Méthodologie de conception de haut niveau orientée modèles pour les équipements de radio logicielle

Stéphane Lecomte

N° d'ordre : 4330 **ANNÉE 2011**

UNIVERSITÉ DE RENNES 1

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Electronique

Ecole doctorale Matisse

présentée par

Stéphane LECOMTE

préparée à l'unité de recherche SCEE
(Signal, Communication et Electronique Embarquée)
(Institut d'Electronique et de Télécommunications de Rennes)

Méthodologie de conception basée sur des modèles de haut niveau pour les systèmes de radio logicielle

Thèse soutenue à Supélec
le 23 novembre 2011

devant le jury composé de :

Jean-Marc JEZEQUEL
Professeur à l'Université de Rennes 1 / *président*

Lionel TORRES
Professeur à l'Université de Montpellier 2 / *rapporteur*

Robert de SIMONE
Directeur de recherche à l'INRIA Sophia-Antipolis / *rapporteur*

Christophe MOY
Professeur à Supélec / *directeur de thèse*

Pierre LERAY
Professeur à Supélec / *co-directeur de thèse*

Pierre HOUEIX
Lab. Manager chez Technicolor / *examinateur*

A tous ceux qui croient encore en l'innovation française,

Remerciements

En premier lieu, je tiens à remercier Christophe Moy et Pierre Leray, enseignants-chercheurs à Supélec sur le campus de Rennes, pour m'avoir encadré durant ces trois années de thèse et de m'avoir guidé ainsi qu'éclairé de leurs précieux conseils. Je tiens également à remercier Jacques Palicot, responsable de l'équipe SCEE de l'IETR, pour m'avoir accueilli dans l'équipe.

Je tiens également à remercier toutes les personnes chez Technicolor sans qui cette thèse n'aurait pas eu lieu. Je pense en premier lieu à Henk Heijnen et Jean-Christophe Le Lann qui m'ont apporté toute leur confiance dès le début de ce projet et suivi durant les premiers mois de cette thèse au sein de l'entité Thomson Silicon Component. Je tiens également à remercier Franck Lamouroux et Pierre Houeix pour leur accueil au sein de l'entité Research & Innovation de Technicolor. Le bon déroulement de cette collaboration entre Technicolor et Supélec n'aurait pu se faire sans le soutien scientifique de Samuel Guillouard. Je souhaite également remercier Sylvie Jouault et Sandra De Beauchesne pour leur assistance administrative très précieuse. Enfin, je tiens également à remercier toutes les personnes que j'ai côtoyées chez Technicolor auprès de qui j'ai passé trois années riches, je tenais à citer tout particulièrement mes collègues du « coin café » : Bernard, Franck, Vincent et Patrick. Je voulais également remercier Anne Huchet pour le temps qu'elle a passé à lire et valider mes articles.

Je tiens également à remercier tout particulièrement les partenaires du projet MOPCOM SoC/SoPC avec qui j'ai passé des moments conviviaux et enrichissants, tant sur le plan personnel que professionnel : Denis, Ali, Joël, Jean-Christophe, Frédéric, Guy, Jorgiano, Florent, Didier, Gilles, Christophe, Pierre et Philippe pour son aide précieuse sur les outils de modélisation et de transformation de modèle.

Pour finir je remercie toutes les personnes qui ont contribué de près et de loin au bon déroulement de cette thèse, le personnel de Supélec, ainsi que toute l'équipe SCEE et notamment Loïg Godard et Wassim Jouini.

Je ne pourrais terminer ces remerciements sans y mentionner mes proches pour leur soutien moral pendant ces trois années, et tout particulièrement ma compagne, Emilie, qui m'a supporté, soutenu, qui m'a apporté une aide précieuse lors de la relecture de ce document.

Table des matières

Acronymes & Abréviations

La signification d'une abréviation ou d'un sigle est indiquée lors de la première apparition dans le présent document. De plus, l'abréviation d'un sigle peut être soit en français soit en anglais. Le plus souvent le terme anglais est le plus répandu.

3GPP	*3rd Generation Partnership Project*
AAA	*Adéquation Algorithme Architecture*
AADL	*Architecture Analysis & Design Language*
ADM	*Architecture Dirigée par les Modèles*
AGC	*Apollo Guidance Computer*
AML	*Abstract Modeling Level*
ASIC	*Application Specific Integrated Circuit*
ATL	*Atlas Transformation Language*
AUTOSAR	*AUTOmotive System ARchitecture*
BB	*Bande de Base*
BPSK	*Binary Phase-Shift Keying*
CAO	*Conception Assistée par Ordinateur*
CAN	*Convertisseur Analogique Numérique*
CIM	*Computation Independent Model*
CNA	*Convertisseur Numérique Analogique*
CSP	*Communicating Sequential Processes*
CWM	*Common Warehouse Metamodel*
DDR	*Double Data Rate*
DE	*Discrete Event*
DMA	*Direct Access Memory*
DML	*Detail Modeling Level*
DoDAF	*DoD Architecture Framework*
DRM	*Detailed Resource Modeling*
DSL	*Domain Specific Language*
DSP	*Digital Signal Processor*
Ecore	*EMF core*
EDA	*Electronic Design Automation*
EDAC	*Electronic Design Automation Consortium*
EDGE	*Enhanced Data rates for GSM Evolution*
EML	*Execution Modeling Level*
EMF	*Eclipse Modeling Framework*
ESL	*Electronic System Level*
FE	*Front End*
FI	*Fréquence Intermédiaire*
FIR	*Finite Impulse Response*
FPGA	*Field Programmable Gate Array*
FSM	*Finite State Machine*
GE	*Giga Ethernet*
GPS	*Global Positioning System*

GSM	*Global System for Mobile*
GUI	*Graphical User Interface*
HDCRAM	*Hierarchical and Distributed Cognitive Radio Management*
HDL	*Hardware Design Language*
HLS	*High Level Synthesis*
IDM	*Ingénierie Dirigée par les Modèles*
IEEE	*Institute of Electrical and Electronics Engineers*
IP	*Intellectual Property*
ITRS	*International Technology Roadmap for Semiconductors*
JTRS	*Joint Tactical Radio System*
KPN	*Kahn Process Networks*
LTE	*Long Term Evolution*
M2M	*Model-To-Model (modèle vers modèle)*
MAC	*Medium Access Control*
MARTE	*Modeling and Analysis of Real Time Embedded system*
MCSE	*Méthodologie de Conception des Systèmes Electroniques*
MDA	*Model Driven Architecture*
MDD	*Model Driven Development*
MDE	*Model Driven Engineering*
MIMO	*Multiple Input Multiple Output*
ML	*Maximum Likelihood*
MMITS	*Multifunction Information Transfer System*
MoC	*Modèles de Calculs* ou *Model of Computing*
MODAF	*MOD Architecture Framework*
MOF	*Metadata Object Facility*
MOPCOM	*Modélisation et spécilisatiOn de Plates-formes et COmposants Mda*
NFP	*Non Functional Properties*
NoE	*Network Engine*
OCL	*Object Constraint Language*
OFDM	*Orthogonal Frequency Division Multiplexing*
OMG	*Object Management Group*
OMT	*Object Modeling Technique*
OOSE	*Object Oriented Software Engineering*
OSCI	*Open SystemC Initiative*
OSI	*Open Systems Interconnection*
PIM	*Platform Independent Model*
PHY	*PHysIc layer*
PM	*Platform Model*
PSM	*Platform Specific Model*
PV	*Programmer View*
PV-T	*Programmer View - Timed*
QAM	*Quadrature Amplitude Modulation*
QPSK	*Quadrature Phase-Shift Keying*
QoS	*Quality of Service*
QVT	*Query Views Transformation*
RAM	*Random Access Memory*
RF	*Radio Fréquence*
RI	*Radio Intelligente*
RIF	*Réponse Impulsionnelle Finie*
RL	*Radio Logicielle*

RLR	*Radio Logicielle Restreinte*
ROM	*Read-only Memory*
RT	*Real Time*
RTES	*Real Time Embedded Systems*
RTL	*Register Transfer Level*
RTOS	*Real Time Operating Systems*
RVB	*Rouge Vert Bleu*
SCA	*Software Communication Architecture*
SDF	*Sequential Data Flow*
SDR	*Software Defined Radio*
SIMO	*Single Input Multiple Output*
SoC	*System on Chip*
SoPC	*System on Programmable Chip*
SPEM	*Software Process Engineering Modeling*
SDRAM	*Synchronous Dynamic Random Access Memory*
SPT	*Schedulability, Performance and Time*
STOBC	*Space Time Orthogonal Block Coding*
SysML	*System Modeling Language*
TDMA	*Time Division Multiple Access*
TEB	*Taux d'Erreur Binaire*
TIC	*Technologies de l'information et de la Communication*
TLM	*Transaction Level Modeling*
TNS	*Traitement Numérique du Signal*
UHF	*Ultra High Frequency*
UML	*Unified Modeling Language*
UMTS	*Universal Mobile Telecommunications System*
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*
VSL	*Value Specification Language*
WiFi	*Wireless Fidelity*
WiMAX	*Worldwide Interoperability for Microwave Access*
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*
XP	*eXtreme Programming*

Introduction

En ce début de XXI^{ème} siècle, l'électronique est de plus en plus présente dans les objets les plus divers qui nous entourent. Il semble aujourd'hui difficile de revenir en arrière et ainsi se priver de l'électronique dans notre quotidien. Celle-ci est présente dans les domaines les plus variés tels que la médecine, les transports, les systèmes de communications et d'informations, l'habitat, etc. En un siècle d'existence, l'électronique n'a pas cessé d'évoluer depuis la création de la diode à tube par le scientifique anglais John FLEMING en 1904 jusqu'à l'arrivée du premier circuit intégré en 1958 développé par Jack KILBY chez Texas Instrument, en passant par la création du premier transistor bipolaire à jonction en 1949, et la réalisation du premier transistor bipolaire sur un substrat silicium en 1954. Depuis les années 1960, l'évolution de l'électronique passe par une course effrénée à la miniaturisation des transistors et l'augmentation de la densité d'intégration de ces mêmes transistors sur une puce de silicium.

En 1965, l'américain Gordon MOORE (co-fondateur d'Intel) énonce les premiers fondements de ce qui sera appelée la « loi[1] » de Moore [1]. En 1975, après l'apparition des premiers microprocesseurs tel que le 4004 d'Intel, Moore réévalua sa prédiction initiale en prédisant un doublement de la densité d'intégration des transistors sur une puce de silicium tous les deux ans (cf. Figure 1).

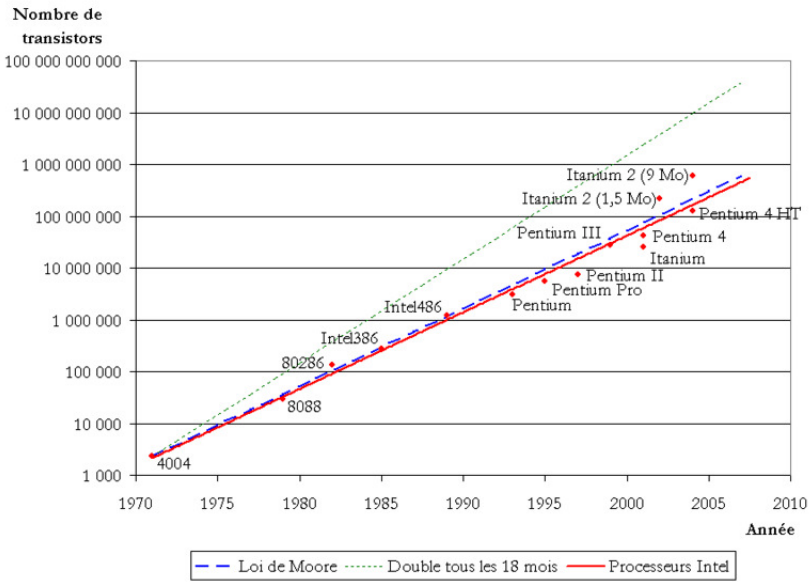


Figure 1. Comparaison entre la loi de Moore et la croissance du nombre de transistors dans les microprocesseurs d'Intel [2].

[1] Terme utilisé de manière abusive car il s'agit plutôt d'une conjoncture présentant des suppositions qu'une véritable loi régie par une démonstration mathématique.

Aujourd'hui cette course à la miniaturisation permet notamment de concevoir des systèmes électroniques intégrés nomades (i.e. embarqués), tels que les téléphones mobiles, toujours de plus en plus complexes. Ainsi cette évolution permet aujourd'hui d'intégrer toujours plus de fonctionnalités au sein d'une même puce.

Comme illustré sur la Figure 2, un système électronique nécessitait auparavant une multitude de puces, chacune ayant une fonction spécifique unique, interconnectées entre elles sur une carte électronique. Aujourd'hui, on parle de système sur puce (ou SoC – *System on Chip* – en anglais) ou de système sur composant programmable (ou SoPC – *System on Programmable Chip* – en anglais), tel que les FPGA (*Field Programmable Gate Array*). Cependant ce propos doit être nuancé car les dernières générations de puces intègrent certes plusieurs fonctionnalités mais n'intègrent pas la totalité d'un système même si la technologie le permet. En effet, par exemple, l'ensemble des fonctionnalités d'un téléphone mobile nécessite aujourd'hui encore l'utilisation plusieurs puces.

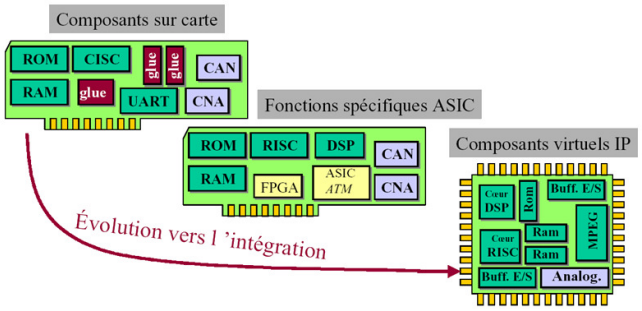


Figure 2. Evolution de la capacité d'intégration [3].

De plus, afin de répondre au mieux aux besoins des utilisateurs mais aussi environnementaux[2] dans lequel se trouve le dit système, et les contraintes que cela impose, le système électronique embarqué est de plus en plus hétérogène. Ainsi, aujourd'hui, les systèmes électroniques embarqués sont constitués de composants électroniques analogiques et numériques, et intègrent de plus en plus de fonctionnalités assurées par des traitements logiciels. L'arrivée massive de l'informatique dans les systèmes électroniques embarqués permet d'accroître considérablement la valeur ajoutée d'un produit et de lui donner plus de souplesse. De ce fait, la conception de tels systèmes fait appel à différents domaines de compétences (électronique, informatique, système…) qu'il convient de faire cohabiter dans une démarche de conception conjointe.

[2] Ici le terme environnemental englobe l'aspect développement durable, l'aspect concurrentiel du marché, les contraintes de sécurité, de conformité aux normes (environnementale, fabrication, qualité, etc.).

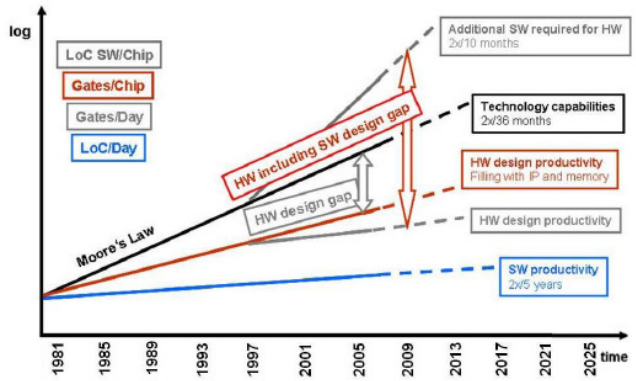


Figure 3. Evolution de la productivité en comparaison de l'évolution de la capacité d'intégration selon MOORE [4].

Avec une augmentation de la capacité d'intégration de 58% par an, et une demande de fonctionnalités croissantes, la conception de systèmes électroniques embarqués est un réel challenge pour les équipes de développement. En effet, au delà du système en lui-même, le processus de conception du dit système est un enjeu majeur dans un marché en constante évolution et très concurrentiel. Depuis les années 1980 le processus de conception évolue afin d'accroître la productivité. Cependant le constat fait par l'ITRS (*International Technology Roadmap for Semiconductors*) est sans appel. Comme la Figure 3 l'illustre très clairement, la productivité des systèmes électroniques n'évolue pas aussi rapidement que la capacité d'intégration. En effet, la productivité des équipes de conception n'augmente que de 21% par an par rapport au 58% d'augmentation de la capacité d'intégration. Même si un ralentissement de cette capacité pourrait se faire sentir dans les années à venir, comme l'indique de nombreux scientifiques et comme l'a lui-même précisé G. MOORE, la productivité ne semble pas en mesure, avec les démarches de conception actuellement utilisées, de suivre et encore moins de rattraper le retard pris depuis les années 1980.

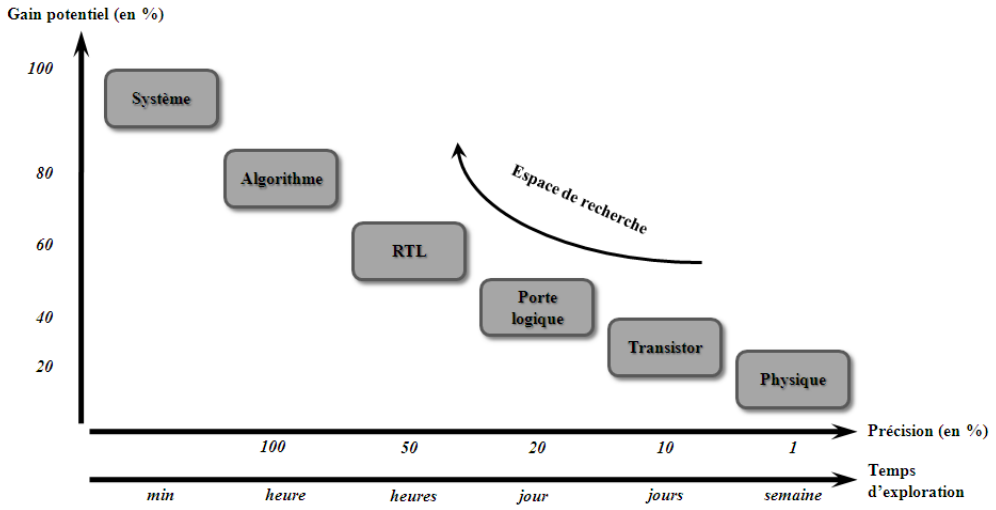


Figure 4. Impact de l'élévation du niveau de représentation d'un système électronique sur le temps d'exploration, la précision par rapport au gain potentiel de productivité [5].

Depuis les années 1970, et l'apparition des premiers systèmes complexes, l'évolution de la démarche de conception des systèmes électroniques embarqués passe par une élévation du niveau de représentation du système comme illustré par la Figure 4. Nous sommes passés d'une conception « full custom » (description du schéma électrique du système) dans les années 1970, à une conception « cell based » (description sous forme de schéma de porte logique du système puis sous forme de langage de description matériel) dans les années 1980 à 2000, à une conception « SoC » de nos jours. La démarche de conception actuelle favorise la réutilisation de fonctionnalités, déjà développées, sous la forme de bloc réutilisable (intitulé IP – *Intellectual Property* – en anglais), la mise en place d'outils d'aide à la conception pour les étapes fastidieuses, et l'utilisation de synthèse comportementale. On parle aujourd'hui de conception électronique au niveau système (ESL – *Electronic System Level* – en anglais) assisté par une multitude d'outils proposés par les industriels de l'EDA[3] (*Electronic Design Automation*).

Cette évolution est nécessaire afin non seulement de prendre en compte l'évolution des besoins (toujours plus importants), et celle des technologies, mais aussi afin de se plier à la loi du marché. Les responsables des équipes de développement cherchent aujourd'hui à optimiser le temps et le coût de développement d'un produit (Time-to-Market en anglais).

De plus, avec l'arrivée de l'informatique dans les systèmes électroniques embarqués, la cohérence de la démarche de conception descendante (en opposition à une démarche ascendante) est rompue dans le cadre d'une conception conjointe comme l'illustre la Figure 5. En effet, la démarche de conception, depuis le cahier des charges jusqu'à la réalisation physique, d'un système purement électronique (ne combinant pas matériel et logiciel) s'enchaîne avec cohérence comme illustré par le synoptique à gauche de la Figure 5.

Le synoptique à droite de la Figure 5 illustre une démarche de conception conjointe (ou co-conception) matérielle/logicielle. Nous utiliserons le terme co-conception tout au long de ce mémoire. Une rupture de ce flot de co-conception apparaît une fois l'étape de partitionnement effectuée (choix des fonctionnalités du système mises en œuvre en logiciel et celles réalisées en matériel). Nous entendons ici par « logiciel », le développment du code qui sera exécuté sur un processeur. Nous entendons ici par « matériel », le développement de circuits életroniques numériques et/ou de code destiné à être exécuté sur une cible matérielle comme un FPGA. Les processus de développement du logiciel et du matériel sont différents : les équipes de développement n'ont pas les mêmes compétences, les sémantiques de modélisation et de réalisation utilisées sont différentes, les outils de développements et de vérifications sont différents. De ce fait, la phase de co-simulation et d'intégration est très souvent complexe et révèle de nombreuses erreurs dues à la séparation des conceptions sur toute une partie du flot.

[3] EDA : catégorie d'outils servant à la conception et la production des systèmes électroniques allant des circuits imprimés jusqu'aux circuits intégrés.

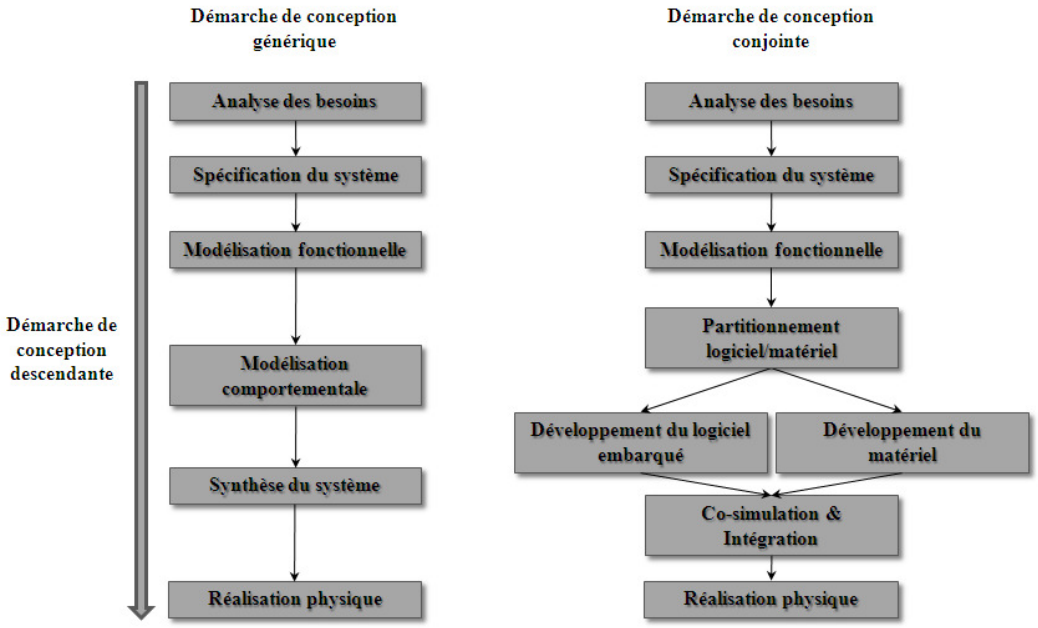


Figure 5. Illustration des étapes de la démarche de conception d'un système électronique à gauche et d'un système combinant matériel (électronique) et logiciel (informatique).

Les méthodologies actuellement utilisées pour développer des systèmes électroniques embarqués complexes sont plus des démarches de conception que de réelles méthodologies formalisées. Chaque industriel utilise ses propres codes et formalismes pour chacune des étapes de la conception d'un système électronique embarqué. Ceci peut être vu comme une forme de richesse mais peut se révéler comme un frein à la productivité et à la qualité.

