F. Corthay, and T. Sterren. Synthesized UML, a Practical Approach to Map UML to VHDL. 3943 :203–217, 2006. 39

[79] S. Rouxel. *Modélisation et Caractérisation de Plates-Formes SoC Hétérogènes : Application à la Radio Logicielle*. PhD thesis, Université de Bretagne Sud, Decembre 2006. V, 37, 38, 57

[80] SAE. Architecture analysis and design language, January 2009. `http://standards.sae.org`. 22

[81] T. Schattkowsky, T. Xie, and W. Mueller. A UML frontend for IP-XACT-based IP management. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 238–243, 2009. VI, 47, 48

[82] S. Sendall and W. Kozaczynski. Model transformation : the heart and soul of model-driven software development. *Software, IEEE*, 20(5) :42–45, 2003. 21, 112

[83] M. Simonot and V. Aponte. A Declarative Formal Approach to Dynamic Reconfiguration. In *IWOCE '09 Proceedings of the 1st international workshop on Open component ecosystems*, pages 1–10. ACM New York, 2009. 92

[84] L. Singhal and E. Bozorgzadeh. Heterogeneous Floorplanner for FPGA. In *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2007. FCCM 2007.*, pages 311–312, 2007. 54, 56

[85] L. Singhal and E. Bozorgzadeh. Novel multi-layer floorplanning for Heterogeneous FPGAs. In *International Conference on Field Programmable Logic and Applications, 2007. FPL 2007, Amsterdman, Netherlands.*, pages 613–616, 2007. VI, 54, 55, 56

[86] C. Szyperski. ACM Press and Addison-Wesley, New York, N.Y, 1998. 98

[87] S. TAHA. *Modélisation conjointe logiciel/matériel de systèmes temps-réel*. PhD thesis, Université des Sciences et Technologies de Lille, 2008. VI, 53, 54, 55

[88] S. Taha, A. Radermacher, S. Gerard, and J. Dekeyser. An Open Framework for Detailed Hardware Modeling. In *International Symposium on Industrial Embedded Systems, 2007. SIES '07*, pages 118–125, 2007. 45

[89] H. Tan, R. F. Demara, A. Ejnioui, and J. D. Sattler. Complexity and performance evaluation of two partial reconfiguration interfaces on fpgas : a case study 1, 2006. 15

[90] H. Tardieu, A. Rochfeld, and R. Colletti. La Méthode Merise : Principes et outils. Editions d'Organisation, 1991. 19

[91] C. Trabelsi. *Contrôle matériel des systèmes partiellement reconfigurables sur FPGA : de la modélisation à l'implémentation*. PhD thesis, Université des Sciences et Technologies de Lille, 2013. 43

[92] M. van Hintum and P. Williams. The Value of High Quality IP-XACT XML, 2011. 120

[93] J. Vidal, F. de Lamotte, G. Gogniat, J.-P. Diguet, and P. Soulard. IP reuse in an MDA MPSoPC co-design approach. In *International Conference on Microelectronics (ICM)*, pages 256–259, 2009. 45

[94] J. Vidal, F. de Lamotte, G. Gogniat, J.-P. Diguet, and P. Soulard. UML design for dynamically reconfigurable multiprocessor embedded systems. In *DATE'10 Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1195–1200, 2010. 43, 57, 77

[95] Y. Wanderperren, W. Mueller, and W. Dehaene. *UML for Electronic Systems Design : A Comprehensive overview*, volume 12. 2008. 22

[96] Y. Wang, X.-G. Zhou, B. Zhou, L. Liang, and C.-L. Peng. A MDA based SoC Modeling Approach using UML and SystemC. In *The Sixth IEEE International Conference on Computer and Information Technology, CIT '06*, pages 245–245, 2006. 35, 39, 57

[97] P. Wattebled, J.-P. Diguet, and J.-L. Dekeyser. Membrane-based design and management methodology for parallel dynamically reconfigurable embedded systems. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, pages 1 – 8. IEEE, 2012. 5, 92

[98] Xilinx (2003). Two Flows for Partial Reconfiguration : Module Based or Difference Based, 2003. 14, 16

[99] Xilinx (2004). Two Flows for Partial Reconfiguration : Module Based or Difference Based, 2004. 14

[100] Xilinx (2009). PlanAhead user's guide, Xilinx UG632, 2009. 72, 114

[101] Xilinx (2010). Early Access Partial Reconfigurable Flow, 2010. 14

[102] Xilinx (2010). Virtex-4 family overview product specification. Technical Report DS112 (v3.1). Technical report, 2010. 89

[103] Xilinx (2011). Constraints guide. technical report ug625. Technical report, 2011. 128, 129

[104] Xilinx (2011). EDK concepts, tools, and techniques a hands-on guide to effective embedded system design, 2011. 72, 114

[105] Xilinx (2011). Embedded system tools reference guide, Xilinx UG111, 2011. 72, 114