Il ne faut pas non plus oublier de prendre en compte lors de la phase de conception le caractère nomade des systèmes embarqués et la qualité de service qu'ils doivent procurer. Ainsi, par exemple dans le cas d'un système de communication sans-fil, une communication ne doit pas être interrompue. Cette contrainte impose des contraintes temps réel sur le temps de calcul maximal admissible sur les données. De plus, le fait que le système soit nomade impose des contraintes sur les caractéristiques de la plate-forme matérielle telle qu'une limitation de la consommation (contrainte de plus en plus importante), une limitation de la puissance de calcul disponible (par exemple, les microprocesseurs embarqués dans les ordinateurs de bureau sont trop consommateurs en énergie pour être utilisés dans un système nomade), ou encore une limitation de la mémoire disponible, etc.

Enfin, avec toujours plus de fonctionnalités, un panel technologique toujours plus important, l'espace d'exploration architecturale, pour obtenir le système répondant au cahier des charges, est de plus en plus vaste. Ainsi le temps de conception est d'autant plus long et le nombre d'erreurs potentielles augmente également.

Ainsi les challenges pour les équipes de conception de systèmes électroniques embarqués sont nombreux et bien réels : systèmes de plus en plus complexes, capacité technologique, Time-to-Market. De plus tous les indicateurs semblent mettre en évidence une inadéquation entre ces challenges et la démarche de co-conception actuellement utilisée. Afin de lever ce verrou, et proposer un niveau de productivité plus proche de l'évolution technologique plusieurs pistes, pour certaines déjà mises en partie en œuvre, sont proposées par la communauté ESL (*Electronic System Level*) rapportées dans le rapport intitulé « Design » de l'ITRS [4]. Le Tableau 1 récapitule les enjeux et les solutions avancées afin d'augmenter la productivité de manière significative, en cohérence avec les avancées technologiques.

Tableau 1. Récapitulatif des challenges et des solutions préconisées dans la conception de système électroniques embarquées.

Challenges	Solutions
Complexité des systèmes ↗ Evolution rapide des technologies Time-to-Market ↘	➲Approche de conception de haut niveau ➲Approche de conception basée sur les modèles ➲Portabilité (indépendance entre modèle fonctionnelle et plate-forme matérielle cible) ➲Réutilisabilité
Communications entre les équipes (multi-compétences : système, logiciel, matériel, chef de projet, commercial, etc.)	➲Formalisme commun pour les différentes équipes de conception ➲Génération automatique de documentation
Obsolescence	➲Capitalisation des compétences et de l'expérience
Qualité du processus de développement	➲Méthodologie de conception formalisée ➲Test & Traçabilité

Une approche qui semble la plus prometteuse pour répondre à ces défis est basée sur l'architecture dirigée par les modèles. Elle est standardisée par l'OMG sous le nom de MDA – *Model Driven Architecture*. Cette approche est issue de l'IDM – *Ingénierie Dirigée par les Modèles* – (ou MDE – Model *Driven Engineering* – en anglais) plus connue dans le monde de l'informatique que dans celui de l'électronique.

Problématique de l'étude

Devant la complexité croissante des systèmes électroniques embarqués (des applications et des plate-formes d'exécution), notamment dans le domaine des radiocommunications, l'évolution des technologies, les démarches de co-conception actuellement utilisées ne permettent plus de maîtriser et limiter les temps de développement, tout en garantissant une qualité de la conception donnée. 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.

Plan du mémoire et contributions

Dans le premier chapitre nous donnons les principales caractéristiques d'un système électronique embarqué, présentons un exemple d'un tel système : un système de radiocommunication. Nous présentons également la radio logicielle, solution pour palier au manque de flexibilité des systèmes de radiocommunications. L'objectif de ce premier chapitre est de mettre en évidence la complexité des systèmes que nous aurons à concevoir.

Dans le second chapitre, nous mettons en évidence le besoin de nouvelles méthodologies de co-conception pour les systèmes électroniques embarqués. Nous proposons ensuite un tour d'horizon des nouvelles approches de conception conjointe basées sur les modèles.

Après avoir mis en évidence les avantages et les faiblesses des méthodologies présentées, nous présenterons dans le troisième chapitre une méthodologie de co-conception formalisée, basée sur les modèles, intitulée MOPCOM, proposée pour répondre aux nouveaux défis de la co-conception de systèmes embarqués. Dans la seconde partie de ce chapitre, nous présentons notre principale contribution à la méthodologie MOPCOM : la définition du niveau intermédiaire de modélisation intitulé Execution Modeling Level. Nous proposons dans la troisième partie de ce chapitre la mise en œuvre la plus complète (encore partielle cependant) qui ait été faite de l'ensemble de la méthodologie MOPCOM.

Puis, le quatrième chapitre est consacré à la génération de code matériel à partir de la modélisation MDA. Pour cela nous proposons de coupler avec un outil de synthèse comportementale avec la méthodlogie MOPCOM et nous mettons en avant la complémentarité de ceux deux approches. Nous présentons et comparons les résultats obtenus pour plusieurs stratégies d'exploration architecturale dans le cas d'un décodeur MIMO de Technicolor. La comparaison avec un code pré-existant développé entièrement à la main nous permettra de proposer une analyse sur la maturité du couplage MDA et synthèse comportementale.

Enfin, dans un contexte de radio logicielle, nous présentons une extension de la méthodologie MOPCOM afin de prendre en compte le caractère flexible des systèmes embarqués. Nous avons intégré dans la méthodologie MDA proposée l'architecture de gestion de radio logicielle de Supélec. Ceci a mené à valider le concept de simulation de modèles de haut niveau sur plate-forme radio réelle, permettant ainsi de confronter au plus tôt les modèles utilisés à différents niveaux de la conception, à la réalité d'implantation.

Chapitre 1 Des systèmes embarqués à la radio logicielle

Sommaire

Un système électronique embarqué, outre ses caractéristiques fonctionnelles toujours de plus en plus complexes, se caractérise par un nombre de paramètres à ne pas négliger durant la phase de conception du système de plus en plus important. Nous présentons les principales caractéristiques d'un tel système, tout au moins celles qui ont le plus de pertinence dans le cadre des thèmes de cette thèse. Les systèmes de radiocommunications sans-fil sont un cas typique de système embarqué, sur lequel nous allons insister dans ce document, sans perte de généralité. La méthodologie de co-conjointe logicielle/matérielle basée sur les modèles présentée dans ce mémoire est particulièrement bien adaptée à la conception des systèmes de radiocommunications. De ce fait, nous donnons une définition d'un système de radiocommunication et présentons un système de transmission sans-fil utilisé pour valider la méthodologie proposée. Enfin, nous présentons succinctement la radio logicielle. La radio logicielle propose une solution au manque de flexibilité des systèmes de radiocommunications actuels. Dans ce chapitre nous mettons en évidence la complexité des systèmes électroniques embarqués et de ce fait le grand nombre de paramètres à prendre en compte dans leur phase de conception. Le chapitre suivant met en évidence le besoin de nouvelles méthodologies de co-conjointe logicielle/matérielle pour de tels systèmes.

1.1 Les systèmes électroniques embarqués temps réel

Un système embarqué est un système autonome, composé dans notre étude d'une plate-forme matérielle (utilisant massivement des composants électroniques tout-en-un plus communément désignés par le terme SoC – *System on Chip*) et d'une partie logiciel. Cette dernière peut se composer de plusieurs couches : le(s) pilote(s), le micro-logicielle, et l'interface homme/machine. Le rôle d'un tel système est d'exécuter une tâche prédéfinie de manière autonome une fois le système installé dans son environnement d'utilisation. A l'origine des premiers systèmes embarqués de l'ère moderne (début de l'ère de la micro-électronique), la fonctionnalité de ces derniers était relativement simple et précise. Le système de guidage de la mission lunaire Apollo (AGC – *Apollo Guidance Computer*) est très souvent cité comme l'un des tous premiers systèmes embarqués de l'ère moderne. Aujourd'hui, un système électronique embarqué offre un/des services de plus en plus complexe(s). Les téléphones portables, les ordinateurs de bords des avions de lignes ou encore les set-top box sont des exemples de systèmes électroniques embarqués complexes.

De plus, ces systèmes font de plus en plus la part belle à l'informatique notamment en intégrant un système d'exploitation ou un noyau temps réel (ou encore intitulé RTOS – *Real Time Operating System* – en anglais). Ces RTOS ont pour rôle de gérer la mémoire, d'ordonnancer, de prioriser les tâches, et de garantir le respect des contraintes temps réel du système.

Outre ces caractéristiques fonctionnelles et physiques, un système électronique embarqué se caractérise par de nombreuses spécificités. Volontairement nous présentons dans ce document uniquement les principales caractéristiques des systèmes électronqiues embarqués que nous avons abordés dans le cadre de nos travaux. Nous ne parlerons pas par exemple, de sureté de fonctionnement (fiabilité, disponibilité du service, maintenabilité, sécurité) bien qu'elle soit très importante. Toutes les caractéristiques présentées doivent être prisent en compte tout au long de la conception du système.

1.1.1 L'environnement de fonctionnement

L'environnement dans lequel évolue le système embarqué peut être considéré comme une des caractéristiques du système. En effet, l'interaction entre le système embarqué et son environnement est plus ou moins étroite. De plus ce dernier est plus ou moins contrôlé (connaissance, prévisibilité) par les concepteurs du système. Une fois le système placé dans son environnement, les équipes de conception n'ont plus la main sur celui-ci, comme c'est le cas durant la phase de développement. De ce fait le système doit réagir en conséquence pour adapter son fonctionnement afin de toujours fournir le service auquel il est destiné.

1.1.2 L'autonomie

Outre le(s) service(s) offert(s) par le système, l'autonomie d'un système embarqué est souvent une des caractéristiques sur laquelle l'utilisateur du système va porter ses premiers jugements.

D'un côté, nous avons l'autonomie énergétique du système. Le système embarqué n'étant pas alimenté par une source d'énergie « infinie », il est impératif de définir le cadre de son autonomie et la manière dont il sera alimenté. L'autonomie énergétique du système dépend de nombreux paramètres parmi lesquels on compte la source d'énergie, la technologie de stockage de cette énergie, la puissance nécessaire, etc. Il est souvent prévu des scénarii différents du cas nominal pour augmenter l'autonomie du système (et implicitement réduire la consommation énergétique). Par exemple, sur les téléphones portables, les connectiques WiFi (*Wireless Fidelity*) et Bluetooth peuvent être désactivées afin de réduire la consommation et implicitement augmenter l'autonomie du téléphone.

D'un autre côté, nous parlons également d'autonomie fonctionnelle du système embarqué. En effet, les fonctionnalités du système sont définies à sa conception. De plus, le plus souvent, le système embarqué ne fonctionne pas en totale autonomie mais est intégré dans un système plus global qu'il contrôle (ou qui le contrôle). Nous pouvons citer comme exemple les ordinateurs de bords des avions ou encore les systèmes électroniques embarqués dans un véhicule.

Note : Il existe aujourd'hui des travaux sur l'intelligence artificielle ou encore sur la radio intelligente qui permet d'envisager des systèmes, plus autonomes, capables d'apprendre pour faire évoluer leur(s) fonctionnalité(s) et ainsi améliorer sensiblement leur autonomie fonctionnelle.

1.1.3 Les ressources

Les ressources à disposition du système embarqué pour accomplir les tâches qui lui incombent sont souvent limitées et induisent ainsi de fortes contraintes lors de la conception du système. Ces ressources sont de différents ordres :

- **Encombrement et poids** : le système embarqué requiert par un faible encombrement et un faible poids suivant l'environnement auquel il est destiné. L'optimisation de ces deux caractéristiques permet de gagner en ergonomie, en coût de fabrication (moins de matières utilisées pour la fabrication) et également en consommation énergétique. En contre-partie, cela complique souvent l'architecture du système afin d'optimiser l'espace ;
- **Energétique** : un système embarqué requiert par une consommation énergétique limitée. La consommation énergétique du système est implicitement liée à son autonomie. La source énergétique du système embarqué est le plus souvent limitée dans le temps. Par conséquent, il est impératif de prendre en compte ce paramètre et de prévoir des scénarii de replis du système dans les cas où la source d'énergie du système n'est plus en mesure de fournir suffisamment de puissance pour faire fonctionner le cas nominal. Ainsi, il est courant de réduire la puissance de calcul, ou/et éteindre des fonctionnalités du système afin de limiter la consommation. Toutefois ces scenarii de repli doivent bien évidemment respecter les caractéristiques de sûreté de fonctionnement du système ;

- **Puissance de calcul** : un système embarqué dispose d'une puissance de calcul (nombres d'opérations/seconde) limitée pour effectuer les tâches dont il a la charge. En effet, une opération étant coûteuse en énergie, un système embarqué ne peut intégrer des puces électroniques trop gourmandes en énergie tels que peuvent l'être les processeurs Intel Xeon[4] (processeurs multi-cœurs pour serveurs) ;
- **Capacité de stockage** : un système embarqué dispose d'une capacité de stockage de données limitée. En effet, tout comme une unité de calcul, une unité de mémorisation est consommatrice d'énergie. De plus, leur encombrement impose une limite au nombre de composants utilisés et donc à la capacité de stockage totale.

Il est donc nécessaire, lors de la phase de conception, de trouver un compromis entre toutes ces ressources et contraintes, afin d'aboutir à une architecture qui permet de supporter les fonctionnalités du système visé.

1.1.4 Les caractéristiques temporelles

Les caractéristiques temporelles d'un système passent par la définition (bornage) du temps d'exécution et de la durée de vie (échéance) des tâches du système, et dans le cas de tâches périodiques, la périodicité de celles-ci. Selon les systèmes, les contraintes temporelles sont plus ou moins « dures ». Un **système embarqué** est dit **temps réel** lorsque le respect des contraintes temporelles a autant d'importance que le résultat des tâches dans le fonctionnement global du système. Dans ce cas là, la mise en œuvre d'un RTOS est le plus souvent appropriée pour gérer et garantir les contraintes temporelles du système.

Les systèmes embarqués peuvent être classés suivant deux catégories basées sur la criticité des caractéristiques temporelles :

- **Contraintes temps réel « dures »** : le système ne doit en aucun cas outrepasser les bornes temporelles fixées sous peine de compromettre la sûreté de fonctionnement du système, voire la sécurité du système embarqué ;
- **Contraintes temps réel « souples »** : le système est autorisé à dépasser les bornes temporelles fixées, dans une certaine limite, sans que ce phénomène ne compromettre la sûreté de fonctionnement et la sécurité du système, mais en fournissant un service dégradé.

Outre les fonctionnalités de plus en plus complexes du système, tous les paramètres présentés caractérisant un système électronique embarqué, doivent être pris en compte lors de la phase de conception du système.

[4] Xeon : processeur basé sur le jeu d'instruction x86, proposé par la firme américaine Intel, dédié au architecture serveur (http://www.intel.com/fr_FR/products/server/processor).

Les activités de recherche de l'équipe SCEE (de Supélec), et, l'une de celles du laboratoire Networking de la division Research & Innovation de Technicolor, au sein desquels ces travaux de thèse se sont conjointement déroulés, est basée les systèmes de radiocommunications. Du fait que ce sont des systèmes électroniques embarqués soient typiques, nous nous sommes appuyés sur deux d'entre eux pour éprouver nos travaux : un système de transmission sans-fil basé sur la technologie MIMO, et un système de radio logicielle. Dans la suite de ce chapitre, nous donnons une définition d'un système de radiocommunication et une définition de la radio logicielle. Nous présentons également les deux systèmes en question que nous retrouverons tout au long de ce document.

1.2 Les systèmes de radiocommunications

En ce début de deuxième décennie du XXI^{ème} siècle, un grand nombre des systèmes de radiocommunications sont des systèmes embarqués temps réels. Les systèmes de radiocommunications naissent à la fin du XIX^{ème} siècle avec l'invention du téléphone filaire (par G. BELL et E. GRAY en 1876), la découverte des ondes radios (par H. HERTZ en 1878) et la réalisation des premières liaisons radios (par G. MARCONI en 1885). Depuis, grâce à l'évolution des techniques du traitement du signal, de la microélectronique, et de l'informatique, les communications sans-fil ont connu un essor formidable pour devenir une des industries marquantes du XX^{ème} siècle.

1.2.1 Définition

Un système de radiocommunication est basé sur l'utilisation d'ondes radioélectriques pour transmettre des informations entre deux points distants. L'onde radio (ou onde radio électrique) est une onde électromagnétique dont la fréquence est inférieur à 3000 GHz (équivalent à une longueur d'onde supérieure à 0.1mm). Comme les ondes ne s'arrêtent pas aux frontières, afin d'éviter toute forme d'anarchie des communications par ondes radios, leur utilisation est régie par l'UIT (*Union Internationale des Télécommunications*). Le spectre des fréquences a été découpé en plusieurs bandes de fréquences dont les noms sont normalisés [6]. Chacune des bandes définie est attribuée à des domaines d'activité en particulier. Par exemple, la première norme de téléphonie mobile GSM (*Global System for Mobile*), le WiFi, ou encore le GPS (*Global Positioning System*), sont situés dans la bande de fréquence UHF (*Ultra High Frequency*).

Il existe aujourd'hui de nombreuses normes de radiocommunications suivant la bande de fréquence et le domaine auquel la communication est destinée. Ces normes définissent non seulement la bande de fréquence utilisable par tout système basé sur celle-ci mais aussi les caractéristiques du signal à transmettre. En effet, par exemple, bien que cohabitant sur la bande de fréquence UHF, une communication par GSM ou WiFi ne se fera pas sur la même fréquence, ni avec le même schéma de modulation. Ainsi que sur toutes les autres bandes de fréquences, la bande UHF est découpée en plusieurs sous-bandes fréquentielles et, chacune de ces sous-bandes est réservée à une utilisation spécifique.

L'information à transmettre subit de nombreuses transformations avant d'être émise pour la rendre plus robuste aux phénomènes suivants :

- L'étroitesse des canaux fréquentiels ;
- Le recouvrement possible entre canaux voisins ;
- La multiplicité des trajectoires de l'onde radio et les autres perturbations dûes à la transmission hertzienne ;
- L'accessibilité multiple à un instant donné à la même fréquence (multi-utilisateur).

La Figure 6 montre une architecture classique dite superhétérodyne [7] d'un récepteur de radiocommunication comprenant un traitement RF (*Radio Fréquence*), un traitement en FI (*Fréquence Intermédiaire*) et un traitement en BB (*Bande de Base*). L'architecture de l'émetteur est symétrique.

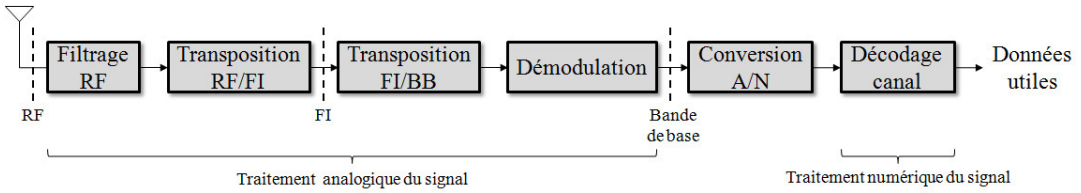


Figure 6. Architecture classique superhétérodyne d'un récepteur par ondes radios.

Pour supporter une telle architecture fonctionnelle la plate-forme matérielle est le plus souvent hétérogène et peu flexible, voir totalement figée lors de la conception du système. En effet, il est souvent nécessaire d'utiliser plusieurs composants pour effectuer l'ensemble des tâches de la chaîne de traitement (plusieurs composants dédiés aux traitements analogiques, et au moins un composant dédié aux traitements numériques). Sur la Figure 6 nous présentons un exemple de réalisation où la conversion A/N a lieu en BB. Par conséquent, tous les traitements exécutés pour convertir le signal de la RF à la BB sont effectués avec des composants analogiques (filtres, amplification, mélangeurs, etc.) et les traitements en BB sont exécutés sur plusieurs unités de traitement numérique (DSP, ASIC ou FPGA). De plus, aujourd'hui, de nombreux systèmes de radiocommunications supportent plusieurs standards de communications, ce qui le plus souvent implique une multiplication des chaînes de traitement du signal et donc une multiplication des composants. Toutefois, aujourd'hui, avec l'évolution de la microélectronique, on peut trouver dans un composant de type SoC des fonctions pour plusieurs standards.

1.2.2 Un système de transmission sans-fil MIMO

Pour le développement de nos travaux sur la modélisation, présentés dans le chapitre 3, et nos travaux sur la synthèse de haut niveau, présentés dans le chapitre 4, nous nous sommes appuyés sur un système de radiocommunication. Nous nous sommes basés sur un système opérationnel développé dans le cadre des activités de recherche de Technicolor sur la transmission sans-fil. Une des activités du laboratoire Networking de la division Research & Innovation de Technicolor traite des communications radio, que ce soit pour le réseau domestique (modules WiFi), pour la télévision ou la radio diffusion

(norme DVB-T/T2/H/NGH), pour les systèmes pour mobiles (3GPP, LTE, WiMAX), ou pour les systèmes propriétaires et les applications professionnelles.

Parmis les études menées sur ces sujets figure l'exploration de techniques de communications radio dites multi-entrées multi-sorties (plus communément utilisées sous le terme anglais MIMO – *Multi-Inputs Multi-Outputs*), c'est à dire disposant de plusieurs antennes d'émission et de réception. On y associe des codages espace temps performants associés à des algorithmes de décodage optimum (algorithmes désignés par le terme ML – *Maximum Likelihood* – en anglais). Des connaissances théoriques ont été acquises sur ce thème, et des ébauches de mises en œuvre matérielles ont été réalisées. Cependant, compte tenu de la complexité des traitements envisagés, il est désormais opportun d'investiguer des méthodes de développements modernes couplant l'efficacité de la synthèse à une capacité de modélisation et de simulation effectuée à haut niveau.

1.2.2.1 Description de la pile protocolaire de communication sans-fil

Aujourd'hui, le laboratoire dispose d'une plate-forme matérielle programmable intégrant l'ensemble des modules d'une pile protocolaire de communication sans-fil basée sur le standard IEEE 802.16 [8] normalisé en 2004 (plus connu du grand public sous la dénomination WiMAX – *Worldwide Interoperability for Microwave Access*). Cette pile [9] intègre des éléments logiciels, exécutés sur des processeurs mis en oeuvre sur FPGA, et des éléments matériels mis en oeuvre sur FPGA. Parmi les éléments matériels mis en oeuvre sur FPGA figurent les fonctions de codage/décodage MIMO.

La chaine de transmission comprend deux cartes de transmissions sans-fil bidirectionnelles (en d'autre terme on peut assimiler une carte à un nœud sans-fil dans un réseau de transmission) et un canal de propagation (ou un émulateur de canal). La transmission est opérationnelle à 2,4 GHz et 5 GHz. La carte de transmission dispose de 4 antennes, chacune associée à un composant RF composé d'un convertisseur CNA (*convertisseur numériques/analogiques*) et d'un CAN (*convertisseur analogique/numérique*). Le cœur de la pile sans-fil est un composant programmable de type FPGA associé à une mémoire externe. La Figure 7 illustre le schéma fonctionnel du nœud sans-fil depuis le réseau Ethernet jusqu'aux antennes d'émission/réception.

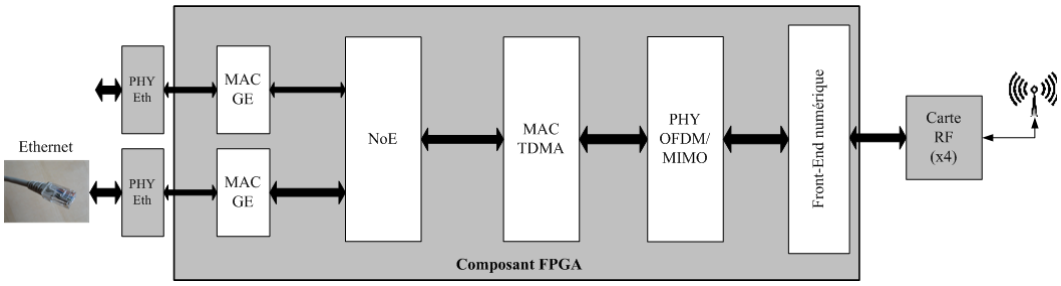


Figure 7. Nœud TxRx d'une transmission sans-fil.

La source que l'on souhaite transmettre par voie hertzienne arrive à la carte par le réseau Ethernet sous forme de paquets IP (*Internet Protocol*). Ces paquets sont aiguillés vers le nœud sans-fil, via le NoE (*Network Engine*). Ce nœud, après avoir ouvert une connexion vers un second nœud sans-fil, vient alimenter cette connexion par ces paquets IP.

Ethernet

Dans le cas de notre transmission sans-fil le réseau Ethernet [10] permet d'acheminer un flux video (dans le cadre des activités du laboratoire Networking) que l'on souhaite transmettre vers la carte de transmission (ou de le récupérer depuis la carte).

Couche physique Ethernet

C'est la couche PHY (*physique*) du modèle d'interconnexion de systèmes ouverts (OSI – *Open Systems Interconnection* – en anglais), extrait de la norme ISO 7498 [11] dédié au protocole Ethernet.

MAC Giga-Ethernet (GE)

C'est une des 2 sous-couches de la couche de liaison de données du modèle OSI.

Network Engine

Le NoE assure les fonctions de routage des paquets IP. Il permet ainsi d'aiguiller vers/depuis le modem radio les paquets qui lui sont destinés

Couche MAC TDMA

La sous-couche MAC (*Medium Access Control*) définie dans le modèle OSI sert à la synchronisation des accès au support physique, la technologie d'accès multiple à répartition dans le temps (TDMA – *Time Division Multiple Access* – en anglais) est utilisée. La technologie TDMA **Erreur ! Source du renvoi introuvable.** est un mode de multiplexage permettant de transmettre plusieurs signaux sur un seul canal fréquentiel. Il s'agit du multiplexage temporel, dont le principe est de découper le temps disponible entre les différentes connexions (utilisateurs) transmises sur une même bande de fréquence.

Couche physique OFDM

La couche PHY assure les fonctions de codage/décodage et modulation/démodulation du signal à transmettre. Une modulation OFDM (*Orthogonal Frequency Division Multiplexing*) est ici employée, ainsi que des solutions MIMO dites de « première génération », c'est-à-dire basée sur des schémas aujourd'hui maîtrisés (schéma d'Alamouti STOBC – *Space Time Orthogonal Block Coding* [13]). La technologie OFDM [14] est une modulation de signaux numériques par répartition en fréquences orthogonales.

Front-end numérique

Le front-end (*FE*) effectue les traitements numériques frontaux, c'est-à-dire respectivement juste avant les CNA, et juste après les CAN. En pratique, cela revient à ajuster la fréquence d'échantillonnage des signaux. Ce module inclut également les modules de pilotage des circuits radio et de conversion AN/NA.

Carte radio fréquence

En mode transmission, ce module permet d'une part de faire la conversion du signal numérique en un signal analogique et, d'autre part, de transposer le signal bande de base analogique obtenu à la fréquence radio souhaitée.

En réception, les opérations duales sont effectuées, à savoir la transposition en bande de base du signal RF reçu et sa conversion en un signal numérique. Un système de réglage de gain (contrôle de gain automatique) est également piloté afin de pouvoir supporter des conditions de propagation variées.

1.2.2.2 Techniques MIMO

Le laboratoire Networking souhaitait basculer les modes de transmission vers des modes MIMO plus performants basés sur un codage espace temps à base de Golden code afin d'exploiter au mieux les capacités multi-antennes de la carte.

Pour cela le système de transmission de données sans-fil dispose de M antennes d'émission et de N antennes de réception comme représentées sur la Figure 8.

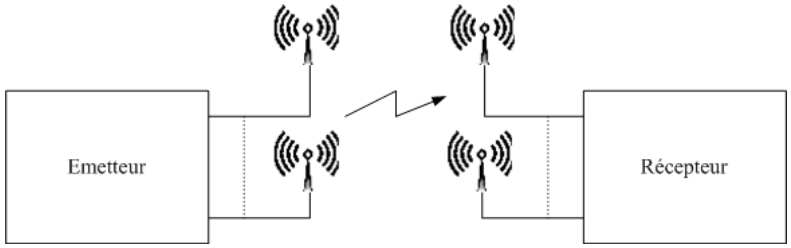


Figure 8. Transmission MIMO M antennes en émission et N antennes en réception.

Un système MIMO peut-être modélisé comme indiqué sur la Figure 9 régie par l'équation suivante :

$$y_N = H_{N\times M}x_N + z_N$$

Où M = nombre d'antenne en émission ;
N = nombre d'antenne en réception ;
H = matrice modélisant le canal ;
x = vecteur du signal en émission ;
y = vecteur du signal en réception ;
z = vecteur modélisant le bruit blanc gaussien.

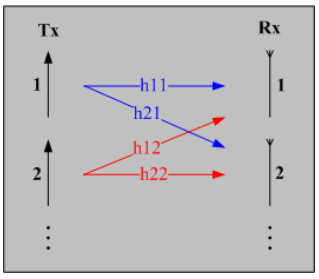


Figure 9. Modèle d'un canal MIMO.

L'implémentation de la technologie MIMO dans une transmission sans-fil de type WiMAX (IEEE 802.16) permet d'augmenter sensiblement l'efficacité spectrale et la robustesse du lien radio.

1.2.2.3 Le décodeur MIMO

Le système étant complexe, nous nous sommes concentrés sur la mise en œuvre du décodage MIMO de la couche PHY du système (l'étude de traitement du signal associée était antérieure à ces travaux de thèse) qui se décompose en deux parties principales : codage et décodage. Cette partie est suffisamment exhaustive pour développer, valider et comparer nos travaux. La Figure 10 présente le synoptique du décodage qui comporte principalement 5 parties : la synchronisation du signal, l'extraction du canal, l'extraction des pilotes, ainsi que l'extraction du signal utile (données à décoder), le décodage SIMO ou MIMO et back-end (désentrelacement, décodage Reed-Solomon, etc.).

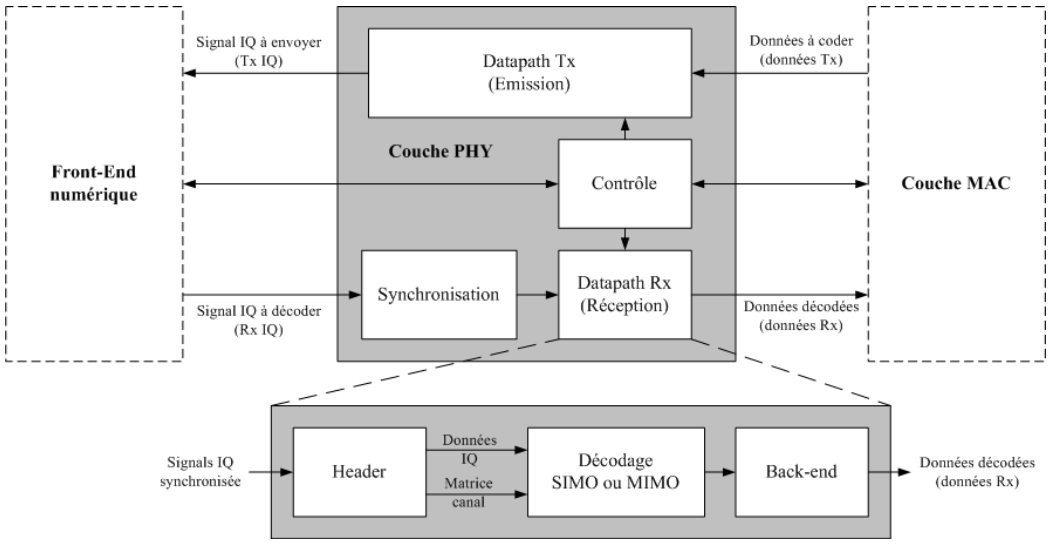


Figure 10. Synoptique de la couche PHY.

La Figure 11 illustre le découpage fonctionnel du décodeur MIMO.

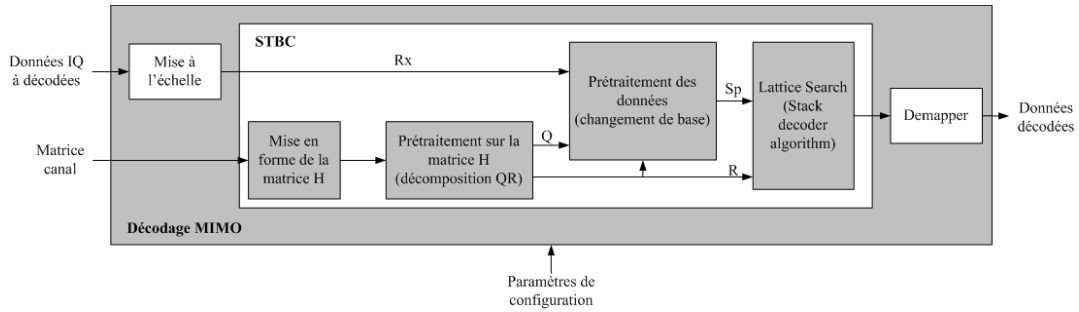


Figure 11. Architecture fonctionnelle du décodeur MIMO.

Prétraitement sur la matrice H

Une décomposition QR de la matrice H (matrice canal) est réalisée, i.e. Q et R sont calculées de telle sorte que H = Q × R, où Q est une matrice orthogonale ($QQ^T = I$), et R une matrice triangulaire supérieure. Il existe plusieurs méthodes pour réaliser cette décomposition :

- La méthode de Householder [15] où la matrice Q est obtenue par produits successifs de matrices orthogonales élémentaires ;
- La méthode de Givens [16] où la matrice Q est obtenue par produits successifs de matrices de rotation plane ;
- La méthode de Gram-Schmidt : cf. Annexe 1.

Chacune d'entre elles a ses avantages et ses inconvénients, en termes de complexité et de performance, ce qui peut donner des résultats sensiblement différents selon la distribution des valeurs propres de la matrice H.

Le code VHDL existant, développé manuellement avant notre étude, applique la méthode de Gram-Schmidt.

Prétraitement des données IQ (Rx sur la Figure 11)

Cette fonction consiste à réaliser un changement de base des données IQ vers une autre base, suite à la décomposition QR : le vecteur complexe Rx est multiplié par la matrice inverse de Q pour obtenir le vecteur projeté Sp, dans lequel les parties réelles et imaginaires ont été dissociées (ceci afin de le rendre compatible avec l'algorithme de lattice search).

Lattice Search [17]

Ce module effectue un décodage optimal (décodage ML) par recherche dans un arbre du point de modulation le plus proche du signal reçu. Cet algorithme est paramétrable : un compromis complexité/performance peut être ajusté.

Le décodage MIMO, traité dans le cadre de nos travaux de thèse, de la pile protocolaire de communication sans-fil, est mise en œuvre sur un FPGA comme le montre la Figure 7, ce qui offre des capacités de souplesse et de flexibilité au système en lui-même. C'est « l'esprit » de la radio logicielle dont nous allons parler maintenant.

1.3 La radio logicielle

Aujourd'hui, l'évolution des TIC (*Technologies de l'information et de la Communication*) conduit à ce que les systèmes de radiocommunications puissent supporter plusieurs standards de communication pour prendre en compte non seulement le volume grandissant de données échangées, leurs types (voie, internet, etc.) mais aussi l'accessibilité du service à tout moment et en tout lieu (mobilité géographique croissante). Par exemple, aujourd'hui, un terminal de téléphonie mobile se doit de supporter plusieurs normes de communication (GSM, EDGE, UMTS, etc.) afin de s'adapter aux réseaux auxquels il se connecte suivant sa localisation géographique. Il doit également proposer d'autres standards de communication tels que le Wifi ou Bluetooth pour l'échange de données. Cependant, le nombre de standards supportés par un système de radiocommunication embarqué est limité. En effet, les différents traitements dans la chaîne (cf. Figure 6) sont le plus souvent exécutés sur plusieurs composants dédiés. De ce fait, pour un seul standard il est parfois nécessaire d'utiliser plusieurs composants. Par conséquent, bien que certaines tâches soient relativement similaires d'un standard à un autre, il y a le plus souvent autant de composants dédiés que de standards. De plus, du fait des caractéristiques des systèmes embarqués (présentées dans la section 1.1 de ce chapitre), le nombre de composants utilisés doit être limité.

Afin d'augmenter sensiblement le nombre de standards supportés par un système embarqué de radiocommunication, de rendre ces derniers plus flexibles, et de diminuer le nombre de composants, Joe MITOLA a proposé la RL (*Radio Logicielle*). En effet, la RL est née de l'idée de proposer un récepteur universel capable de s'adapter à la multitude de standards de radiocommunication utilisés dans le monde pour véhiculer de l'information, quel que soit le type de cette information (voix, vidéo, internet, etc.).

1.3.1 Définition

La radio logicielle (RL)

La RL idéale (du point de vue d'un récepteur) consiste à effectuer la numérisation du signal dès la sortie de l'antenne de réception. Ainsi, tous les traitements de filtrages, de transpositions fréquentielles et les traitements en bande de base sont effectués par traitements logiciels et non plus par des composants dédiés. La flexibilité d'une plate-forme logicielle (processeur généraliste) par rapport à une approche utilisant historiquement des composants analogiques dédiés procure de grands avantages. Ainsi le traitement logiciel de la totalité de la chaîne de réception permet de basculer plus facilement d'un standard à un autre en chargeant le logiciel nécessaire.

Le premier système RL opérationnel fut présenté en 1994 dans le cadre du projet de recherche militaire SpeakEasy [18]. Les militaires américains avaient besoin de pallier au problème posé par la diversité des systèmes de radiocommunications employés par différents corps d'armée. Le résultat fut un succès sur le plan fonctionnel mais très loin de l'encombrement d'un terminal mobile (pour un coût faramineux).

L'introduction de la RL dans la recherche civile dans le domaine des systèmes de télécommunications est à l'initiative de J. MITOLA [19][20]. Les résultats des travaux de J. MITOLA ont donné lieu dans les années 90 à de nombreuses initiatives dont la création du forum MMITS (*Multifunction Information Transfer System*), devenu SDR forum (*Software Defined Radio*) en 1999 et renommé Wireless Innovation Forum depuis 2010 [21]. Ce forum regroupe de nombreux acteurs civils et militaires dans le domaine de la RL. Une autre initiative a donné lieu au projet GNU radio [22], basé sur le concept de logiciel libre, qui propose un ensemble d'outils et de librairies (écrite en C++ et en Python) destinés à la radio logicielle. Enfin, l'une des plus importantes initiatives dans le domaine de la RL est le SCA (*Software Communication Architecture*) [23]. Issu de travaux de recherche des militaires américains (projet JTRS – *Joint Tactical Radio System*), le SCA a pour but d'aider au déploiment de systèmes RL et définit une architecture qui abstrait la plate-forme matérielle pour l'application logicielle (radio) du système RL. L'évolution et la promotion du SCA sont aujourd'hui discutés au Wireless Innovation Forum. Le SCA est aujourd'hui non seulement un standard nécessaire dans les systèmes de radiocommunications destinés à l'armée américaine, mais tente également de s'implanter dans le domaine des applications civiles. Nous pouvons ajouter que les européens travaillent également à un concept similaire au SCA pour faire cohabiter les futurs systèmes de radiocommunications des armées européennes [24][25].

La RL idéale serait capable de numériser une bande fréquentielle infinie. La Figure 12 met en évidence l'architecture fonctionnelle d'une RL réalisable dans certains contextes d'application plus restreints (sous-bande fréquentielle). Contrairement à une architecture superhétérodyne (cf. Figure 6) l'ensemble des traitements appliqués sur le signal réceptionné est réalisé dans le monde numérique. Cela permet de bénéficier de la flexibilité du traitement numérique totalement mise en œuvre en logiciel, supporté par une plate-forme matérielle composée de processeurs.

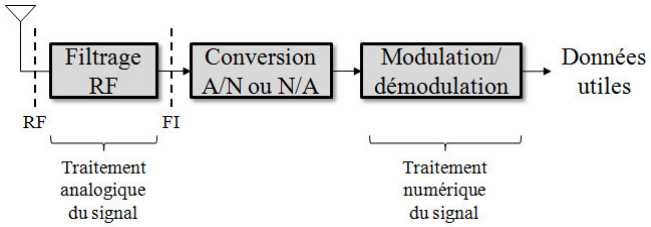


Figure 12. Architecture réalisable d'une radio logicielle.

Toutefois, il existe encore aujourd'hui de nombreux challenges à relever (notamment au niveau du traitement RF), avant de pouvoir prétendre proposer des systèmes RL idéaux pour tous les cas d'application (quels que soient la fréquence porteuse, la bande passante, le coût, la consommation, etc.). Afin de réduire la marche entre un système de radiocommunication superhétérodyne et un système RL, une étape intermédiaire est apparue : la RLR (*Radio Logicielle Restreinte* ou SDR – *Software Defined Radio*).

La radio logicielle restreinte (RLR)

Contrairement à la RL, dans un système RLR la numérisation du signal n'est pas effectuée directement en sortie de l'antenne sur une bande infinie mais, par exemple, une fois le signal transposé en FI, voire en bande de base, après filtrage analogique RF.

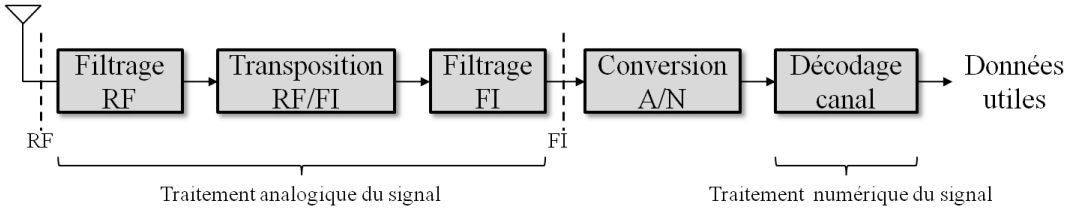


Figure 13. Architecture fonctionnelle typique d'un système de radio logicielle restreinte (RLR).

La Figure 13 illustre l'architecture d'un système RLR. Contrairement à une architecture RL, le traitement analogique du signal réceptionné n'est pas réduit à son plus simple élément. Une plate-forme matérielle d'un système RLR est donc nécessairement constituée de composants dédiés (ASICs) au traitement analogiques. De plus contrairement à la RL idéale, le traitement numérique du signal n'est pas totalement effectué par traitement logiciel mais conjointement par traitement logiciel et matériel. Un système RLR peut se caractériser par exemple par une plate-forme matérielle configurable composée de composants reprogrammables (processeurs) et/ou reconfigurables (FPGA), ce qui offre une certaine flexibilité au système pour basculer d'un mode de communication à un autre. Les traitements numériques du signal des différents modes ou standards sont effectués sur le(s) même(s) composant(s) programmable(s)/configurable(s) au lieu de composants dédiés (ASIC) pour chacun des standards, comme c'est le cas pour un système de radiocommunications classique.

La Figure 14 met en évidence l'évolution entre une architecture classique superhétérodyne et une architecture RLR d'un système de radiocommunication multistandards.

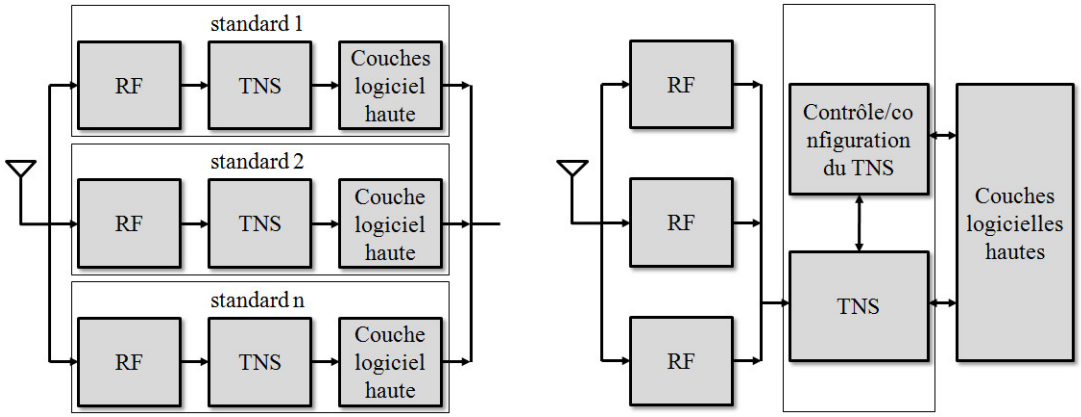


Figure 14. Architectures juxtaposées et RLR d'un système de radiocommunications multistandards [26].

La principale évolution de la RLR, par rapport à un système radiocommunication classique, vient de la partie TNS (*Traitement Numérique du Signal*) qui est reconfiguré selon le standard utilisé à un instant donné, et en un lieu donné. Pour contrôler et configurer la chaîne de TNS, un module de gestion commun aux différents standards est nécessaire. Celui-ci est le plus souvent exécuté sur un processeur, alors que les différentes tâches de la chaîne de TNS sont effectuées sur un(des) processeur(s) dédié(s) au traitement du signal (DSP) et/ou des composants reconfigurables en temps réel FPGA (la technologie dite de reconfiguration dynamique partielle de FPGA est ici prônée [26]). L'une des difficultés de la mise en œuvre d'un système RLR est de gérer la reconfiguration de la chaîne de traitement du signal sans rompre le service.

La radio intelligente (RI)

Le concept RL pose les bases d'un système de radiocommunication plus flexible (souplesse de basculement d'un mode de communication à un autre accru). Au-delà de la RL, les chercheurs pensent que les futurs systèmes de radiocommunications devront être en outre intelligents. En effet, les acteurs du monde de la radiocommunication envisagent des systèmes capables d'apprendre de leur environnement pour adapter leur fonctionnement : c'est la RI (Radio Intelligente). Le système sera capable d'analyser une multitude de métriques extraites de stimuli issus de son environnement, mais aussi extraites de son état (niveau de batterie, dysfonctionnement d'un composant, etc.), analyse qui permettra au système de prendre des décisions afin d'adapter son fonctionnement en reconfigurant la chaîne de TNS. Pour cela, une architecture dédiée à la gestion de l'intelligence (prise de décision) et à la gestion de la reconfiguration est nécessaire [27][28].

Le livre [29], dirigé par J. PALICOT, donne une définition plus détaillée de la RI et présente plus en profondeur tous les enjeux et challenges à relever autour de la RL. De plus, les auteurs dressent un état des lieux de ce qui se fait aujourd'hui et le chemin qu'il reste encore à parcourir pour aboutir un système RL idéal tel que ce concept a été défini à sa naissance.

Au-delà des caractéristiques des systèmes embarqués temps réel, les concepts de RL, RLR et RI engendrent par conséquent de nouveaux challenges (support de multistandard plus flexible : contrôleur multistandard, gestionnaire de reconfiguration, plate-forme reconfigurable, etc.) à prendre en compte lors de la conception du système de radiocommunication. C'est dans ce contexte que cette thèse se positionne et va apporter quelques contributions.

1.3.2 Un système radio logicielle restreinte

Pour illustrer nos travaux sur la modélisation de systèmes reconfigurables, présentés dans le chapitre 5 de ce document, nous utiliserons un système radio logiciel restreint. Nous avons décidé, pour ce contexte, d'utiliser un autre cas d'étude que clui du système de transmission sans-fil MIMO qui sera utilisé pour l'étude de la modélisation et de la génération de code des chapitres 3 et 4. Ce choix se justifie, d'une part, car la complexité du système n'est pas nécessaire pour la preuve de concept visée, et d'autre part, cela aurait nécessité de mettre en œuvre la reconfiguration dynamique partielle sur la plate-forme (aspect sur lequel nous reviendrons dans le chapitre 5), ce qui n'était pas dans l'objectif de ces travaux de thèse.

1.3.2.1 Une application configurable

La Figure 15 illustre le découpage fonctionnel de l'application RLR simplifiée. Le terme simplifié est ici employé car la chaîne de TNS effectuée en bande de base ne comporte pas l'ensemble des éléments mis en place dans le cadre d'une chaîne de transmission typique (codeur, entrelaceur, transposition fréquentielle, etc.).

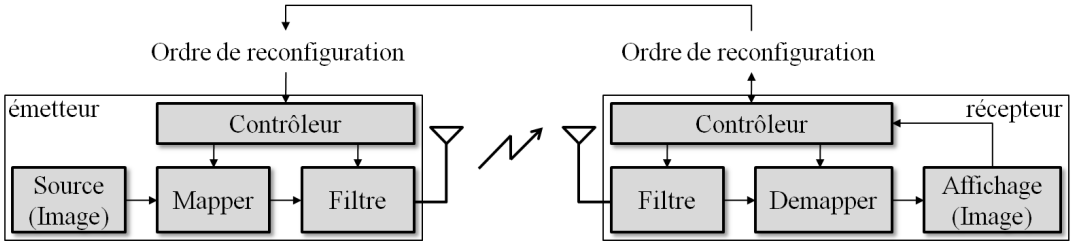


Figure 15. Synoptique du système de transmission sans-fil RLR simplifié.