[106] Xilinx (2011). Partial Reconfiguration User Guide UG702, 2011. 17, 89

[107] Xilinx (2011). Platform specification format reference manual, Xilinx UG642, 2011. 72, 114, 130

[108] Xilinx (2012). P. R. U. Guide. Ug702 (v14.3), October 16, 2012. 14

[109] Xilinx (2012). V.-. F. Configuration. Ug360 (v3.5), September 11, 2012. 15

[110] Q. Zhu, R. Oishi, T. Hesegawa, and T. Nakata. Integrating UML into SOC Design Process. In *In Proceedings of the Design, Automation and Test in Europe (DATE'05)*, 2005. V, 29

[111] Y. Zhu, Z. Sun, W.-F. Wong, and A. Maxiaguine. Using UML 2.0 for system level design of real time SoC platforms for stream processing. In *Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on*, pages 154–159, 2005. 39

Approche basée sur les modèles pour la conception des systèmes dynamiquement reconfigurables : de MARTE vers RecoMARTE

Résumé : Dans cette thèse, nous proposons une méthodologie de co-conception des systèmes dynamiquement reconfigurables basés sur FPGA. Notre méthodologie s'appuie sur l'Ingénierie Dirigée par les Modèles (IDM) pour la mise en œuvre de notre flot de conception, dont la spécification des modèles est décrite avec le profil standard UML MARTE (*Modeling and Analysis of Real-Time and Embedded Systems*). Les travaux présentés dans ce manuscrit visent à garantir la flexibilité, la réutilisabilité et l'automatisation afin de faciliter le travail du concepteur et d'améliorer sa productivité. La première contribution de cette thèse réside dans la modélisation à haut-niveau d'abstraction permettant de cacher un grand nombre de détails d'implémentation. Nous avons donc défini un flot de conception pour la modélisation des FPGAs dynamiquement reconfigurables, basé sur l'IDM afin d'assurer l'automatisation de la génération de code. Suivant ce flot, plusieurs modèles sont créés moyennant principalement les concepts du profil MARTE. Cependant, la modélisation de certains concepts de la reconfiguration dynamique des FPGAs a nécessité des extensions dans MARTE que nous avons identifiées et intégrées dans un nouveau profil qui étend MARTE baptisé RECOMARTE (*Reconfigurable MARTE*). La seconde contribution est l'automatisation de la chaîne de transformations et la validation expérimentale. Afin d'assurer l'automatisation de notre flot de conception vers la génération du code, une chaîne de transformations a été utilisée. Le modèle final en MARTE résultant du flot de conception est donné comme entrée à cette chaîne. Nous passons ainsi d'un modèle MARTE/RecoMARTE vers une description intermédiaire selon le standard IP-XACT afin de générer finalement des fichiers décrivant le système complet dans l'environnement XPS de Xilinx. Cette automatisation permet d'accélérer la phase de conception et éviter les erreurs dues à la manipulation directe des détails. Enfin, un exemple d'application de traitement d'image a été élaboré afin de démontrer et valider notre méthodologie. Ceci a fait apparaître les avantages de nos contributions en termes de réutilisabilité de la conception et l'automatisation.

Mots-clés : UML MARTE , Modélisation haut-niveau, Ingénierie Dirigée par les Modèles, Flot de conception, Reconfiguration dynamique partielle, FPGA .

A model driven based approach for the design of dynamically reconfigurable systems : from MARTE to RECOMARTE

Abstract : The works presented in this dissertation propose a co-design methodology of dynamically reconfigurable systems based on FPGA. Our methodology is based on the Engineering Model Driven approach (MDE) and the models specification is done in UML MARTE (*Modeling and Analysis of Real-Time and Embedded Systems*) profile. It aims to ensure flexibility, reusability and automation to facilitate the work of designer and improve his productivity. The first contribution related to this thesis is identifying parts of dynamically reconfigurable FPGA that can be modeled at high abstraction levels. So, we defined a design flow based on the MDE to ensure the automation of code generation. Using this flow, several models are created mainly through MARTE profile concepts. However, the modeling concepts of dynamic reconfiguration on FPGAs required extensions in MARTE. Thus, we identified the missing concepts to be integrated in a new profile that extends MARTE called RecoMARTE. The second contribution allows the chain automation and experimental validation. To integrate our design flow and to automate code generation, a processing chain was used. The final model resulting from the proposed MARTE-based design flow is given as input to this chain. We thereby move from MARTE/RecoMARTE models to an intermediate description according to the IP-XACT standard to finally generate files describing the complete system in the Xilinx XPS environment. This automation allows to accelerate the design phase and avoid errors due to the direct manipulation of these details. Finally, the proposed MARTE-based design flow and transformation chain were used for an image processing system design, which showed the benefits of our contributions in terms of design reusability and automation.

Keywords : UML MARTE , high level modeling, Model-Driven Engineering, Design flow, Partial dynamic reconfiguration, FPGA .