Le système doit transmettre une application vidéo sur un canal de transmission radio. L'émetteur et le récepteur sont constitués de quatre modules, dont deux modules de traitement, et un module de contrôle.

Source

Le module *Source* lit un fichier image (au format RVB) et sérialise les données de celui-ci ligne par ligne. Chacune d'entres elle étant envoyée sous forme de paquets, paquets constitués d'un entête de début de ligne et des données utiles. Chaque pixel de l'image est codé en RVB sur 8 bits par couleurs. Le paquet de la dernière ligne dispose en plus d'un entête, d'une fin de fichier utilisée pour indiquer la fin de l'image. L'entête est connu par le récepteur et permet la synchronisation.

Mapper

Ce module permet de réaliser la transformation des informations binaires en symboles selon le format de modulation choisi. Le système offre la possibilité de 3 types de modulations sur porteuse en quadrature :

- BPSK où 2 bits sont nécessaires pour former un symbole ;
- QPSK (ou 4-QAM) où 2 bits sont nécessaires pour former un symbole ;
- 16-QAM où 4 bits sont nécessaires pour former un symbole.

Ce module est donc reconfigurable.

Note : Les symboles générés sont représentés dans le corps des Complexes z = I+jQ, sous forme d'une constellation IQ (I : partie réelle et Q : partie imaginaire).

Filtre

Ce module est un filtre numérique de type RIF (*Réponse Impulsionnelle Finie* ou FIR en anglais) qui réalise une mise en forme du signal. Ce filtre dispose de deux jeux de coefficient différents : un dont le roll-off[5] est de 0,15 et le second dont le roll-off est de 0,22. Comme le module précédent, le filtre est donc reconfigurable.

Contrôleur de l'émetteur

Ce module permet de gérer la reconfiguration des modules de la chaîne de TNS (*Mapper* et/ou du *Filtre*), reconfiguration effectuée lorsque ce module reçoit, soit un ordre de reconfiguration depuis l'extérieur (utilisateur), soit un ordre de reconfiguration décidé par lui-même à la suite d'analyses de métriques (provenant du récepteur). Nous reviendrons un peu plus en détails sur ce module dans le chapitre 5, chapitre dédié à la modélisation de la reconfiguration. Nous présenterons notamment une architecture qui permet de prendre des décisions de reconfiguration de la chaîne de TNS et de gérer la reconfiguration des modules de la chaîne de TNS, reconfiguration pouvant être logicielle et/ou matérielle.

Le récepteur est le symétrique de l'émetteur. Nous pouvons toutefois noter des différences au niveau du module de contrôle et du module d'affichage.

Affichage

Ce module permet d'afficher ligne par ligne l'image issue des données réceptionnées. De plus, par le biais d'une corrélation, ce module relève le TEB (*Taux d'Erreur Binaire*) du signal reçu. Cette information est transmise au contrôleur pour qu'il prenne des décisions de reconfiguration.

Contrôleur en réception

Ce module diffère de la version de l'émetteur. En effet, ce module dispose d'une partie dite intelligente conforme à une philosophie de radio intelligente. Celui-ci récupère la métrique TEB, capturée par le module d'affichage, et l'analyse. En fonction du niveau de TEB, une décision de reconfiguration est prise. L'ordre de reconfiguration est non seulement destiné à la chaîne de TNS du récepteur mais aussi à l'émetteur. L'analyse, la prise de décision et la gestion de la reconfiguration (logicielle et/ou matérielle) sont assurés par une architecture dédiée présentée dans le chapitre 5.

1.3.2.2 Une plate-forme matérielle reconfigurable

La plate-forme matérielle supportant l'application RLR est une carte de prototypage ML506 [30] de Xilinx composée d'un FPGA Virtex-5 supportant la technologie dite de reconfiguration dynamique partielle, technologie qui permet de reconfigurer une partie du composant sans interrompre le service du système. L'équipe SCEE de Supélec est reconnue pour ses travaux de recherche dans ce domaine [31][32].

[5] Roll-off : le roll-off d'un filtre est la pente du carré du gain dans un diagramme de Bode.

1.4 Conclusion

La présentation des principales caractéristiques d'un système électronique embarqué abordées dans cette thèse et la présentation d'un système de radiocommunication, permettent de mettre en évidence la complexité d'un tel système. Ainsi, de plus en plus de paramètres sont à prendre en compte lors de la phase de conception des systèmes électroniques embarqués, ce qui ne fait qu'augmenter le challenge à relever lors de cette étape.

Le manque de flexibilité des plate-formes d'exécution des systèmes de radiocommunication, couplé au nombre de tâches que doit pouvoir exécuter un tel système et à la multiplication des standards de communication à prendre en considération a poussé les acteurs du monde des télécoms à proposer une architecture offrant plus de flexibilité pour basculer d'un standard à un autre. La radio logicielle prône la numérisation du signal dès la sortie de l'antenne. Il en découle que la chaîne de traitement numérique du signal est effectuée de manière logicielle et non plus massivement effectuée par des composants matériels dédiés. La radio logicielle ou la radio logicielle restreinte (variante basée sur une plate-forme hétérogène composée de processeurs et de composants reprogrammables FPGA) offre certes plus de flexibilité, mais en contrepartie la conception et la gestion de la reconfiguration de la chaîne de traitement du signal est beaucoup plus complexe à mettre en œuvre que dans le cas d'un système de radiocommunication classique.

Nous mettons en évidence dans le chapitre suivant que les méthodologies de co-conception logicielle/matérielle actuellement utilisées pour le développement des systèmes électroniques embarqués, ne sont plus adaptées, au développement de systèmes RL. Il est donc nécessaire de proposer et mettre œuvre de nouvelles méthodologies adaptées, ce qui est l'objet de cette thèse.

Chapitre 2 Un besoin de nouvelles méthodologies

Sommaire

L'environnement de développement des systèmes électroniques embarqués ne cesse de se complexifier depuis l'avènement du transistor. De plus, l'évolution de ces dernières années laisse présager une amplification significative de ce phénomène dans les années à venir (cf. Introduction et [4]).

Dans ce chapitre, nous exposons les limites des méthodologies de co-conception actuellement mises en œuvre pour élaborer des systèmes électroniques embarqués. Ensuite, nous présentons une des solutions prônées par la communauté de l'ESL (*Electronic System Level*) : la conception dirigée par les modèles. Enfin, nous dressons un état des lieux des différentes méthodologies de co-conception proposées pour améliorer la productivité.

2.1 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'impose 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 [33], les auteurs présentent en quoi consiste la conception des systèmes embarqués et mettent en évidence les challenges à relever.

2.1.1 Les méthodologies de co-conception

Depuis maintenant 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 16 illustre un flot de conception de SoC/SoPC dit « classique » ou « traditionnel », mature et majoritairement utilisé par les industriels.

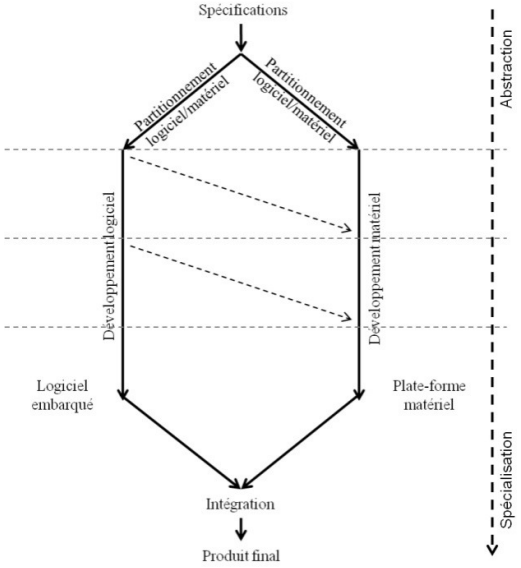


Figure 16. Flot de co-conception logicielle/matérielle depuis les spécifications jusqu'au système final.

L'approche préconise un choix architectural du système très en amont dans le flot de développement afin de partitionner le logiciel embarqué et la plate-forme d'exécution. A partir de cette étape, deux flots de développement distincts sont parallèlement utilisés jusqu'à l'intégration. Ces deux flots utilisent des environnements de développement différents (langages de programmation, outils, processus), ce qui rend difficile la communication et la compréhension des préoccupations des équipes de développement logiciel et matériel. Bien que des points de synchronisation existent, des erreurs surviennent souvent trop tard dans le processus de développement. A ce stade, il est souvent complexe de revoir l'architecture du système. Une erreur est d'autant plus coûteuse qu'elle est détectée tardivement dans le processus de développement. De plus, pour les mêmes raisons, la phase d'intégration est un point critique dans le flot de développement.

Les équipes de développement, tant dans le domaine du génie logiciel, que de celui du développement matériel, disposent, bien entendu, aujourd'hui d'outils d'aide à la conception et à l'implémentation, tels que les outils de l'EDA (*Electronic Design Automation*). De plus, la réutilisation de briques fait partie intégrante de la culture du développement. Cependant, l'implémentation reste majoritairement manuelle : ce qui est aujourd'hui un frein de plus en plus mis en évidence.

Au-delà des limites évidentes exposées précédemment, l'insuffisance, voire l'absence de capitalisation du domaine métier et des éléments développés, est également un manque de la méthodologie « traditionnelle ». Il semble aujourd'hui primordial de combler cette carence.

Arrivés à maturité les environnements de co-conception « traditionnels » sont loin d'avoir résolus toutes les challenges imposés par le développement de systèmes complexes, tels que les systèmes de télécommunications. En effet, aujourd'hui les limites des méthodologies de co-conception sont de plus en plus critiques et il est de plus en plus difficile pour les équipes de conception de relever les défis.

2.1.2 Les challenges à relever

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

2.1.2.1 La communication

Un des challenges, qui n'est en lui-même pas nouveau, est le problème de communication entre les êtres humains, d'origines et de cultures différentes. Cette difficulté de communication induit un problème de compréhension. Dans le contexte de la conception d'un système électronique embarqué, le problème de communication et de compréhension entre les différents corps de métiers intervenant tout au long du processus ne cesse de prendre de l'ampleur car les systèmes sont de plus en plus hétérogènes. Ce problème de communication est dû à de nombreux paramètres : domaine métier (marketing, système, développeur logiciel, développeur matérielle, intégrateur), langages, outillages.

2.1.2.2 L'évolution technologique

Les technologies disponibles pour concevoir des systèmes électroniques embarqués ne cessent d'évoluer à grande vitesse. Les équipes de conception doivent sans cesse appréhender ces évolutions. Cependant, comme le dit le proverbe : « le temps, c'est de l'argent ». Effectivement, nous accordons de moins en moins de temps aux équipes de conception pour maitriser une technologie avant l'arrivée de sa remplaçante.

2.1.2.3 La complexité

Du fait de l'évolution technologique, mais aussi de la demande et de l'exigence du marché, les systèmes embarqués intègrent de plus en plus de fonctionnalités. Pour réaliser ces systèmes, de nombreuses ressources sont nécessaires. L'hétérogénéité des technologies utilisées nécessite une très bonne maîtrise de celles-ci afin de les faire

cohabiter dans un même système : ce défi est aujourd'hui toujours plus complexe à relever.

2.1.2.4 Le temps de mise sur le marché

Dans un monde où la consommation est effrénée et la concurrence toujours plus féroce, le temps est un des éléments capital à prendre en compte dans la réalisation d'un produit. En effet, dès l'apparition de nouvelles normes (technologique et/ou tendancielle), les équipes du marketing ne donnent que peu de temps aux équipes de conception pour développer un nouveau système. Tout ceci se fait dans un souci de mettre sur le marché le dit système avant la concurrence et ainsi avoir une longueur d'avance. Cette contrainte ne laisse pas beaucoup de temps à la compréhension, et la réflexion. C'est pourquoi il est primordial de proposer aux équipes de conception de nouvelles méthodologies et de nouveaux outils afin qu'elles puissent se concentrer sur la valeur ajoutée du système (architecture du système, innovation fonctionnelle, innovation technologie) plutôt que sur l'implémentation en elle-même, sans toutefois négliger cette dernière.

2.1.2.5 Le coût de conception

Outre le temps, le coût de conception d'un système est un élément décisif dans un monde très concurrentiel, et tout particulièrement dans les télécommunications. Plus le coût de conception sera contrôlé et optimisé, plus le système sera attrayant : prix attractifs et marge sont les termes qui semblent aujourd'hui de plus en plus diriger la vie d'un système. Ce coût de développement dépend non seulement du temps de conception, du coût humain (souvent plus élevé pour des ingénieurs matériel que pour des ingénieurs logiciel), mais également du coût engendré par la méthodologie en elle-même et par les outils utilisés par les équipes. Par exemple, l'utilisation d'une méthodologie unifiée et commune tout au long du processus de développement du système permettrait sans aucun doute de faciliter le dialogue entre les différentes équipes et ainsi accroître la productivité. Ceci permettrait donc de réduire le temps passé à la communication.

Le terme **productivité** est clairement ici le point central de tous ces défis. Il semble établi qu'il est nécessaire d'améliorer cette productivité de manière significative pour répondre à tous ces challenges. Selon l'ITRS qui regroupe notamment les grands noms du marché des semi-conducteurs, les principaux points à prendre en compte pour améliorer la productivité sont :

- La réutilisation des ressources ;
- La vérification et le test ;
- La plate-forme de conception ;
- L'implémentation fiable ;
- Le processus de conception ;
- Et la pérennité des outils.

2.1.3 Des solutions ESL

La communauté de l'ESL propose, depuis plusieurs années, des solutions aux équipes de conception pour répondre aux challenges présentés précédemment. Nous pouvons notamment citer les techniques suivantes :

- La réutilisation d'IP (*Intellectual Property*) [34] : technique qui consiste à concevoir un système en utilisant un maximum de composants (logiciels et matériels) déjà existants et en les intégrant. Cette approche suppose une compatibilité des interfaces telle que le propose le standard IP-XACT [35] ;
- La synthèse comportementale ou synthèse de haut niveau [36] : technique qui permet de générer une implémentation HDL d'un modèle fonctionnel décrit avec un langage tel que le C ou le C++ ;
- Platform-based design [37] : technique qui consiste à programmer une plate-forme générique selon les spécificités de l'application ; plate-forme générique basée sur des composants programmables et reprogrammables (processeurs, FPGA, DSP, etc.).

Dans [38], les auteurs proposent une classification basée sur la représentation en Y des outils industriels et académiques de la conception ESL.

Bien que de nombreux efforts soient réalisés, comme le montre les techniques présentées ci-dessus, il semble de plus en plus évident pour les différents acteurs de l'EDA et de l'ESL, et plus généralement pour les acteurs du monde des systèmes électroniques embarqués, que pour prendre en compte tous les challenges, il faut tendre vers un environnement de co-conception unifié tout au long du processus de développement. Afin de répondre à toutes ces attentes, plusieurs approches ont vu le jour, notamment des approches basées sur le concept de modèle. L'une de celles-ci est présentée dans ce document.

2.2 L'architecture dirigée par les modèles

Avant de parler de modélisation et de méthodologie de conception basée sur les modèles, il nous semble nécessaire de présenter les concepts sous-jacents.

2.2.1 Les éléments de base de l'IDM

Dans le domaine de l'ingénierie, la réalisation d'un système, assimilable à un objet quelqu'en soit sa complexité, passe par une bonne compréhension de celui-ci à toutes les étapes du cycle de conception. Pour une meilleure compréhension du système, une représentation de celui-ci est effectuée à chaque étape du développement du système. Cette représentation est plus ou moins abstraite et simplifiée : nous parlons de modélisation. En fonction des besoins de chaque équipe qui intervienne tout au long du cycle de conception, un modèle représente certaines caractéristiques du système. Ce modèle fait abstraction des éléments d'implémentation. En effet, les équipes du marketing n'ont pas les mêmes besoins de connaissances du système que les équipes systèmes ou que les équipes de développement de la plate-forme matérielle. L'une des approches qui met en œuvre la modélisation d'un système est l'IDM (*Ingénierie Dirigée par les*

Modèles) [39] utilisée dans le domaine du développement logiciel. Cette approche est dédiée au domaine du logiciel et ne peut s'appliquer directement dans le cadre d'un développement de co-conception.

Sous l'impulsion de l'OMG (*Object Management Group*), l'approche MDA (*Model Driven Architecture ou Architecture Dirigée par les modèles*) [40], dérivée de l'IDM, est apparue plus adaptée à la co-conception. L'approche MDA utilise les mêmes éléments de base que l'IDM. Avant de présenter les concepts de l'approche de modélisation préconisés par le MDA il est important de définir les éléments de base sur lesquels s'appuie cette approche :

- le méta-modèle ;
- le modèle ;
- la transformation de modèle.

2.2.1.1 Le modèle

Un modèle correspond à une représentation abstraite de la réalité. De plus, le modèle permet une représentation textuelle et/ou graphique formalisée ou non. Dans le cadre de l'IDM et du MDA le modèle se base sur des concepts formalisés (nous le verrons par la suite). Cependant, le concept de modèle n'est pas nouveau en soi. Comme le précise Jean-Marie FAVRE, archéologue à l'Université Joseph Fourrier de Grenoble, ce concept de modèle n'est pas nouveau. Les personnes à l'origine de l'IDM n'ont pas réinventé la roue, ils n'ont fait que remettre au goût du jour un concept qui remonte à plusieurs millénaires, au temps de l'alphabet ougaritique cunéiforme [41]. Toutefois, le modèle est ici défini et appliqué au domaine de l'ingénierie.

2.2.1.2 Le méta-modèle

Comme pour écrire un texte, le français est régi par une grammaire, il faut utiliser un langage de modélisation pour décrire un modèle dont les concepts et la sémantique sont définis par une grammaire. Cette dernière est appelée dans le monde de la modélisation le méta-modèle. Partant de ce fait, tout modèle doit être conforme à un méta-modèle. De plus l'IDM préconise de modéliser un système à différents niveaux d'abstraction selon le besoin et les équipes à qui celui-ci est destiné. Il va de soi qu'un modèle manipulé par les équipes systèmes ne sera pas aussi détaillé qu'un modèle du même système manipulé par les équipes de développement logiciel et matériel. En effet, ces derniers n'ont pas les mêmes préoccupations. Il n'y a pas clairement de niveau d'abstraction défini, toutefois on retrouve régulièrement dans la littérature 4 couches d'abstraction représentées par la Figure 17.

Le niveau M0 représente la réalité, le système réel physique. Ce dernier est représenté par un premier niveau d'abstraite M1 qui est le modèle. Le modèle du système réel utilise une syntaxe pour modéliser celui-ci. Cette syntaxe est le méta-modèle au niveau M2. Le méta-modèle est lui-même un modèle qui doit être conforme à une syntaxe. Le niveau M3, intitulé méta-méta-modèle, est le langage qui permet de décrire le méta-modèle. Il pourrait en être ainsi de manière infinie mais dans la pratique il est considéré qu'un méta-méta-modèle est conforme à lui-même.

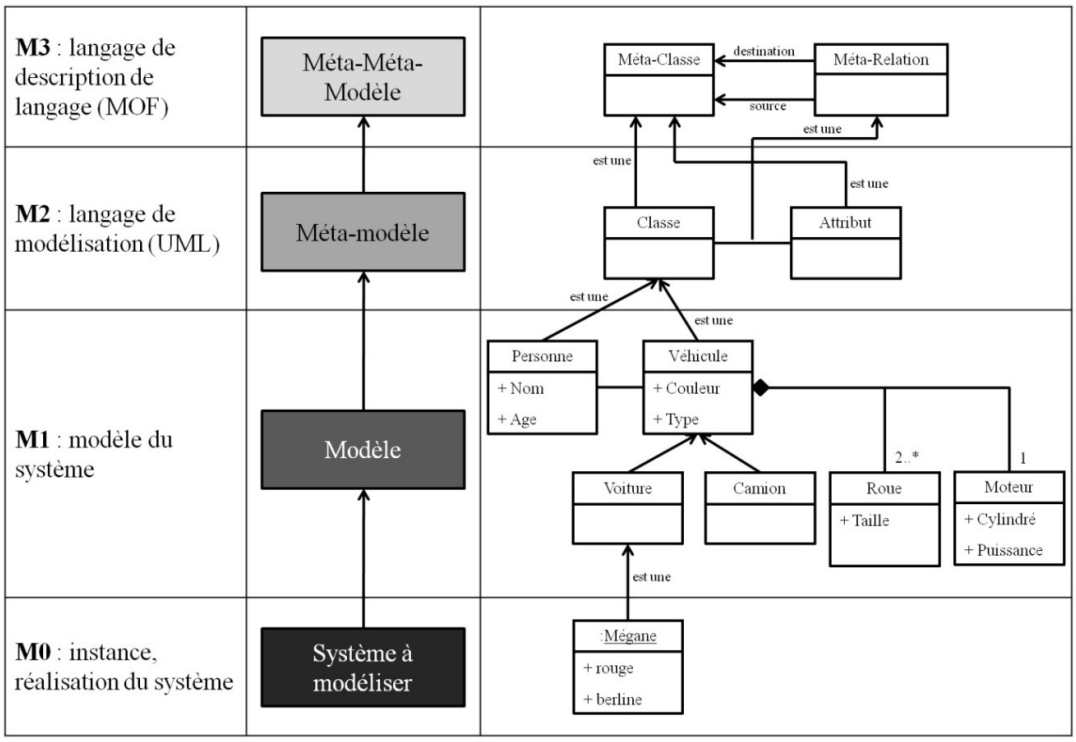


Figure 17. Représentation des couches de méta-modélisation.

- **M0** : une Renault Mégane est un objet physique réel ;
- **M1** : une représentation abstraite de cet objet physique réel est décrite par exemple par un modèle UML ;
- **M2** : le langage de modélisation UML est régi par un méta-modèle qui définit la sémantique des différents éléments de ce langage tels que les classes, les attributs, les relations, etc ;
- **M3** : niveau de représentation le plus élevé, le méta-méta-modèle est le langage de définition de la sémantique UML. La classe et l'attribut sont des méta-entités utilisant des méta-relations. Le méta-méta-modèle s'auto-définit. Nous pouvons citer MOF (*Meta Object Facility*) [42] ou encore Ecore (EMF core) qui définissent les concepts de classe, d'objets, d'héritage, etc.

2.2.1.3 La transformation

L'approche MDA, et plus généralement l'IDM, fait la part belle à la transformation d'un modèle vers un autre modèle de niveau d'abstraction équivalent ou non. Chacun des deux modèles (le modèle source et le modèle destinataire) est conforme à son méta-modèle respectif. Pour basculer d'une représentation d'un objet vers une autre représentation de ce même objet, le mécanisme passe par l'écriture d'un jeu de règles qui définit la transformation (cf. Figure 18). Ce jeu de règles précise le cadre de la transformation notamment en précisant comment un élément du méta-modèle source sera traduit en un élément du méta-modèle destinataire.

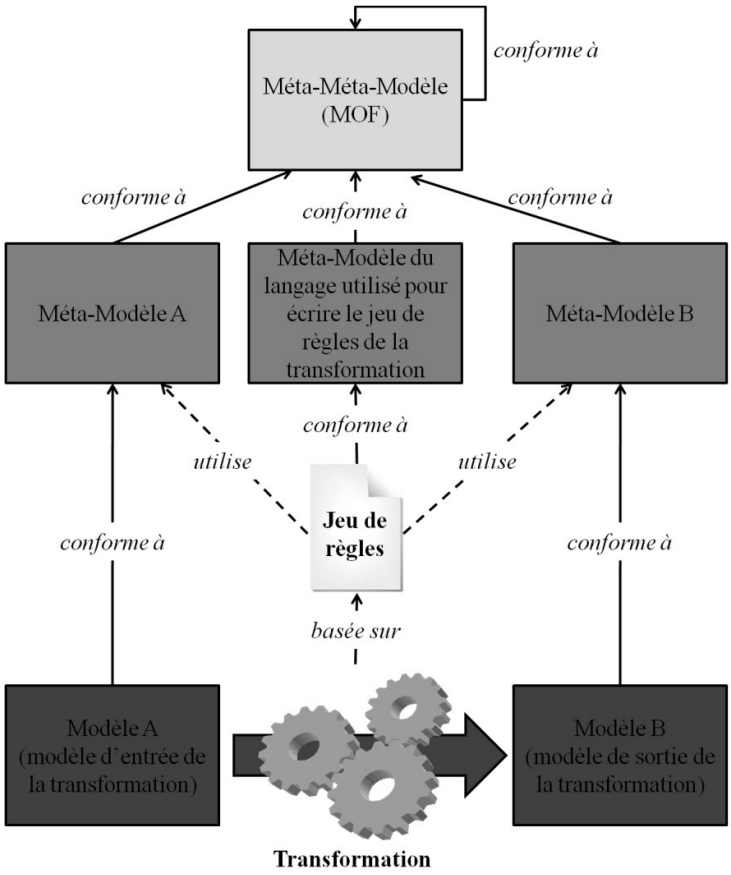


Figure 18. Eléments mis en jeu dans le mécanisme de transformation d'un modèle source en un modèle destination.

Plusieurs langages existent pour décrire ces jeux de règles. Nous pouvons notamment citer Kermeta [43], utilisé dans nos travaux comme nous le verrons dans la suite de ce document. Nous pouvons également mentionner l'ATL (*Atlas Transformation Language*) [44] ou encore l'EMF (*Eclipse Modeling Framework*) [45]. Cependant seul le langage QVT (*Query Views Transformation*) [46] est un standard de l'OMG, ce qui ne fait pas de lui le candidat le plus utilisé pour autant.

2.2.2 Les éléments de base du MDA

L'approche MDA [40] développée et standardisée par l'OMG propose un cadre concret à la mise en œuvre de l'IDM pour la modélisation de systèmes complexes dans le domaine du génie logiciel, sans toutefois cantonner l'approche MDA à ce domaine. Il est impératif de comprendre que MDA n'est pas une méthodologie de conception mais uniquement un cadre générique pour mettre en application l'utilisation de modèle dans un processus de développement d'un système. La Figure 19, extraite de [47], illustre les différentes couches sur lesquelles s'appuie le MDA.

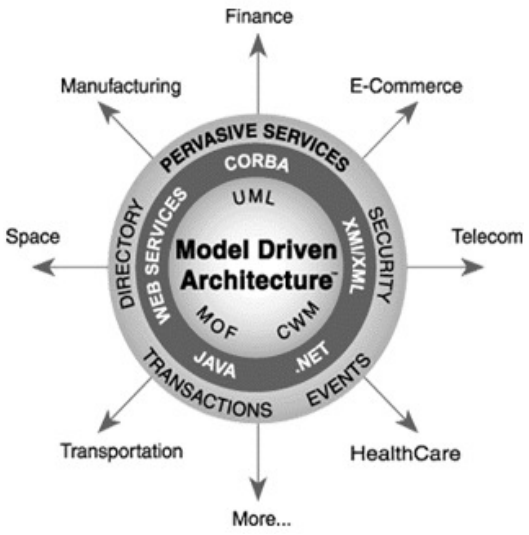


Figure 19. Représentation en couche de l'approche MDA, issue du standard de l'OMG [47].

Comme le montre la Figure 19 le MDA se base sur trois piliers :

- Le MOF [42] : méta-méta-modèle, langage qui s'auto-définit et qui définit les éléments de grammaire de base pour définir des langages de modélisation ;
- Le CWM (*Common Warehouse Metamodel*) [48] : spécification de l'OMG qui décrit un langage d'échanges de méta-données. Ce langage est défini à l'aide du MOF ;
- L'UML (*Unified Modeling Language*) [49] : langage de modélisation unifié défini à l'aide du MOF.

L'approche MDA, qui reprend la représentation en Y des modèles, propose deux concepts pour permettre de formaliser et d'automatiser le processus de développement :

- **Séparation des intérêts** : préconise de modéliser l'application indépendamment de la cible matérielle par laquelle elle sera supportée ;
- **Transformation de modèle** : qui permet, d'un côté, de passer d'une représentation d'un modèle à une autre représentation de ce même modèle (ou pour raffiner un modèle) et, d'un autre côté, de générer du code exécutable.

De plus, l'approche MDA met en avant l'interopérabilité des outils de modélisation entre eux, ainsi que des modèles développés. Comme le prônent les acteurs de l'ESL, l'approche MDA reprend l'idée de réutilisation des modèles. Théoriquement, l'interopérabilité et la réutilisation sont d'autant plus aisées si les modèles sont développés à l'aide d'un langage standardisé tel que l'UML.

Nous reviendrons plus en détails dans la suite de ce chapitre sur la définition des modèles MDA. Dans un premier temps, nous présentons plus en détails l'UML, élément fortement couplé à l'IDM et de ce fait à MDA.

2.2.3 UML : un langage de modélisation orienté objet

L'UML (*Unified Modeling Language*) [49] est un langage de modélisation unifié généraliste par opposition aux langages dédiés (DSL en anglais) tels que peuvent l'être le VHDL, le SystemVerilog pour modéliser l'implémentation matérielle ou encore l'AADL (*Architecture Analysis and Design Language*) [50], standardisé par le SAE, utilisé pour modéliser des systèmes avioniques notamment. L'UML, issu de plusieurs propositions précédentes (Boosh [51], OMT – *Object Modeling Technique* – [52] et OOSE – *Object Oriented Software Engineering* – [53]), permet de représenter un système avec un certain niveau d'abstraction. Le méta-modèle UML définit la sémantique du langage basée sur le concept d'objet, concept de langage informatique qui se généralise dans les années 80. Aujourd'hui dans sa version 2.3, le langage est standardisé par l'OMG depuis 1997. Le méta-modèle UML définit pas moins de 240 éléments qui permettent de modéliser un système. La base de tout modèle UML d'un système se caractérise par une représentation basée sur des classes (élément de représentation plus ou moins abstraite) et des relations entre les différentes classes du modèle.

Aujourd'hui, l'UML est majoritairement utilisé dans le monde du génie logiciel pour modéliser une partie d'un système et générer de la documentation. Toutefois, l'utilisation de l'UML pour la conception de systèmes électroniques embarqués est une idée apparue seulement au début des années 2000, et ce grâce aux caractéristiques de ce langage. Dans [54] et [55], les auteurs 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.

2.2.3.1 Des vues graphiques du modèle

L'UML propose non seulement un cadre générique et standardisé pour modéliser un système, mais il offre également des vues graphiques de ce modèle permettant une meilleure compréhension et lecture de celui-ci. Le méta-modèle UML comporte aujourd'hui 13 diagrammes (ou vues) différents. Ces différents diagrammes sont généralement classés en deux groupes distincts :

- **les diagrammes structurels** (cf. Tableau 2) : permettent de représenter les différents éléments du système, leurs caractéristiques et leurs relations entre eux. Cependant ces vues sont purement statiques ;

Tableau 2. Présentation des principaux diagrammes structurels UML.

Types de diagramme	Objectifs/Intérêts
Diagramme de classes	Représente les classes qui interviennent dans le système modélisé et les relations entre elles
Diagramme d'objets	Représente une instance du diagramme de classe, cas particulier
Diagramme de composants	Décrit l'architecture physique du système en identifiant les composants et leurs interfaces

- **les diagrammes comportementaux** (cf. Tableau 3) : permettent de représenter le comportement dynamique du système. Ces diagrammes permettent d'illustrer comment réagit un élément du système lorsqu'une action (un évènement) est menée.

Tableau 3. Présentation des principaux diagrammes comportementaux UML.

Types de diagrammes	Objectifs/Intérêts
Diagramme de cas d'utilisation	Exprime les interactions entre le système et les acteurs extérieurs
Diagramme d'état-transition	Exprime sous forme de machine d'état et de transition le comportement d'un sous-ensemble du système, sa réaction suite à un évènement
Diagramme d'activité	Exprime l'enchainement d'actions d'un sous-ensemble du système à la suite d'un évènement ; variante du diagramme d'état-transition
Diagramme de séquence	Exprime les échanges d'informations, sous forme de message, dans le temps entre les différents éléments du système

De toute évidence, la modélisation d'un système ne doit pas systématiquement faire appel à l'ensemble des diagrammes qu'offre UML. Certains diagrammes illustrent les mêmes informations, alors que d'autres diagrammes présentent des points de vue différents du système. De ce fait, selon les besoins et l'utilisation qui est fait du modèle, celui-ci doit faire appel aux diagrammes appropriés.

A noter qu'UML ne définit en aucun cas une méthodologie de conception. C'est à la fois un avantage et un inconvénient. En effet, ne pas avoir formalisé une méthodologie associée au langage, offre de la souplesse, ce qui permet aux équipes de modéliser leur système de la manière dont elles le souhaitent, suivant leurs besoins et leurs contraintes : le choix des diagrammes est totalement libre. Cet avantage peut-être également vu comme un inconvénient car pour un même système, plusieurs modélisations différentes peuvent être réalisées par des équipes différentes.

2.2.3.2 Des extensions d'UML

Couramment utilisé dans le monde du génie logiciel, l'UML n'en est pas moins cantonné à ce domaine. Il peut potentiellement s'utiliser dans tous les domaines métiers. 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, prenant la forme de profil, permet de définir un nouveau méta-modèle le plus souvent adapté à un domaine en particulier comme le sont les DSL[6] (*Domain Specific Language*). Cependant, contrairement aux DSL, le profil UML se base sur tout ou partie du méta-modèle UML et l'étend en définissant de nouveaux éléments (Figure 20).

[6] 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.

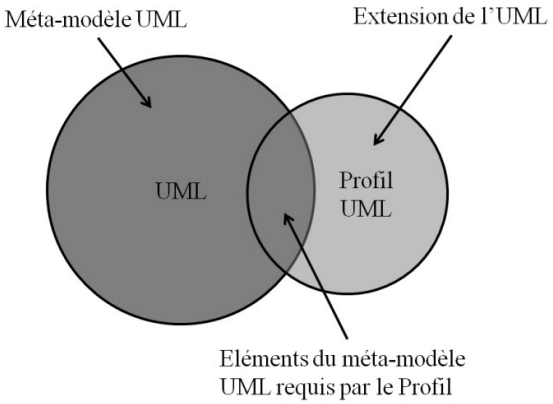


Figure 20. Extension d'UML à l'aide du mécanisme des profils.

L'OMG standardise un certain nombre de profil tel que :

- SysML (*System Modeling Language*) [56] : 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 [57] : 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*) [58] : 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*) [59] : profil qui reprend des éléments du profil SPT (*Schedulability, Performance and Time*) [60] et ajoute des concepts pour la modélisation de système soumis à des contraintes temps réelles.

D'autres profils existent mais ne sont pas standardisés par l'OMG, et de ce fait ne sont pas aussi facilement exploitable. Nous pouvons notamment citer :

- UML for SystemC [61] : profil dédié à la modélisation de système électronique qui permet une génération de code SystemC à différents niveaux d'abstraction, profil développé par STMicroelectronics ;
- UML-RT [62] : profil qui étend le langage UML avec des éléments du langage ROOM [63], proposé par B. SELIC, dédié à la modélisation du logiciel embarqué temps réel. Ce profil est la base du profil SPT ;

Pour utiliser les éléments de ces profils dans un modèle UML il faut utiliser le concept de stéréotype[7].

[7] Stéréotype : mécanisme qui permet d'annoter un modèle UML avec des éléments d'un profil UML pour ajouter des informations de sémantiques.

2.2.3.3 Un format de fichier d'échanges des modèles

Dans un flot de développement logiciel et matériel, la réutilisation de composants est un mécanisme qui fait de plus en plus partie de la culture de l'ingénieur. Le problème de la réutilisation vient souvent de la multitude de langages utilisés. De ce fait la réutilisation des composants est le plus souvent complexe bien que très pratique. Cette réutilisation est non seulement facilitée par des composants basés sur des interfaces standardisés tel que le propose l'IP-XACT [35], standard qui définit les interfaces des composants électroniques et ainsi facilite leur interopérabilité et leur réutilisation, mais aussi sur un langage commun standardisé. L'IDM reprend ce principe et préconise la réutilisation des modèles. Pour cela, la notion d'interopérabilité est primordiale. L'utilisation d'un langage de modélisation UML facilite celle-ci. L'échange de modèle se fait par le biais de fichier XMI (*XML Metadata Interchange*) [64] basé sur le langage de balises XML (*eXtensible Markup Language*) [65]. Pour échanger, réutiliser un modèle UML il est donc préconiser d'utiliser le format XMI. Une transformation du modèle UML est effectuée pour obtenir le modèle XMI.

2.2.3.4 Les outils associés

Pour réaliser un modèle UML un grand nombre d'outil soumis à licence payante ou en accès libre sont disponibles. Nous n'avons pas velléité à en faire une liste exhaustive mais citer quelques-uns qui se démarquent du lot soit de part leur implantation, soit de part leur particularité qui font d'eux leur principal intérêt :

- Rational Rose Modeler d'IBM [66] : outil de modélisation UML probablement le plus répandu sur le marché ;
- Rational Rhapsody d'IBM [67] : outil de modélisation UML (intégrant également le profil SysML, et des DSL tel que AUTOSAR [68], DoDAF [69], MODAF [70], etc.) offre la possibilité d'exécuter les modèles UML, ce qui n'est pas définit dans la norme UML. Toutefois des travaux à ce sujet sont en cours pour proposer un standard d'exécution des modèles UML dans un groupe de travail à l'OMG ;
- Papyrus du CEA [71] : outil de modélisation UML, alternative sous licence gratuite, intégré sous forme de plug-in dans l'environnement Eclipse. Cet outil intègre nativement les profils SysML et MARTE et propose également un générateur de code C++ intégré.

L'un des principaux inconvénients de la majorité des outils de modélisation UML est le fait que les outils n'offre pas une implémentation fidèle de la norme UML 2.1 en tous points. De ce fait, la réutilisation et la pérennité des modèles développés est donc le plus souvent fortement contraint à la durée de vie de l'outil de modélisation à l'origine du modèle.

2.2.4 Les modèles du MDA

Comme nous l'avons indiqué précédemment, l'un des principaux concepts préconisés par MDA est la séparation du modèle domaine métier de la plate-forme matérielle. Ceci permet d'assurer l'indépendance du modèle applicatif du système des contraintes de l'architecture de la plate-forme matérielle. MDA définit 4 modèles qui peuvent se mapper sur un diagramme en Y (Figure 21).

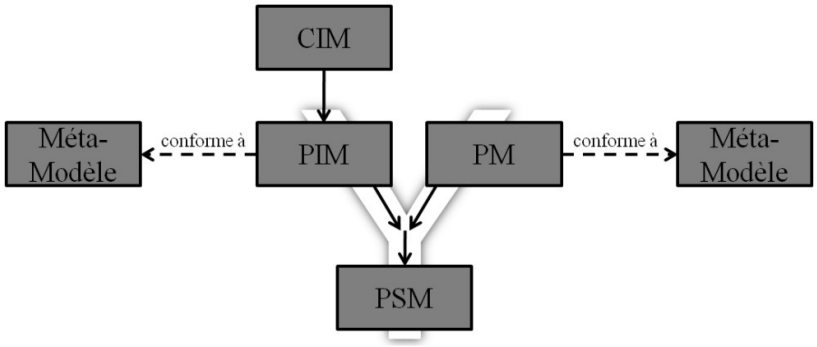


Figure 21. Diagramme en Y des modèles du MDA.

- CIM (*Computation Independent Model*) : représentation du modèle métier au niveau système. Ce modèle, s'abstrayant tout ou en partie de l'architecture fonctionnelle du système, représente les cas d'utilisation du système ; Modélisation des exigences métiers et des interactions du système avec son environnement. Le profil SysML est particulièrement bien adapté pour représenter ce modèle ;
- PIM (*Platform Independent Model*) : représentation du modèle fonctionnel du système. Ce modèle représente l'architecture fonctionnelle ainsi que la description comportementale des entités et des services du système. Cette représentation est indépendante de la plate-forme matérielle cible et donc des contraintes d'implémentations. Cette indépendance permet une certaine pérennité du modèle tout au long du processus de développement. Une représentation UML est souvent associée à un langage de modélisation des contraintes tel que l'OCL[8] (*Object Constraint Language*) ;
- PM (*Platform Model*) : représentation de la plate-forme matérielle cible. Ce modèle représente l'architecture de la plate-forme matérielle, ses ressources ainsi que les services associés de ces dernières. Les caractéristiques techniques et physiques peuvent également apparaître selon les besoins du PM ;
- PSM (*Platform Specific Model*) : représentation fonctionnelle du système dépendant des caractéristiques de la plate-forme matérielle cible. Ce modèle est le résultat de l'allocation des entités de l'application (décrites par le PIM) sur les ressources de calcul, de mémorisation et de communication du modèle de la plate-forme matérielle PM.

[8] OCL : langage qui permet d'ajouter des informations de contraintes (invariant) aux modèles UML sous la forme d'annotations.

L'épine dorsale du MDA (le diagramme en Y : PIM, PM, PSM) n'est pas figée. L'OMG met en avant la transformation de modèle, concept de base de l'IDM repris dans le MDA. L'idée consiste à remplacer le paradigme des méthodologies habituelles (analyse conception codage manuelle) par une succession de transformations sur les modèles comme l'illustre le diagramme en Y à plusieurs branches de la Figure 22.

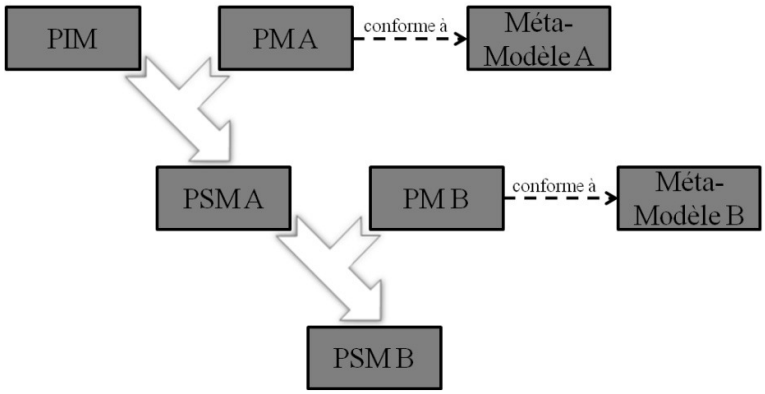


Figure 22. Dérivée du diagramme en Y.

2.3 Etat des lieux des méthodologies proposées

Afin de prouver qu'une nouvelle méthodologie de co-conception, pour le développement de systèmes temps réel embarqués tels que les systèmes de télécommunications, est nécessaire nous avons fait un état des lieux des différentes solutions proposées. Notre objectif dans ce mémoire n'est pas de présenter toutes les approches proposées mais de mettre en évidence les approches les plus significatives et intéressantes de notre point de vue. Il existe une multitude de classification possible des différentes approches de conception, basées sur la modélisation, proposées par la communauté. Le choix des catégories est régi par les caractéristiques que l'on souhaite mettre en avant des approches. Par exemple, nous aurions pu classer les méthodologies proposées suivant les trois catégories :

- les méthodologies orientées flot de développement logiciel embarqué ;
- les méthodologies orientées flot de développement matériel ;
- les méthodologies orientées exploration d'architecture.

Cependant nous souhaitons dans ce document marquer plus particulièrement l'accent sur l'utilisation ou non de l'approche MDA. Nous avons donc fait le choix de classer les approches en deux catégories :

- D'un côté, les méthodologies basées sur l'approche MDA ;
- De l'autre côté, les méthodologies non-basées sur l'approche MDA.

2.3.1 Les approches non-MDA

Nous présentons dans cette section des approches de co-conception basées sur la notion de modélisation. Toutefois celles-ci ne sont pas issues du MDA.

2.3.1.1 L'outil Matlab/Simulink associé à l'outil HDL Coder

L'approche Model-Based Design proposée par la société The MathWork [72] n'est pas en tant que telle une méthodologie mais plutôt un environnement qui permet de faire du prototypage rapide, depuis une modélisation fonctionnelle jusqu'à l'implémentation matérielle sur DSP ou FPGA, en intégrant l'automatisation de tests. La société américaine propose une mise en œuvre concrète de la conception de système embarqué à partir de modèle de haut niveau. A partir d'un modèle fonctionnel Matlab/Simulink, transformé en un modèle équivalent utilisant des opérations virgule fixe (à l'aide de Simulink Fixed Point), une génération de code HDL (VHDL ou Verilog) par le biais de l'outil Simulink HDL Coder [73] est réalisée. De plus, à l'aide de l'outil EDA Simulator Link une co-simulation du code HDL généré et du C embarqué (pour la partie logiciel embarqué), peut-être effectuée afin de valider le partitionnement. A noter que ce flot est plus particulièrement dédié pour le domaine du traitement du signal. De notre point de vue, c'est aujourd'hui probablement l'une des approches de conception basé sur les modèles la plus mature. Cette approche est déjà expérimentée par des industriels tel Xilinx ou encore ST-Ericsson pour la réalisation d'une partie d'un système [74]. L'un des avantages de cette approche est l'utilisation massive de Matlab par les industriels de l'électronique embarqué. Par contre, l'un des points faible de cette approche est l'utilisation de modèle non-standardisé à l'opposé de l'UML. De plus, cette approche ne propose pas une séparation PIM/PM comme le prône MDA et donc ne semble pas aussi souple. Toutefois il existe des passerelles entre Matlab/Simulink et des modèles UML/SysML, profil dont la sémantique est proche de celle des modèles Matlab, comme nous explique les auteurs de [75][76].

2.3.1.2 La méthodologie AAA

La méthodologie AAA (*Adéquation Algorithme Architecture*) [77] se base sur une approche en Y comme MDA. L'objectif de cette méthode est de proposer un environnement pour faire du prototypage rapide et permet de faire une analyse du partitionnement des tâches applicatives sur les ressources de la plate-forme d'exécution. Contrairement à MDA, la modélisation PIM et PM ne se base pas sur un standard tel que l'UML mais sur une représentation propriétaire des fonctionnalités du système (exprimant les interactions entre les fonctions et donc les possibilités de parallélisations) et des ressources matérielles (exprimant le parallélisme effectivement disponible). La Figure 23 illustre le principe de cette méthodologie AAA. Les deux modèles sont annotés avec des caractéristiques temporelles.

Un algorithme d'adéquation basé sur des heuristiques permet de déterminer un partitionnement en projetant le modèle fonctionnel sur le modèle de plate-forme matérielle afin de répondre au mieux aux contraintes imposées. Il en résulte un graphe temporel et un code d'exécution générique.

Cette méthodologie est supportée par l'outil Syndex [78] initié par Yves Sorel de l'INRIA.

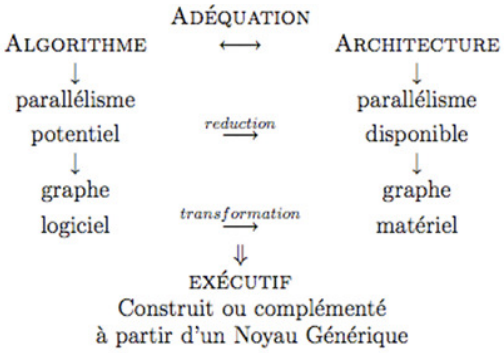


Figure 23. Illustration de la méthodologie AAA.

2.3.1.3 Ptolemy II

L'environnement Ptolemy II [79], évolution de Ptolemy [80], propose une modélisation d'un système sous forme de composant hétérogène indépendant de toute plate-forme d'exécution. L'environnement utilise la composition hiérarchique pour traiter l'hétérogénéité. L'approche se focalise sur la modélisation et l'analyse des communications entre composants, communications définit par des sémantiques appelées MoC (*Modèles de Calculs*). La modélisation se base sur la notion d'acteur et de directeur. Les acteurs définissent les composants, décrit en Java, et les directeurs définissent le modèle d'exécution : sémantique de communication entre les acteurs (MoC). Dans un même modèle plusieurs MoC peuvent-être utilisés. Ptolemy II propose notamment la mise en œuvre des MoC CSP[9], SDF, KPN, DE.

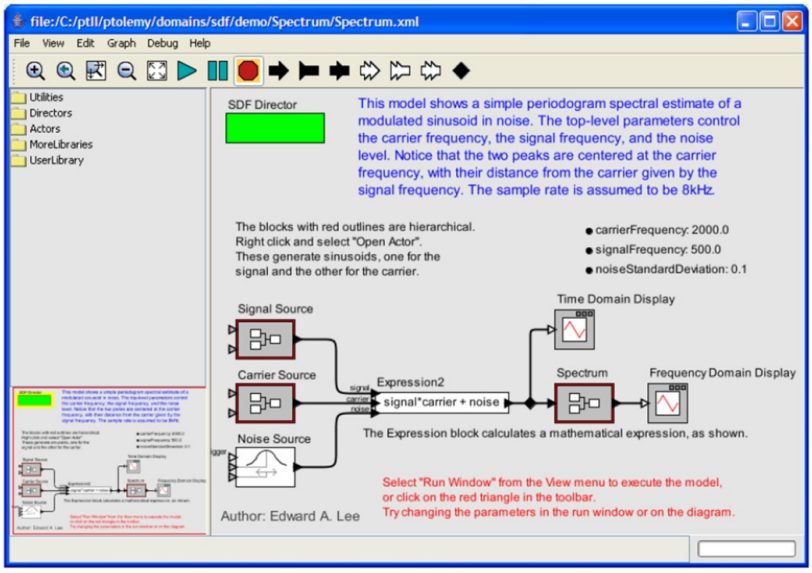


Figure 24. Vue générale de l'environnement Ptolemy.

[9] Modèle de calcul : CSP (*Communicating Sequential Processes*), SDF (*Sequential Data Flow*), KPN (*Kahn Process Network*), DE (*Discrete Event*).

La Figure 24 illustre l'environnement Ptolemy. Dans l'exemple illustré, un seul directeur régit la sémantique de communication entre les acteurs du système : SDF. Il est également intéressant de noter que la représentation graphique du modèle est proche de ce que l'on peut trouver sous Matlab/Simulink.

Cet environnement se concentre donc sur la modélisation fonctionnelle de haut niveau. Un des inconvénients de cette approche est l'utilisation de modèles basés sur une représentation non-standardisée, et donc contrainte à la durée de vie de l'outil.

2.3.1.4 Metro II

Une référence souvent citée est l'environnement de conception Metro II [81], évolution de Metropolis [82]. Cet environnement est basé sur la technique intitulée Platform-based design [37]. Metro II propose un environnement de conception pour des systèmes hétérogènes. Cet environnement est basé sur une représentation structurelle propriétaire mais utilisant le langage SystemC pour décrire le comportement interne de chacun des composants. Un peu à la manière de Ptolemy, Metro II se focalise sur la modélisation de la sémantique des communications entre les composants du système. L'infrastructure de l'environnement Metro II met à disposition différents MoC et permet sur un même modèle de mettre en œuvre plusieurs MoC différents. Les points d'entrée sont les spécifications textuelles ou graphiques des contraintes de conceptions non fonctionnelles, de l'application et de l'architecture de plate-forme d'exécution. Metro II propose une séparation à différents niveaux de granularité des fonctionnalités du système, de la plate-forme d'exécution, des communications et des contraintes temporelles. Nous retrouvons le principe de séparation des modèles prôné par le MDA. Cependant, cette approche n'utilise pas UML.

Afin de pallier aux limites [81] de la précédente version, Metropolis, les principales évolutions de l'environnement de co-conception Metro II sont :

- La capacité d'intégrer des IP existantes (réutilisation de modèles) ;
- La capacité d'explorer l'espace de conception de manière structurée.

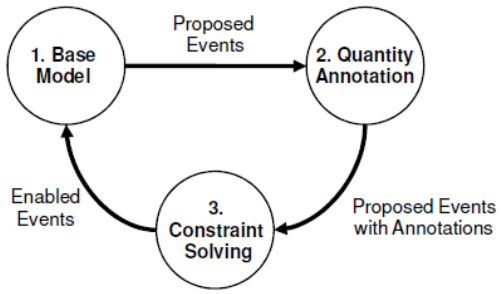


Figure 25. Illustration des 3 phases d'exécution dans Metro II, extrait de [81].

Le développement d'un modèle avec Metro II passe par 3 phases. La Figure 25 récapitule les 3 phases de développement de Metro II :

- La première étape, nommée *Base Model*, consiste à développer les modèles (applicatif et plate-forme d'exécution) et notamment à identifier les sémantiques qui régissent les communications entre les éléments du modèle. Celles-ci sont basées sur des événements ;
- La seconde étape nommée *Quantity Annotation* consiste à annoter les événements modélisant les communications, avec des quantités telles que des contraintes temporelles ;
- La troisième étape, nommée *Constraint Solving*, consiste à exécuter le modèle et entre autre valider les communications. Les résultats de l'exécution peuvent être réinjectés dans le modèle afin de le faire évoluer si ce dernier ne répond pas au cahier des charges.

Note : les 3 phases sont itératives afin de raffiner le modèle et ainsi obtenir un modèle du système respectant le cahier des charges.

2.3.1.5 La méthodologie MCSE

La méthodologie MCSE (*Méthodologie de Conception des Systèmes Electroniques*) [83] propose les éléments permettant une exploration de l'architecture du système au niveau transactionnel (TLM – *Transaction Level Modeling*), un des points cruciaux de la co-conception. Cette méthodologie est supportée par l'outil Cofluent Studio [84] de CoFluent Design. MCSE est basé sur le paradigme en Y, comme l'approche MDA. A l'inverse de cette dernière, MCSE utilise un méta-modèle propriétaire pour modéliser le modèles PIM, PM et PSM comme illustré par la Figure 26. Un peu à la manière de l'approche AAA, le modèle fonctionnel (functional model) caractérise les possibilités de parallélisassions des fonctionnalités du système entre elles. Quant au modèle d'exécution (executive structure), il représente les capacités effectives de parallélisations de la plate-forme d'exécution. A partir du modèle alloué (architectural model), une simulation permet d'évaluer l'ordonnancement et la parallélisation effective des tâches. La simulation proposée illustre également la charge des unités de calcul et des bus de communications (cf. Figure 27). Les résultats obtenus permettent de vérifier si les contraintes temporelles du système sont respectées et ainsi de valider l'architecture du système. Cette simulation se base sur l'environnement SystemC.

A noter, cette approche n'a pas pour objectif de proposer un flot complet depuis les spécifications système jusqu'à la génération des codes d'implémentation du code logiciel embarqué et du code RTL.

Dans [85] la société Cofluent Design présente une solution pour en partie palier à la représentation propriétaire des modèles : l'outil Cofluent Studio offre la possibilité d'importer des modèles UML/SysML/MARTE capturés dans l'outil MagicDraw [86] (accessible directement dans Cofluent Studio via un plug-in). Une vérification de ce dernier est effectuée (compatibilité du modèle avec la méthodologie MCSE) avant de transformer le modèle UML en un équivalent MSCE.

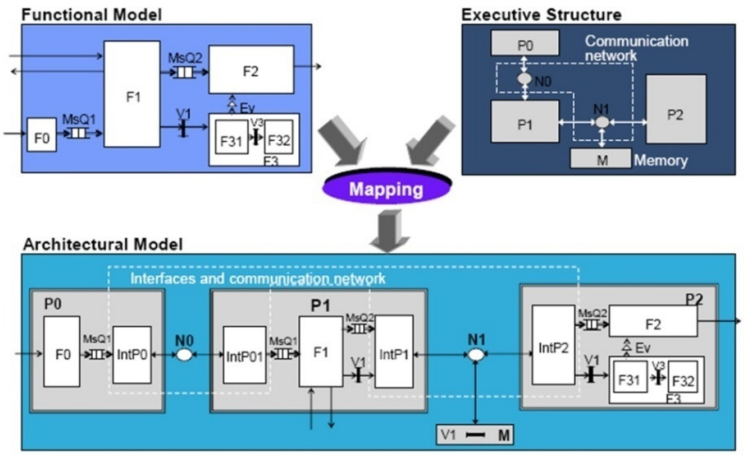


Figure 26. Illustration des modèles MCSE [83].

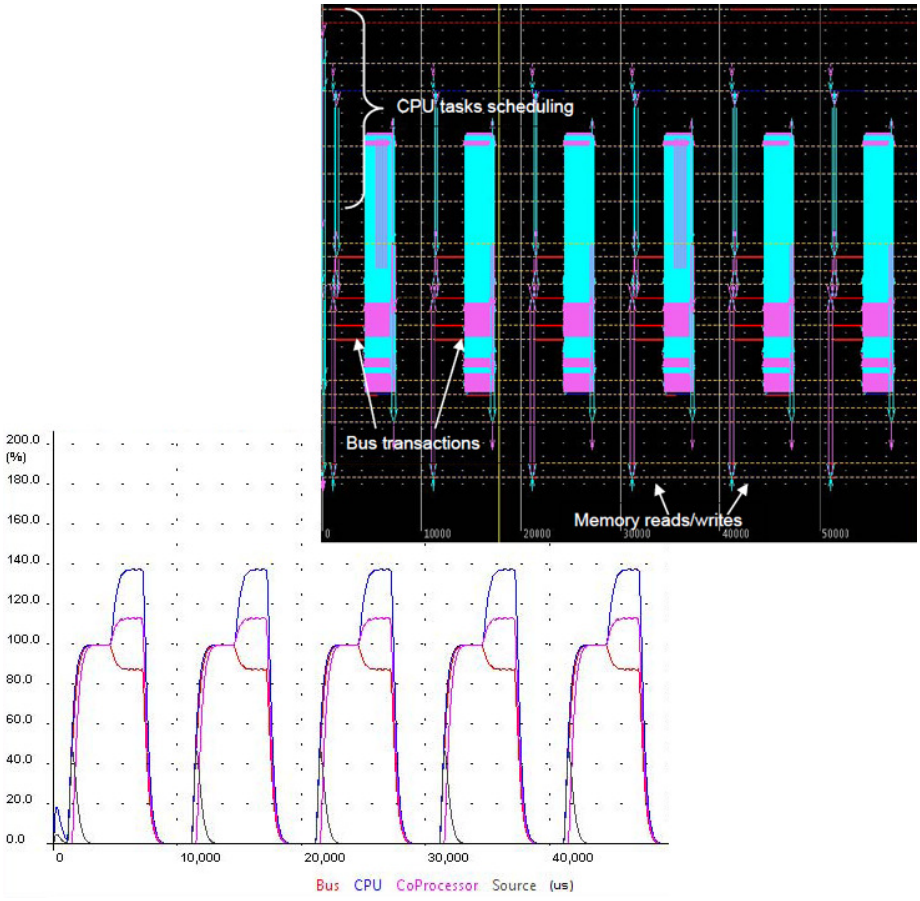


Figure 27. Extrait de résultats de simulations sous Cofluent Studio, charges des unités de calcul et analyse des transactions [85].

Cette approche fut utilisée dans le projet de recherche européen MARTES [87], projet proposant une approche de conception basée sur les modèles pour la conception de systèmes temps réel embarqués. Au travers de ce projet, le centre de recherche de Nokia (NRC) a mis en œuvre la méthodologie MCSE dans le cadre de ses activités de recherche sur les architectures des téléphones de future génération conçues pour la convergence numérique [88].

2.3.2 Les approches MDA

Nous présentons dans cette section les propositions de méthodologie de co-conception basées MDA qui nous semble les plus significatives. A l'inverse des précédentes méthodologies, les propositions basées sur le MDA en sont encore principalement au stade de recherche et manque de maturité pour être concrètement utilisées par les industriels.

2.3.2.1 A3S

Le projet de recherche A3S [89] a donné lieu à une des premières mises en œuvre concrètes de l'approche MDA. Le projet a abouti à la définition du profil A3S. La Figure 28 illustre l'environnement de développement A3S. Cette méthodologie permet de modéliser une application fonctionnelle et de décrire une architecture de plate-forme d'exécution à l'aide des diagrammes de l'UML 1.4. Pour réaliser les modèles, A3S se base sur une librairie de composants applicatifs (SWComponentLib) et sur une librairie de composants matériels (*HWComponentLib*). Le déploiement des fonctions du PIM se fait manuellement sur les ressources matérielles de la plate-forme matérielle (PM). La méthodologie A3S est supportée par l'environnement de modélisation Objecteering UML [90].

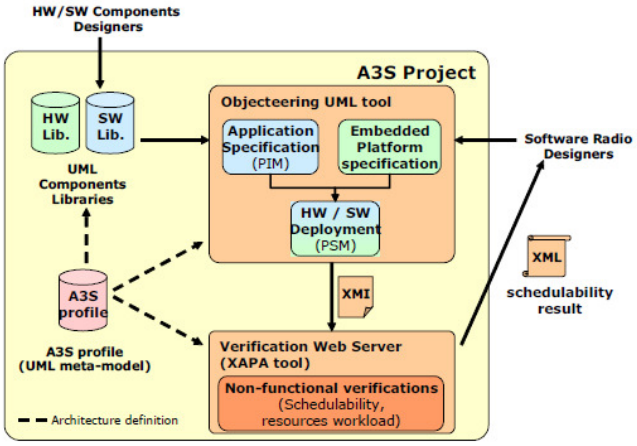


Figure 28. Environnement de modélisation du projet A3S [89].

Pour valider les caractéristiques non-fonctionnelles, le partitionnement et l'ordonnancement des tâches, une simulation du modèle résultant (PSM) est réalisée dans l'outil XAPAT. Un diagramme de GANT met en évidence, pour chacune des ressources de calcul de plate-forme matérielle, la charge de travail et le parallélisme.

L'approche proposée est particulièrement adaptée pour les systèmes de transmission.

2.3.2.2 UPES

L'Université de Milan, en collaboration avec STMicroElectronics, propose une méthodologie MDA intitulée UPES (*Unified Process for Embedded Systems*) [91]. Cette méthodologie se focalise plus particulièrement sur l'exploration de l'architecture matérielle au niveau TLM sans toutefois négliger la modélisation fonctionnelle du système. La partie gauche de la Figure 29 met en évidence deux parties distinctes dans le processus de développement proposé. Tout d'abord, la partie modélisation UML, PIM, PM et PSM permet de modéliser le système. Pour valider l'architecture de la plate-forme, le déploiement (partitionnement logiciel/matériel), l'ordonnancement des tâches, les auteurs proposent de changer de langage de modélisation et passer dans l'environnement SystemC. Afin de permettre cela, les modèles UML sont surchargés de stéréotypes issus du profil UML for SystemC [61].

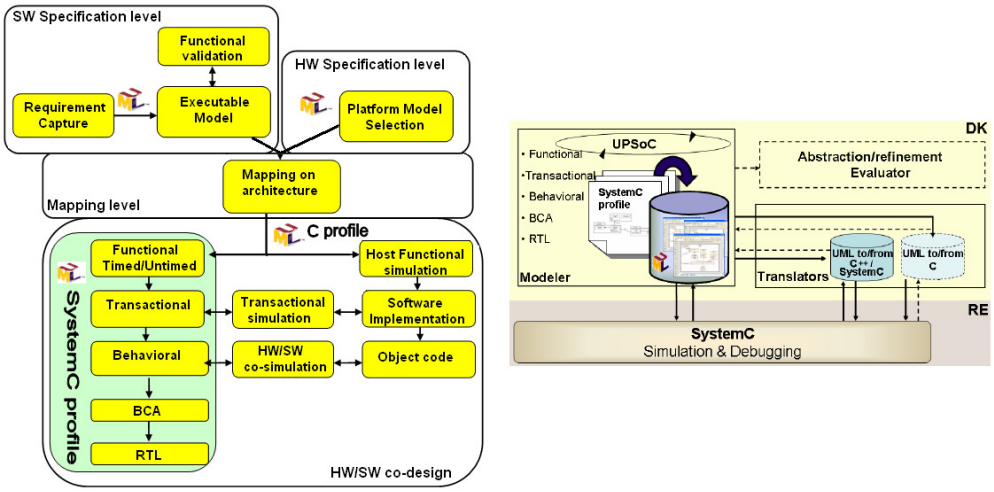


Figure 29. Méthodologie et environnement de développement UPES [91].

A partir du modèle alloué, une génération de code SystemC pour la partie matérielle et C pour le logiciel embarqué est effectuée. Le monde SystemC permet d'effectuer une co-simulation logiciel embarqué/matériel à différents niveaux d'abstraction.

2.3.2.3 GASPARD

L'équipe de recherche DaRT du LIFL, dirigée par Jean-Luc DEKEYSER, propose une implémentation de l'approche MDA dans un environnement Eclipse : Gaspard [92]. Aujourd'hui, dans sa seconde version, cet environnement permet une modélisation MDA associant une modélisation UML avec le profil MARTE, profil dédié à la modélisation de système embarqué temps réel. Le flot de co-conception proposé, illustré sur la Figure 30, est tout particulièrement adapté à la modélisation d'applications massivement parallèles et distribuées supportées par des plates-formes d'exécution homogènes. Pour exprimer le parallélisme, Gaspard se base sur le langage Array-OL [93]. De plus, cette méthodologie définit des jeux de règles pour réaliser une génération de code VHDL et SystemC depuis les modèles UML.

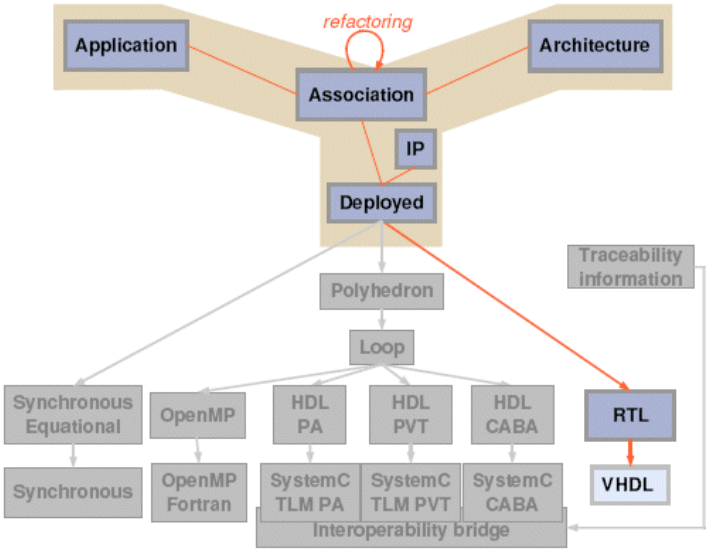


Figure 30. Flot de co-conception basé MDA dans l'environnement Gaspard [92].

De plus, une des toutes dernières évolutions de Gaspard est la prise en compte du caractère reconfigurable d'un système tant au niveau applicatif qu'au niveau de la plate-forme matérielle. En effet, l'environnement Gaspard permet de modéliser un FPGA disposant de la technologie de reconfiguration dynamique partielle.

2.3.2.4 Et bien d'autres …

Il existe de nombreuses autres propositions qui se basent sur le MDA. Nous pouvons notamment citer les travaux de l'Université de Paderborn [94] qui propose une méthodologie basé sur le MDA pour la modélisation et une co-simulation C/SystemC pour valider le modèle. La co-simulation utilise l'environnement QEMU [95] pour simuler le logiciel embarqué et Modelsim pour la partie matérielle. Cette méthodologie est proche d'UPES. D'autres méthodologies se focalisent sur l'une ou l'autre des branches du Y. Par exemple, la méthodologie ACCORD$|_{UML}$ [96] est plus particulièrement dédiée au développement du logiciel embarqué à partir de modèles UML. Cette dernière méthodologie est l'une des toutes premières initiatives basées sur le MDA. Pour la branche de droite du Y, à savoir le développement de la plate-forme matérielle, le passage du monde UML vers le SystemC est privilégié par de nombreuses propositions, telle que [97] un peu comme à la manière d'UPES.

2.3.3 Synthèse

Comme nous venons de le montrer, il existe de nombreuses propositions de méthodologies de co-conception basées sur des modèles de haut niveau. Le Tableau 4 récapitule les caractéristiques des différentes propositions.

Les premières (Matlab, Ptolemy, Metro II, MCSE) se basent sur des méta-modèles propriétaires pour modéliser les systèmes. De plus, la plupart d'entre elles ne couvrent pas la totalité du flot de développement d'un système temps réel embarqué. En effet, elles se focalisent sur un point en particulier, tel que la modélisation de la sémantique des communications (Ptolemy, Metro II) ou encore l'exploration d'architecture (MSCE). Seule la proposition de Matlab couvre l'ensemble du flot, de la modélisation jusqu'à la mise en œuvre du logiciel embarqué et du code RTL pour la plate-forme matérielle. Toutes ces propositions sont intéressantes mais n'offrent pas un environnement basé sur des standards. De ce fait l'interopérabilité, la réutilisation, ou encore la communication entres les équipes de conception n'en est pas amélioré. Toutefois, ces propositions sont suffisamment matures pour être déjà utilisées par des industriels.

Les secondes (A3S, UPES, Gaspard, etc) mettent en œuvre l'approche MDA, standardisée, chacune en l'adaptant à ses besoins. Au contraire des premières propositions, la majorité de ces dernières proposent un flot de co-conception complet, allant de la modélisation du système jusqu'à la génération de code logiciel et matériel. Toutefois, certaines d'entre elles se focalisent plus particulièrement sur l'une ou l'autre des deux branches du Y (PIM, PM, PSM) : à savoir plutôt sur le développement logiciel ou plutôt sur le développement de plate-forme matérielle. Le principale avantage de ces méthodologies est de proposer un environnement offrant l'interopérabilité des modèles, la réutilisation des modèles et donc améliorer la productivité. Cependant ces méthodologies manquent encore de maturité.

L'un des points faibles majeur des méthodologies actuellement utilisées chez les industriels, ainsi que des nouvelles méthodologies basées sur les modèles, présentées précédemment, est le manque de formalisme de celles-ci. En effet, les différentes étapes d'un processus de co-conception ne sont pas explicitement formalisées. Ce qui, de notre point de vue, est préjudiciable puisque chaque industriel, chaque équipe de développement va finalement mettre en œuvre ces étapes de manière différente. La formalisation de chacune des étapes du processus de développement, et des activités de chacune de ces étapes, permettrait de mieux guider les équipes de conception et d'éviter de commettre des erreurs non pas liées au domaine métier, aux compétences des équipes, mais à un manque du processus de développement en lui-même. Or dans un contexte de plus en plus hétérogène tel que celui du développement des systèmes électroniques embarqués, plus le processus de développement sera clarifié plus les équipes seront efficaces et concentrées sur leur objectif principal : à savoir le développement du système en question et non sur des soucis liées au processus. Bien que répondant aux challenges qu'impose la co-conception de système électronique embarqué temps réel, les approches proposées soit ne couvrent pas l'ensemble du flot de développement, soit ne se basent pas sur des standards de modélisation. Toutefois la principale carence de toutes ces approches est le manque de formalisme de celles-ci.

Tableau 4. Synthèse et comparaison des différentes approches de co-conception basées sur des modèles de haut niveau.

Méthodologie	Objectif	Approche MDA	Méta-modèle	Partitionnement	Génération de code
Matlab + HDL coder + EDA simulator [72]	Prototypage rapide	Non MDA	Matlab/Simulink	?	C et RTL (VHDL ou Verilog)
AAA [77]	Prototypage rapide	Non MDA	?	automatique	C générique
Ptolemy II [79]	Modélisation à haut du système et simulation de la sémantique des communications (MoC)	Non MDA	Représentation propriétaire + Java	Aucun, pas de plate-forme matérielle	-
Metro II [81]	Environnement de conception pour système hétérogène	Proche MDA amis nin UML	Méta-modèle plateform based design	manuel	-
MCSE [83]	Exploration d'architecture	Proche MDA mais non UML	propriétaire	manuel	SystemC TLM
A3S [89]	Valider un système au niveau système	MDA	UML 1.4 + profile A 3S	manuel	-
UPES [91]	Conception conjointe et simulation du modèle résultant	MDA	UML 2.1 + UML for SystemC	manuel	SystemC et C
Gaspard [92]	Environnement de Conception pour architecture distribuée et massivement parallèle	MDA	UML 2.1 + MARTE	manuel	SystemC et VHDL

2.4 Conclusion

Dans ce chapitre, nous avons, tout d'abord, mis en évidence le besoin de proposer de nouvelles méthodologies de conception pour les systèmes temps réel embarqués. Effectivement, les nombreux challenges qu'impose le développement de ces systèmes sont de plus en plus délicats à relever avec les méthodologies de co-conception actuellement utilisées. Aujourd'hui, l'écart entre la productivité et les technologies à disposition des ingénieurs est de plus en plus important. Ainsi, pour améliorer la productivité des équipes de conception, il est impératif de leur mettre à disposition des méthodologies de co-conception améliorant cette productivité. L'une des approches prometteuses est l'utilisation de modèles de haut niveau, approche prônée par la communauté de l'ESL.

Ensuite, nous avons présenté une des approches basées sur la modélisation de haut niveau : l'architecture dirigée par les modèles (MDA), dérivée de l'ingénierie dirigée par les modèles (IDM). Cette approche prône la séparation du modèle applicatif et de la plate-forme d'exécution, la transformation de modèles, l'interopérabilité, la réutilisation des modèles. Pour cela l'approche MDA se base sur un langage de modélisation standardisé : UML.

Enfin, nous avons réalisé un état des lieux des propositions de méthodologies de co-conception basées sur des modèles de haut niveau, classées en deux catégories : approches basées MDA et approches non basées MDA. Toutes les approches présentées offrent des avantages et des inconvénients. Cependant, les différentes étapes de conception de toutes ces propositions manquent de formalisme. Ainsi, bien que de nombreuses propositions soient existantes, il nous semble évident qu'il faille proposer une nouvelle méthodologie se démarquant des précédentes en mettant l'accent sur la formalisation des étapes de conception.

Chapitre 3 Une méthodologie basée sur les modèles

Sommaire

Dans le chapitre précédent, nous avons mis en évidence la nécessité de proposer une autre méthodologie de conception basée sur l'approche ADM (*Architecture Dirigée par les Modèles*, ou MDA pour *Model Driven Architecture*) pour les systèmes embarqués temps réel. Dans ce chapitre, nous présentons la méthodologie MOPCOM, méthodologie basée sur MDA, pour laquelle nous avons contribué à la définition et à la mise en œuvre dans un contexte de système de communications sans-fil temps réel embarqué présenté dans le chapitre 1 de ce mémoire. Dans un premier temps, nous présentons une vue générale de la méthodologie et des outils associés pour la mettre en œuvre. Dans un second temps, nous ferons une description détaillée d'un des trois niveaux de modélisation de la méthodologie MOPCOM, intitulé EML dont cette thèse a établi la définition, dédié à l'exploration de l'architecture de la plate-forme matérielle dont nous avons décrit les caractéristiques. Enfin, nous proposons une mise en œuvre de la méthodologie dans son ensemble et une évaluation des étapes de modélisation.

3.1 La méthodologie MOPCOM

La méthodologie MOPCOM (*Modélisation et spécialisatiOn de Plates-formes et Composants MDA)* a été définie dans le cadre du projet ANR du même nom, afin de répondre aux challenges imposés par la co-conception de systèmes de plus en plus complexes. La méthodologie MOPCOM a pour objectif de proposer un nouveau paradigme du flot de co-conception pour des systèmes électroniques embarqués temps réel et plus particulièrement pour des systèmes SoPC (*System on Programmable Chip*) hétérogènes. En effet, dans la première partie du chapitre 2 nous mettions en avant les limites du flot de co-conception actuellement utilisé que nous appelons « traditionnel ». Dans celui-ci (partie gauche de la Figure 31), le partitionnement logiciel/matériel intervient très en amont dans le flot de conception. Ce qui est aujourd'hui clairement identifié comme une limite.

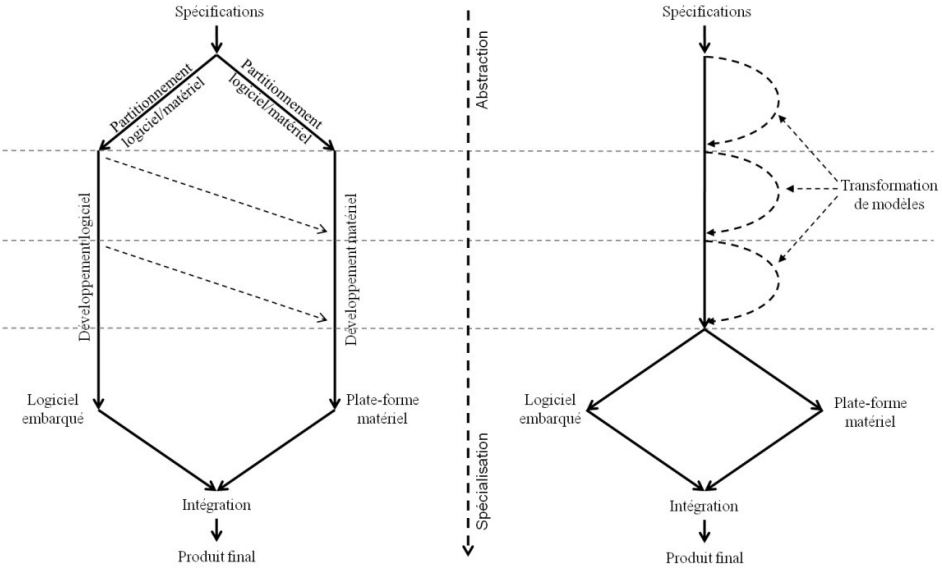


Figure 31. Flot de co-conception traditionnelle versus flot de co-conception MDA.

A l'inverse, le flot de co-conception (partie droite de la Figure 31) propose une approche de co-conception dirigée par les modèles, préconisée par l'IDM (*Ingénierie Dirigée par les Modèles*), et formalisée dans le MDA (*Model Driven Architecture*), présenté dans la section 2.2.2 du chapitre 2. En lieu et place de deux flots de développement parallèles (de la partie gauche de la Figure 31), l'approche vise à représenter le système sous forme de modèles et de transformations (reproductibles et automatisées) successives de modèles pour aboutir, d'un côté, à la génération de code du logiciel embarqué et, de l'autre, de la génération de la mise en œuvre de la plate-forme matérielle. Cette approche basée sur un même langage de modélisation standardisé (UML) tout au long du processus de développement, à la fois pour l'applicatif et pour la plate-forme d'exécution, permet de proposer un flot de développement plus pérenne dans le temps car il est générique, basé sur des standards et se veut indépendant des outils de mise en œuvre le plus longtemps possible pendant l'ensemble de la conception.

La Figure 32 met en évidence le positionnement de la méthodologie MOPCOM dans l'ensemble du processus de développement d'un système.

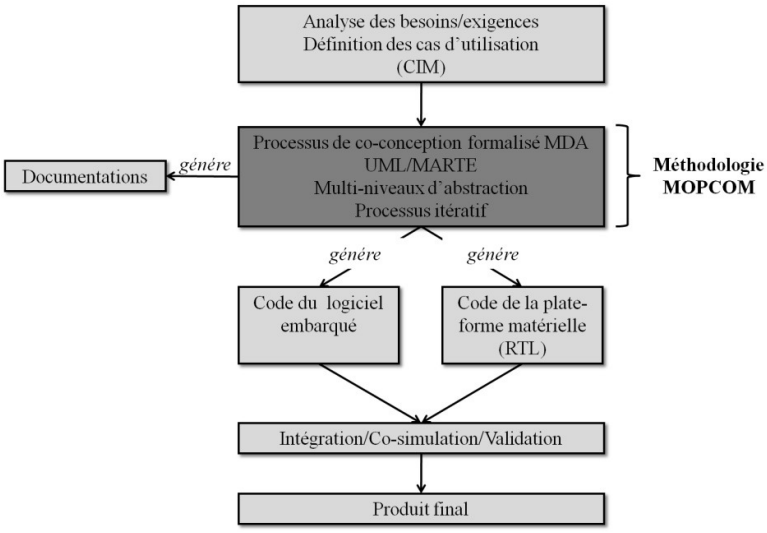


Figure 32. Positionnement de la méthodologie MOPCOM dans le flot de développement d'un système.

Le point d'entrée du processus est le modèle CIM (*Computation Independent Model*) du système. En sortie de la méthodologie MOPCOM trois générations sont effectuées à partir des modèles :

- la documentation du modèle ;
- le code pour le logiciel embarqué sur DSP (*Digital Signal Processing*), microcontrôleurs, etc ;
- le code pour la plate-forme matérielle pour composant FPGA (*Field Programmable Gate Array*).

La méthodologie MOPCOM se base sur l'approche MDA. De plus, cette méthodologie est tout particulièrement adaptée pour les systèmes électroniques embarqués temps réel. De ce fait, MOPCOM est l'une des premières initiatives, avec les travaux de Thalès (associé au CEA et au laboratoire de recherche I3S) [98], qui met en œuvre le profil MARTE (*Modeling and Analysis of Real Time Embedded system*). En effet, comme l'UML, le profil MARTE ne définit en aucun cas une méthode d'utilisation des éléments définis par lui-même. Pour récapituler, les éléments clés de la méthodologie MOPCOM sont les suivants :

- Couvrir le domaine de l'ESL (*Electronic System Level*) ;
- Utiliser l'approche MDA ;
- Modélisation UML/MARTE ;
- Processus de co-conception itératif descendant (Top-down) ;
- Génération automatique de code ;
- Génération documentaire.

Note : la génération documentaire est souvent négligée mais reste un point clé dans la production industrielle, voire un point primordial dans un contexte de certification (dans le domaine de l'aéronautique par exemple avec la norme DO-254).

3.1.1 Le profil MARTE

Bien qu'étant un des piliers de la méthodologie MOPCOM, la présentation détaillée du profil MARTE [59] n'est pas l'objectif de ce mémoire. En plus de la norme très volumineuse (équivalent à la norme UML), il existe un tutoriel très complet qui offre une très bonne visibilité sur le contenu et les objectifs de ce profil [99].

Basé sur le mécanisme d'extension du langage de modélisation UML (cf. Figure 20), le profil MARTE est un profil dédié à la modélisation de système temps réel embarqué (RTES pour *Real Time Embedded Systems*). L'un de ces principaux apports est la modélisation de la plate-forme d'exécution. Ce profil remplace le profil SPT (*Schedulability, Performance and Time*) [60], extension d'UML 1.x. Il permet de pallier aux manques de ce dernier et se veut plus concret et facile d'utilisation que le profil SPT, clairement dénoncé comme trop abstrait et trop complexe à mettre en œuvre. Les principales raisons d'être de ce nouveau profil sont les suivantes :

- Alignement avec la version 2.x d'UML, évolution notable par rapport à la version 1.x ;
- Besoin de modélisation de la plate-forme d'exécution (logicielle et matérielle) ;
- Besoin de prendre en compte la problématique de sémantique des communications entres les composants du système.

Le profil MARTE se base sur le méta-modèle UML 2.0 et utilise l'OCL (*Object Constraint Language*). De plus, non seulement ce profil reprend des éléments du profil SPT, mais il reprend également des éléments de quatre autres profils de l'OMG. Le Tableau 5 synthétise les éléments que le profil MARTE reprend des autres profils UML standardisés par l'OMG.

Tableau 5. Eléments définis dans les autres profils standardisés par l'OMG et adressés par le profil MARTE.

Profil UML	Eléments adressés par le profil MARTE
UML for QoS, FT characteristics and Mechanisms [57]	Propriétés non-fonctionnelles
SySML [56]	Spécialisation du concept d'allocation et réutilisation du concept de flot
RT-CORBA [100]	Architecture de plate-forme d'exécution peut-être annotée avec des éléments de MARTE pour l'analyse.
UML for SoC [58]	Eléments de modélisation de la plate-forme matérielle

Le profil MARTE est basé sur des concepts de haut niveau d'abstraction mais aussi sur des concepts proche d'une description RTL pour la plate-forme matérielle, permettant ainsi de couvrir une modélisation depuis les spécifications jusqu'à une description détaillée des systèmes embarqués temps réel. La Figure 33 illustre le domaine du profil MARTE.

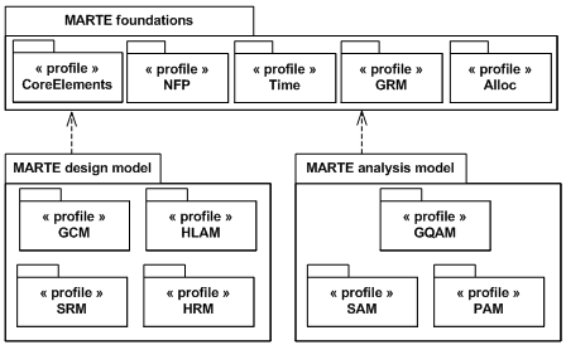


Figure 33. Architecture du profil MARTE [59].

Les quelques 700 concepts formalisés dans le profil MARTE sont regroupés dans des paquetages :

- MARTE foundations : paquetage qui regroupe les éléments de base du profil :
  - **Core Elements** : paquetage qui regroupe les concepts de base génériques du profil ;
  - **NFP** (*Non-Functional Properties*): paquetage qui regroupe les éléments permettant de définir des caractéristiques non-fonctionnelles ;
  - **Time** : paquetage qui regroupe les éléments temporels (horloge, durée, etc.) ;
  - **GRM** (*Generic Resource Modeling*) : paquetage qui regroupe des éléments permettant de modéliser une plate-forme d'exécution générique supportant des applications temps réel ;
  - **Alloc** : paquetage qui regroupe des éléments permettant de spécifier l'allocation des éléments du modèle PIM (*Platform Independant Model*) sur le modèle PM (*Platform Model*) pour obtenir le modèle PSM (*Platform Specific Model*) ;
- MARTE design model : paquetage qui regroupe les éléments permettant de modéliser l'architecture logicielle et matérielle du système, modélisation des ressources et des services :
  - **GCM** (*Generic Component Modeling*) : ce paquetage regroupe des éléments qui permettent de modéliser ;
  - **HLAM** (*High Level Application Modeling*) : paquetage qui regroupe des éléments permettant de modéliser l'architecture de la plate-forme d'exécution avec un certain niveau d'abstraction (aucune différence entre ressources logicielles et matérielles à ce niveau) ;
  - **SRM** (*Software Resource Modeling*) : paquetage qui regroupe des éléments permettant de modéliser les ressources logicielles de la plate-forme d'exécution du système ;
  - **HRM** (*Hardware Resource Modeling*) : paquetage qui regroupe des éléments permettant de modéliser les ressources matérielles de la plate-forme d'exécution du système.

- MARTE analysis model : paquetage qui regroupe les éléments permettant d'annoter le modèle en vue d'analyse d'ordonnancement, de performances :
  - **GQAM** (*Generic Quantitative Analysis Modeling*) : paquetage qui regroupe des éléments génériques permettant de faire de l'analyse du modèle ;
  - **SAM** (*Schedulability Analysis Modeling*) : paquetage qui regroupe des éléments permettant d'annoter le modèle pour faire de l'analyse d'ordonnancement ;
  - **PAM** (*Performance Analysis Modeling*) : paquetage qui regroupe des éléments permettant d'annoter le modèle pour faire de l'analyse de performance.

Note : Il est à noter que le modèle UML MARTE ne permet pas en lui même d'effectuer de telles analyses. Les éléments de ce profil ne pourront être concrètement exploités que par des outils d'analyse.

Nous souhaitons ici mettre en évidence le nombre conséquent de concepts proposés pour modéliser un système temps réel embarqué. Cependant, cela rend d'autant plus complexe de réaliser un modèle de qualité : il faut non seulement bien choisir le point de vue et le niveau d'abstraction que l'on souhaite représenter, mais en plus, il faut utiliser à bon escient les éléments UML et MARTE. De plus, le profil MARTE est encore récent, donc encore peu utilisé et par conséquent il manque de maturité. Enfin, comme pour l'UML, aucune méthodologie n'est définie avec MARTE. C'est à la fois un avantage et un inconvénient :

- d'un côté, cela laisse une totale liberté aux développeurs d'utiliser les éléments qu'ils souhaitent pour représenter le système ;
- d'un autre côté, les éléments du profil MARTE peuvent être mal compris et par conséquent mal utilisés.

De ce constat, il nous est apparu qu'il était nécessaire de proposer à travers MOPCOM une méthodologie permettant de mettre en œuvre concrètement ce profil MARTE dans une modélisation UML/MDA.

3.1.2 MOPCOM : un raffinement du MDA

La méthodologie MOPCOM peut-être vue comme un raffinement de l'approche MDA pour le contexte des équipements reconfigurables embarqués hétérogènes. Cette méthodologie repose sur 3 niveaux de modélisation : 3 niveaux d'abstraction. Chacun des 3 niveaux de modélisation reprend la structure en Y préconisée par MDA. Les différentes étapes de modélisation de cette méthodologie MOPCOM sont formalisées : un méta-modèle décrit les différentes activités de chaque étape du processus afin de guider les équipes de conception. Ce méta-modèle décrit le(s) point(s) d'entrée(s), le but, la stratégie à appliquer et le(s) point(s) de sortie(s) de chacune des activités qui constituent le processus de conception et précisent les éléments du profil MARTE à utiliser. Ce méta-modèle, utilise le langage de modélisation MODAL (*MOdel Driven Architecture Language*), extension de SPEM (*Software Process Engineering Modeling*) [101], présenté dans [102].

De plus, à l'aide du langage Kermeta [43], issu de l'INRIA, des contraintes liées à l'utilisation des méta-classes et/ou des stéréotypes pour chacune des activités sont définis. Comme nous l'avons présenté précédemment, comme l'UML, le profil MARTE se compose d'un nombre de méta-éléments importants. Au vue d'une telle richesse de concepts pour modéliser un système, les équipes de conception sont amenées à faire des choix pour représenter tel ou tel élément de leur système. Afin de leur éviter ce travail, souvent fastidieux et pouvant engendrer des erreurs lors de la conception, l'un des objectifs de la méthodologie MOPCOM est de proposer des règles méthodologiques permettant de faciliter les activités de conception. Toutefois, la méthodologie ne verrouille pas fortement le choix des méta-classes et des stéréotypes pour chacune des étapes du processus de conception. Une certaine souplesse est laissée aux équipes de développement. Dans ce chapitre, nous donnerons une description des différentes étapes du processus MOPCOM sans toutefois présenter le méta-modèle et les règles associées qui le régissent.

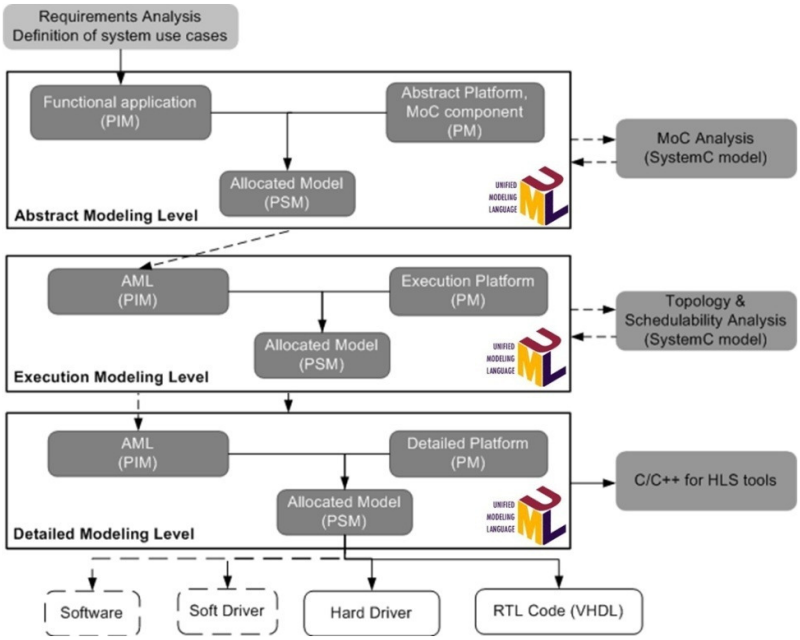


Figure 34. Flot de conception MOPCOM en 3 niveaux de modélisation UML/MARTE depuis les spécifications systèmes jusqu'à la génération de code.

Nous souhaitons mettre en avant le fait que les 3 niveaux de modélisation de MOPCOM ont été pensés de manière à être indépendant les uns des autres afin d'offrir la possibilité de les utiliser sans avoir à développer l'ensemble des modèles des 3 niveaux selon le centre d'intérêt des équipes de conception. La Figure 34 illustre les 3 niveaux de modélisation de MOPCOM.

Le point d'entrée du processus MOPCOM est le modèle CIM (*Computation Independent Model*) défini à partir du cahier des charges du produit : les exigences sont capturées sous la forme de cas d'utilisation.

Les travaux de thèse présentée dans ce document ont porté sur le niveau de modélisation EML en contribuant pour près des deux tiers environ de sa définition. De plus ces travaux ont également concerné dans une moindre mesure les niveaux de modélisation AML et DML.

3.1.2.1 AML (Abstract Modeling Level)

AML est le niveau de modélisation le plus abstrait du processus MOPCOM : ce niveau ne tient pas compte des contraintes temporelles et de mise en œuvre. Nous retrouvons 3 modèles distincts à ce niveau de représentation :

- Un modèle PIM : exprime l'architecture fonctionnelle du système (modèle de l'application) issu du modèle métier qui capture les exigences sous la forme de cas d'utilisation notamment ;
- Un modèle PM : propose une modélisation abstraite de la plate-forme qui représente des éléments de calcul virtuels offrant des ressources de calcul concurrents (utilisation du paquetage HLAM du profil MARTE) ainsi que des communications point-à-point régies par des modèles de calculs[10] (*MoC*). Un méta-modèle de MoC nommé CoMeta [103] est proposé avec MOPCOM ;
- Un modèle PSM : modèle résultant de l'allocation des fonctions applicatives du modèle PIM sur les ressources de calculs virtuelles du modèle PM. Ce modèle permet de valider l'architecture fonctionnelle, son comportement et de faire une analyse des communications (MoC).

Les principaux éléments du profil MARTE utilisés à ce niveau sont issus du paquetage HLAM pour le modèle PM virtuel, issus du paquetage NFP pour spécifier les propriétés non-fonctionnelles, et issus du paquetage VSL (*Value Specification Language*) pour exprimer propriétés et contraintes sur le système.

En résumé, le niveau AML adresse l'expression de la concurrence et de la communication sans présumer de la limite des ressources d'exécution de la plate-forme matérielle. Ce niveau est plus particulièrement traité dans la thèse d'Ali KOUDRI [104].

[10] MoC : modèles de calculs (ou encore model of computation en anglais). Les modèles de calcul sont indépendants de la plate-forme matérielle d'exécution. Un MoC définit la sémantique de communication entre deux éléments communiquant entre eux. Il existe de nombreux MoC tel que CSP, KPN, SDF, DE, etc.

Note : les éléments des paquetages NFP et VSL sont utilisés tout au long du processus dans les différents modèles des trois niveaux de MOPCOM.

3.1.2.2 EML (Execution Modeling Level)

EML est le niveau de modélisation intermédiaire du processus MOPCOM : ce niveau fait apparaitre l'architecture (topologie) réelle de la plate-forme d'exécution. Toutefois ce niveau ne spécifie pas le type des ressources (processeur, FPGA, etc.). Comme pour le niveau AML, nous retrouvons 3 modèles distincts à ce niveau de représentation :

- Un modèle PIM : modèle qui reprend le résultat du niveau AML (PSM) ;
- Un modèle PM : exprime la topologie réelle de l'architecture de la plate-forme d'exécution à l'aide notamment du paquetage GRM de MARTE. La topologie de la plate-forme matérielle est représentée à l'aide de nœuds de calcul (ressource de calcul générique), de nœuds de communication (bus qui implémentent des protocoles de haut niveau) et de nœuds de mémorisation (ressource mémoire générique partagée ou non) ;
- Un modèle PSM : modèle qui représente l'allocation des éléments du modèle PIM sur les composants du modèle PM. De plus, des annotations à l'aide des paquetages GQAM, SAM et PAM, permettront d'obtenir une analyse « gros grain » des performances et de l'ordonnancement du système.

En résumé, le niveau EML se focalise sur la modélisation de l'architecture de la plate-forme matérielle dans le but de faire de l'exploration d'architecture. Nous reviendrons plus en détails sur ce modèle dans la suite de ce chapitre.

3.1.2.3 DML (Detailed Modeling Level)

DML est le niveau de modélisation de MOPCOM le plus proche du niveau RTL. Nous retrouvons 3 modèles distincts à ce niveau de représentation :

- Un modèle PIM : modèle qui reprend le résultat du niveau AML (PSM) ;
- Un modèle PM : raffinement du modèle PM du niveau EML. Ce modèle offre une représentation détaillée de la plate-forme matérielle. Ce modèle se concentre à la fois sur les services fournis par les ressources que sur leurs caractéristiques physiques, en précisant le type des différents nœuds de celle-ci ainsi que leurs caractéristiques. Pour cela, les éléments du paquetage DRM (*Detailed Resource Modeling*) et de ses sous-paquetages SRM et HRM sont utilisés ;
- Un modèle PSM : modèle résultant de l'allocation des éléments du PIM sur les ressources de la plate-forme matérielle PM. A partir de ce modèle, une génération de code est proposée, VHDL pour la partie matérielle, et, C pour la partie logicielle embarquée ou utilisé pour faire de la synthèse comportementale comme nous le présentons dans le chapitre 4.

En résumé, le niveau DML *se focalise sur la modéli*sation détaillée de la plate-forme matérielle par un raffinement du PM du niveau de modélisation précédent en vue d'une génération de code de mise en œuvre matérielle et/ou logicielle. Ce niveau est plus particulièrement traité dans la thèse de Jorgiano VIDAL [105].

3.1.3 Génération de code

La génération de code est un service généralement proposée par les approches MDA. MOPCOM ne déroge pas à cette règle. Au-delà d'un processus de modélisation MDA multi-niveau, MOPCOM offre la possibilité de générer du code à partir des modèles UML. Aujourd'hui, MOPCOM se focalise sur la génération de 3 langages utilisés dans l'ESL : le VHDL, le C/C++ et le SystemC. Ces choix se justifient par le fait que MOPCOM adresse tout particulièrement les plate-formes SoC/SoPC. La génération de code s'appuie sur le mécanisme de transformation de modèle présenté sur la Figure 18. Chacune des 3 générations de code est basée sur un jeu de règle de transformation.

3.1.3.1 SystemC

La génération de code SystemC depuis les modèles UML se justifie principalement par le fait que les modèles UML ne sont pas nativement exécutables (des travaux sont en cours dans ce sens). Nous ne prétendons pas être novateur dans ce domaine puisque les méthodologies UPES [61][91] ou encore Gaspard [92] proposent également une telle génération de code. La librairie SystemC [106], basée sur le langage C++, permet une modélisation et une co-simulation multi-niveau du logiciel embarqué et de la plate-forme matérielle. Contrairement à l'approche UPES qui nécessite de stéréotyper le modèle UML à l'aide du profil UML profil for SystemC (qui reprend les éléments de la librairie SystemC), la génération de code UML vers SystemC mis en œuvre dans le cadre de MOPCOM ne nécessite pas de surcharger les modèles UML. Pour chacun des trois niveaux du processus de MOPCOM, une transformation des modèles PSM en un modèle équivalent SystemC (non-timé, timé, cycle-accurate) est effectuée. Le modèle SystemC généré est simulé pour valider le modèle UML. Nous reviendrons plus en détail sur l'intérêt et l'utilisation de ce générateur dans le chapitre 5.

3.1.3.2 VHDL

A partir du niveau DML de MOPCOM, niveau de modélisation le plus proche du niveau RTL, un générateur de code permet de générer l'infrastructure du code VHDL. Le code résultant décrit l'architecture hiérarchique de la plate-forme matérielle, les connexions entre les entités. De plus, du code VHDL est généré depuis les machines d'états souvent utilisées pour décrire la partie contrôle d'un composant. Par contre, la description comportementale de chaque entité VHDL n'est pas générée depuis le modèle UML. Nous privilégions la réutilisation d'IP (composants sous-forme de boite noire) soit issue de librairies, soit obtenue par synthèse comportementale. Nous reviendrons un plus en détail sur ce générateur dans le chapitre 4.

3.1.3.3 C/C++

La génération de code UML vers C/C++ est également proposée. Celle-ci a pour principale vocation de compléter le générateur UML vers VHDL. En effet, ce dernier génère uniquement sous forme de boite noire les différentes entités matérielles de la plate-forme ainsi que les machines d'états (représentation du contrôle). La description algorithmique de chaque entité est issue soit d'IPs (librairies), soit d'une synthèse de haut niveau (ou encore appelée synthèse comportementale). Ce générateur de code est à la fois basé sur l'UML et sur un langage d'action, dérivé du C++. Nous reviendrons un plus en détail sur ce générateur dans le chapitre 4.

3.1.3.4 D'autres générateurs de code

MOPCOM se focalise volontairement sur la génération de code en lien avec la plate-forme matérielle. Toutefois, la génération de code à partir de modèle MOPCOM ne se limite pas à ces langages. L'environnement proposé avec la méthodologie MOPCOM met à disposition le générateur et les jeux de règles afin de les modifier selon les besoins et de développer de nouveau jeu de règles.

3.1.4 Les outils associés

En plus de proposer une méthodologie de conception MDA formalisée à multi-niveau d'abstraction, de générateurs de code associés, MOPCOM propose un environnement d'outils qui permet de supporter la dite méthodologie pour développer un système temps réel embarqué. La méthodologie MOPCOM est supportée par un environnement d'outils basé sur les standards de l'OMG (MOF, UML, MDA, XMI) ainsi que sur l'environnement libre fourni par le Framework Eclipse (EMF, EMOF, Ecore). Ceci permet de rendre indépendant la méthodologie MOPCOM des outils pour la mettre en œuvre dans un soucis de portabilité, de réutilisation et de pérennité des modèles développés à l'aide de celle-ci. La Figure 35 illustre l'environnement d'outillage associé à MOPCOM.

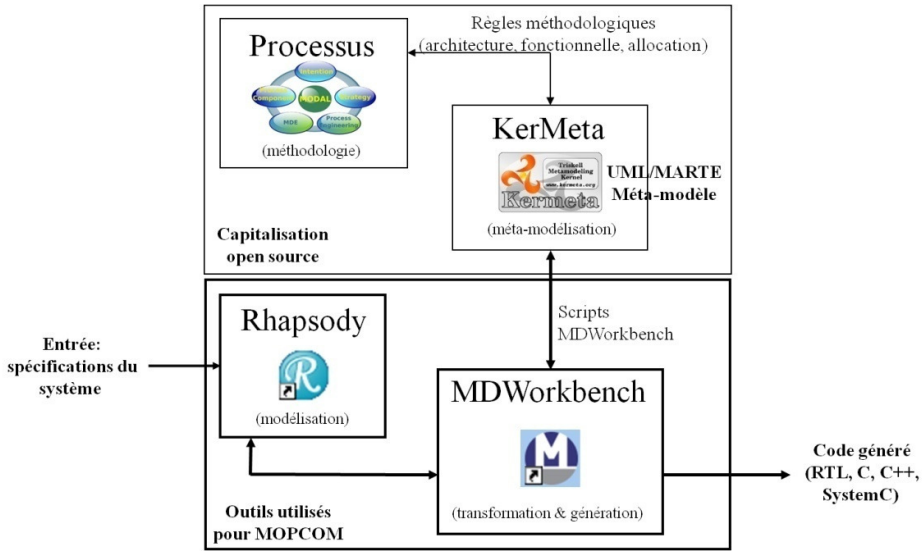


Figure 35. Environnement d'outillage de la méthodologie MOPCOM.

La partie haute de la Figure 35 illustre la partie de l'environnement uniquement utilisée dans le cadre du développement de la méthodologie en elle-même :

- **Le langage MODAL** [102] : utilisé pour formaliser la méthodologie MOPCOM sous la forme d'un méta-modèle ;
- **La syntaxe Kermeta** [43] : utilisée pour définir les contraintes d'utilisation des méta-éléments d'UML et du profil MARTE associé à chaque modèle de MOPCOM.

Ces deux outils sont totalement transparents pour l'utilisateur de la méthodologie MOPCOM.

La partie basse de la Figure 35 illustre les deux outils que les équipes de développement auront besoin d'utiliser dans le cadre de la mise en œuvre de la méthodologie MOPCOM :

- **Un modeleur UML** : principal outil qui supporte le développement des modèles UML/MARTE en respectant la méthodologie MOPCOM. L'outil en question doit supporter le profil MARTE (si ce n'est pas le cas il sera nécessaire d'intégrer le profil à l'outil) ;
- **Un outil de transformation** : outil qui permet de transformer des modèles vers d'autres modèles. Cet outil permet à la fois de développer un jeu de règles de transformation et est également le support de la transformation en elle-même.

De plus, la modélisation de système temps réel embarqué requiert l'utilisation d'un langage d'action pour la description bas niveau afin de compléter la sémantique de haut niveau d'UML et des diagrammes associés. Ce langage d'action permet également de spécifier le corps des opérations, les gardes/triggers des transitions et des états. Le langage d'action choisi est un sous-ensemble du C++ complété par un ensemble de macros adressant, par exemple, l'envoi et la réception d'événements. Ce choix est essentiellement motivé par la flexibilité et l'extensibilité du langage ainsi que sa simplicité de prise en main.

Dans le cadre de la validation de la méthodologie MOPCOM, Rhapsody® [67] est utilisé comme modeleur UML, et MDWorkbench® [107] est utilisé comme outil de transformation et génération de code. Nous avons choisi Rhapsody, d'une part, de part la présence d'un langage d'action, et d'autre part, du fait qu'il offre la possibilité de simuler certains diagrammes UML (diagrammes de séquences et diagrammes d'activités). Finalement, cette propriété n'a pas été utilisée car trop fortement liée à l'outil et par conséquent à l'encontre des concepts de généricité prônés par le MDA. Concernant l'outil de transformation et génération de code, notre choix s'est porté sur MDWorkbench, d'une part, car un sous-ensemble de celui-ci intitulé RulesComposer® est intégré à Rational Rhapsody, et d'autre part, il est basé sur des standards.

Un outil de vérification automatique des modèles, intitulé ModelChecker, est également proposé, développé sur la base de Kermeta. Celui-ci est intégré à l'outil de modélisation UML. Il permet de vérifier la conformité des modèles développés UML/MARTE avec les contraintes MOPCOM. Cette vérification est réalisée par niveau de modélisation (AML, EML, DML) et s'effectue lors de la génération de code. L'outil de vérification de code est appelé avant la génération de code effective, ceci de manière totalement transparente pour l'équipe qui développe les modèles. Si le modèle n'est pas conforme aux règles imposées par la méthodologie MOPCOM, il est impératif de corriger celui-ci en adéquation avec les contraintes avant de passer à l'étape suivante.

La génération de code peut-être utilisée de 2 manières différentes :

- Directement appelée dans l'outil de modélisation Rhapsody : RulesComposer étant intégré dans Rhapsody, le code peut-être généré depuis le modeleur UML et le résultat est directement retourné dans l'outil de modélisation UML. L'utilisateur doit choisir le jeu de règles et le périmètre du modèle sur lequel s'appliquera cette génération de code ;
- En dehors du cadre de l'outil de modélisation : en exportant le modèle UML vers l'outil de transformation par le biais du langage XMI.

Conceptuellement, la mise en application de la méthodologie MOPCOM n'est pas contrainte à ces outils. Cependant, nous avons toutefois constaté qu'il n'est pas si évident de mettre en œuvre la méthodologie MOPCOM avec un autre modeleur UML. Du fait, d'une part de l'intégration de la norme UML et du profil MARTE, et d'autre part de l'intégration de l'outil de vérification des modèles basé sur les contraintes d'utilisation des méta-éléments de l'UML de du profil MARTE.

3.2 Le niveau de modélisation EML

Dans le cadre du développement de la méthodologie MOPCOM présentée, les travaux de cette thèse se sont plus particulièrement portés sur la définition du niveau de modélisation intermédiaire du flot : le niveau de modélisation EML pour *Execution Modeling Level* (cf. Figure 36).

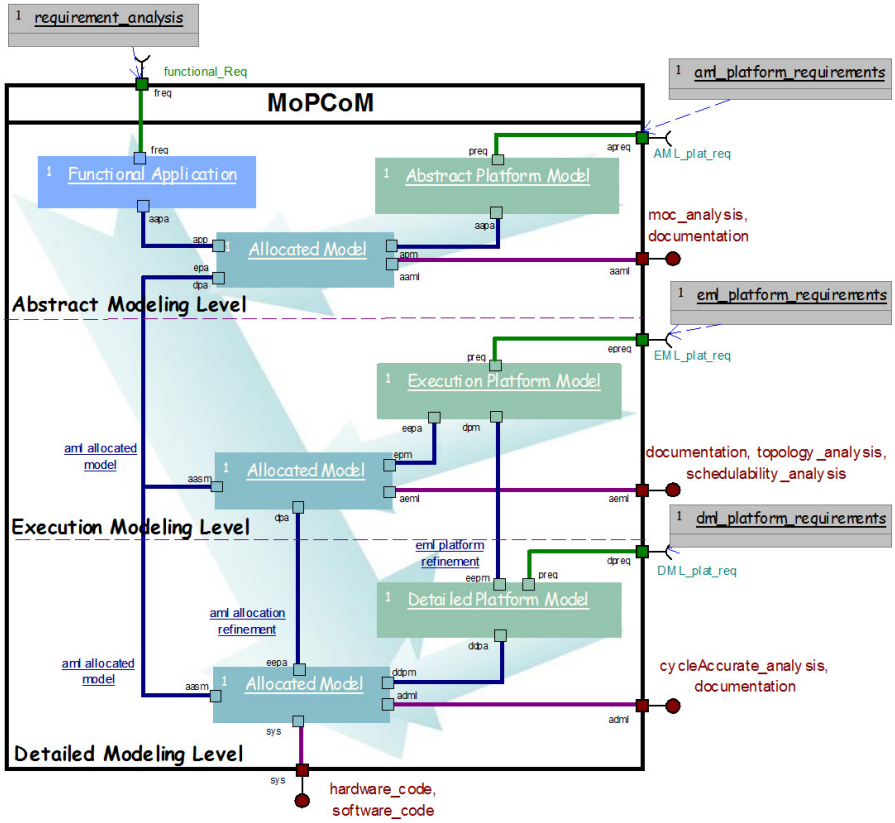


Figure 36. Positionnement du niveau de modélisation EML par rapport à la méthodologie MOPCOM.

Le but de ce niveau est de définir la topologie de la plate-forme d'exécution du système et de faire une analyse des communications, autrement dit il s'agit à ce niveau de définir une architecture matérielle supportant l'exécution du modèle résultant du niveau de modélisation AML.

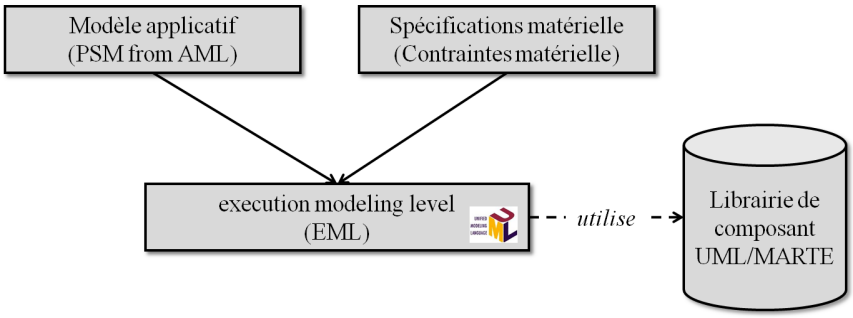


Figure 37. Les points d'entrée du niveau de modélisation EML.

Comme le montre la Figure 37, les points d'entrée du niveau EML sont principalement de deux ordres :

- Le modèle résultant du niveau AML : PSM du niveau AML, résultat de l'allocation des éléments du PIM sur les ressources de calcul virtuelles du PM ;
- Les spécifications/contraintes matérielles issues du cahier des charges du système.

Nous pouvons envisager dans le futur un autre point d'entrée : une librairie de composants UML/MARTE. Ceci permettrait de limiter le temps passé au développement du modèle UML et de se focaliser principalement sur l'exploration de l'architecture pour aboutir au choix optimal.

A ce niveau, le modèle de l'architecture fait apparaitre des nœuds de calcul, de communication et de mémorisation connectés par des bus de communications mettant en œuvre des protocoles de haut niveau. Les différentes ressources de la plate-forme offrent des services, définis notamment par des qualités de services et des caractéristiques temporelles, pour exécuter l'application. A ce niveau de modélisation, les caractéristiques temporelles des ressources de la plate-forme matérielle sont des caractéristiques que nous pouvons qualifier de gros-grain, ce qui veut dire qu'à ce niveau les caractéristiques temporelles ne sont pas dans le domaine bit-accurate. Par exemple, les ressources de la plate-forme matérielle modélisées sont caractérisées par des vitesses d'exécution relatives (caractéristiques temporelles non bit-accurate).

A partir du modèle résultant de l'allocation de l'application sur les ressources de la plate-forme matérielle, i.e. le PSM du niveau EML, une analyse de l'ordonnancement des tâches et des performances temporelles du système peut-être réalisée. Pour cela, les contraintes temps réel sont vérifiées à l'aide d'une horloge idéale (utilisation du patron singleton), qui permet d'incrémenter le temps, élément issu du paquetage Time de MARTE. Suivant les résultats obtenus, des ajustements, des raffinements, voire une refonte de l'architecture de la plate-forme sont alors effectués si nécessaire. Il est également envisageable de revoir l'architecture du modèle applicatif (PIM). Ceci de manière itérative jusqu'à obtenir une topologie de la plate-forme permettant l'exécution de l'applicatif en respectant les contraintes temporelles imposées au système.

De toute évidence, la définition de l'architecture de la plate-forme d'exécution et l'allocation des fonctions applicatives sur les ressources matérielles passe par un compris coût/performances.

A noter que nous n'avons pas travailler sur la modélisation de système d'exploitation temps réel (RTOS), tel que VxWorks[11], couche servant d'interface entre une partie (ou l'ensemble) de l'applicatif et la plate-forme matérielle. Il s'agit d'une piste potentielle de travail pour faire évoluer la méthodologie d'autant plus que le profil MARTE propose des éléments pour modéliser des RTOS.

3.2.1 Organisation du niveau EML

Chacun des trois niveaux de modélisations AML, EML et DML sont répartis dans 3 paquetages stéréotypés du même nom.

Note : Ces stéréotypes font partie intégrante du profil MOPCOM (cf. Figure 38). Ce profil simple est constitué d'éléments pour formaliser le processus, et d'éléments de sémantiques non définis dans MARTE.

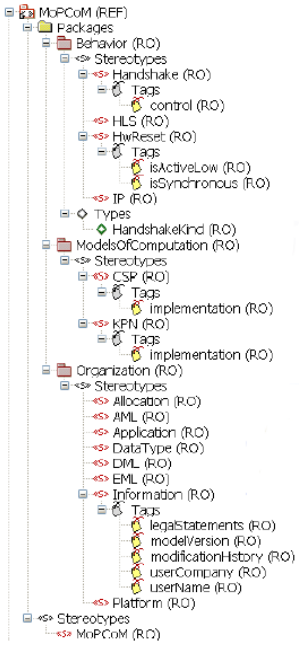


Figure 38. Illustration du profil MOPCOM.

Le profil MOPCOM est composé :

- D'éléments pour l'organisation des modèles du processus ;
- D'éléments liés au modèle de calcul utilisé au niveau AML : CSP, KPN ;
- D'éléments liés au comportement des modèles : HLS (*High Level Synthesis*), IP, reset, Handshake.

[11] VxWorks : système d'exploitation temps réel pour les systèmes embarqués de Wind River.

Tous les éléments sont définis sous la forme de stéréotypes, applicables sur un projet MOPCOM. Ce profil est accessible par défaut lors de la création d'un nouveau projet de type MOPCOM, tout comme le sont les profils SysML et MARTE.

Comme pour les deux autres niveaux (AML et DML), le niveau EML est constitué principalement de 3 modèles distincts : PIM, PM et PSM. Ces trois modèles différents sont répartis dans 3 paquetages :

- Un paquetage stéréotypé « *Application* » : qui contient le modèle PSM résultant du niveau AML ;
- Un paquetage stéréotypé « *Platform* » : qui contient le modèle PM ;
- Un paquetage stéréotypé « *Allocation* » : qui contient le modèle PSM et ses contraintes associées.

Note : il est possible d'avoir autant de paquetages Platform que de définitions de topologie de la plate-forme matérielle, et autant de paquetages Allocation que de différents modèles alloués. Ces derniers n'ont aucun lien entre eux.

3.2.2 Définition du niveau EML

Le diagramme de cas d'utilisation illustré par la Figure 39 met en évidence les actions à mener pour définir le niveau EML.

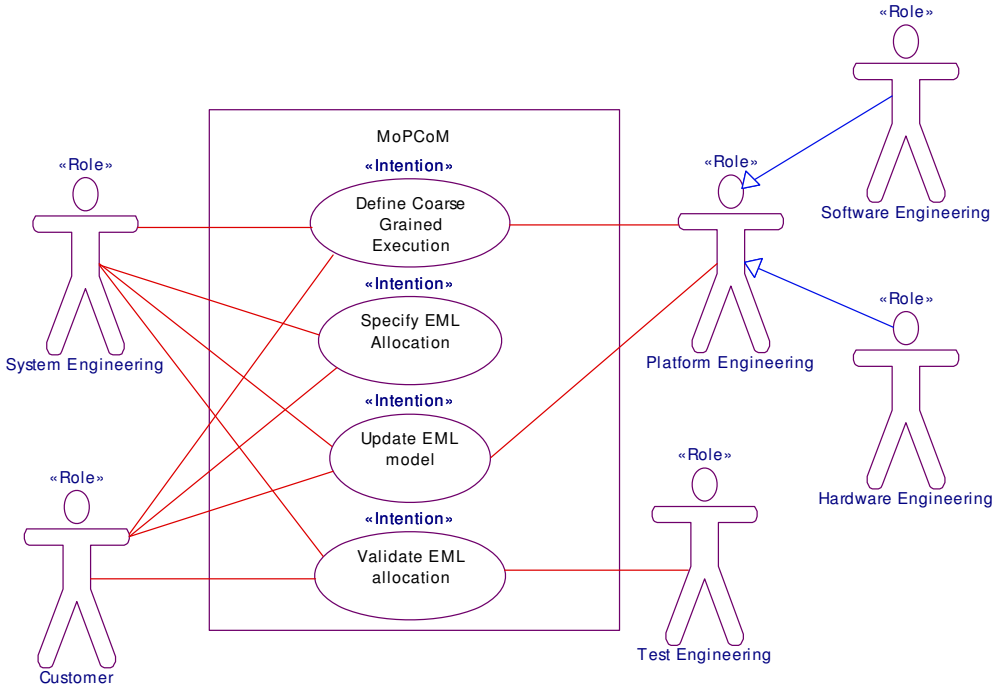


Figure 39. Diagramme de cas d'utilisation, définition des intentions du niveau EML.

3.2.2.1 Définition du modèle PIM

Vous pouvez constater, sur le diagramme de la Figure 39, qu'aucun cas d'utilisation ne fait apparaître la définition du modèle PIM du niveau EML. En effet, le modèle PIM de ce niveau de modélisation n'est autre que le modèle résultant du niveau AML : **PIM du niveau EML = PSM du niveau AML**, car le modèle de la plate-forme matérielle du niveau AML est purement abstraite. Toutefois, pour les besoins d'analyse de ce niveau, analyse d'ordonnancement et de performances temporelles, comme nous l'avons précédemment indiqué, les développeurs peuvent être amenés à ajouter des contraintes telles que le temps de calcul estimé, associé à un traitement. Les sous-profils MARTE à utiliser sont les mêmes que ceux utilisés pour définir le PIM du niveau AML, c'est-à-dire, les sous-profils HLAM, SRM, GCM et NFPs. De plus, les équipes de conception peuvent être amenées à retoucher, voire complètement revoir, l'architecture de l'application suivant les résultats des analyses du niveau EML.

3.2.2.2 Définition du modèle PM

La première étape, représentée par le cas d'utilisation « *Define Coarse grained Execution* » sur le diagramme de la Figure 39, consiste principalement à définir le modèle PM du niveau EML. La définition du modèle PM est principalement menée par des ingénieurs hardware. Le méta-modèle MOPCOM formalise les étapes, sous la forme d'un diagramme d'activités (cf. Figure 40), à suivre pour aboutir au PM du niveau EML.

Dans l'EML, le modèle PM représente uniquement la topologie de la plate-forme d'exécution (plate-forme matérielle) basée sur des ressources/composants génériques de haut niveau. En effet, l'objectif de la plate-forme d'exécution virtuelle est de cacher, entre autre, les caractéristiques physiques de la plate-forme physique à l'application. Pour modéliser la plate-forme au niveau EML, les éléments de modélisation suivant sont utilisés :

- ressources/composants/éléments (ou encore nœuds) de calcul : composants maître/esclave qui offrent un, voire plusieurs, service(s) de calcul, ressources caractérisées par une vitesse d'exécution ;
- ressources de mémorisation : composants esclaves qui offrent des services de stockages ou/et de lecture de données, caractérisées par une capacité de stockage et une bande passante ;
- ressources de communications : composants supportant une communication de données entres ressources de calculs et ressources de mémorisation qui offrent des services de contrôle/d'arbitrage des accès concurrents (gestion de priorité). Les ressources de communications sont définies par leurs interfaces associées ;
- ports : composants de connexion des ressources de calculs et les ressources de mémorisation avec les ressources de communications ou directement entre elles sans passées par des ressources de communications. Les ports offrent des services d'envoi et de réception des données (définition des interfaces).

A ce niveau, le modèle PM est caractérisé par des annotations temporelles et principalement au niveau des communications.

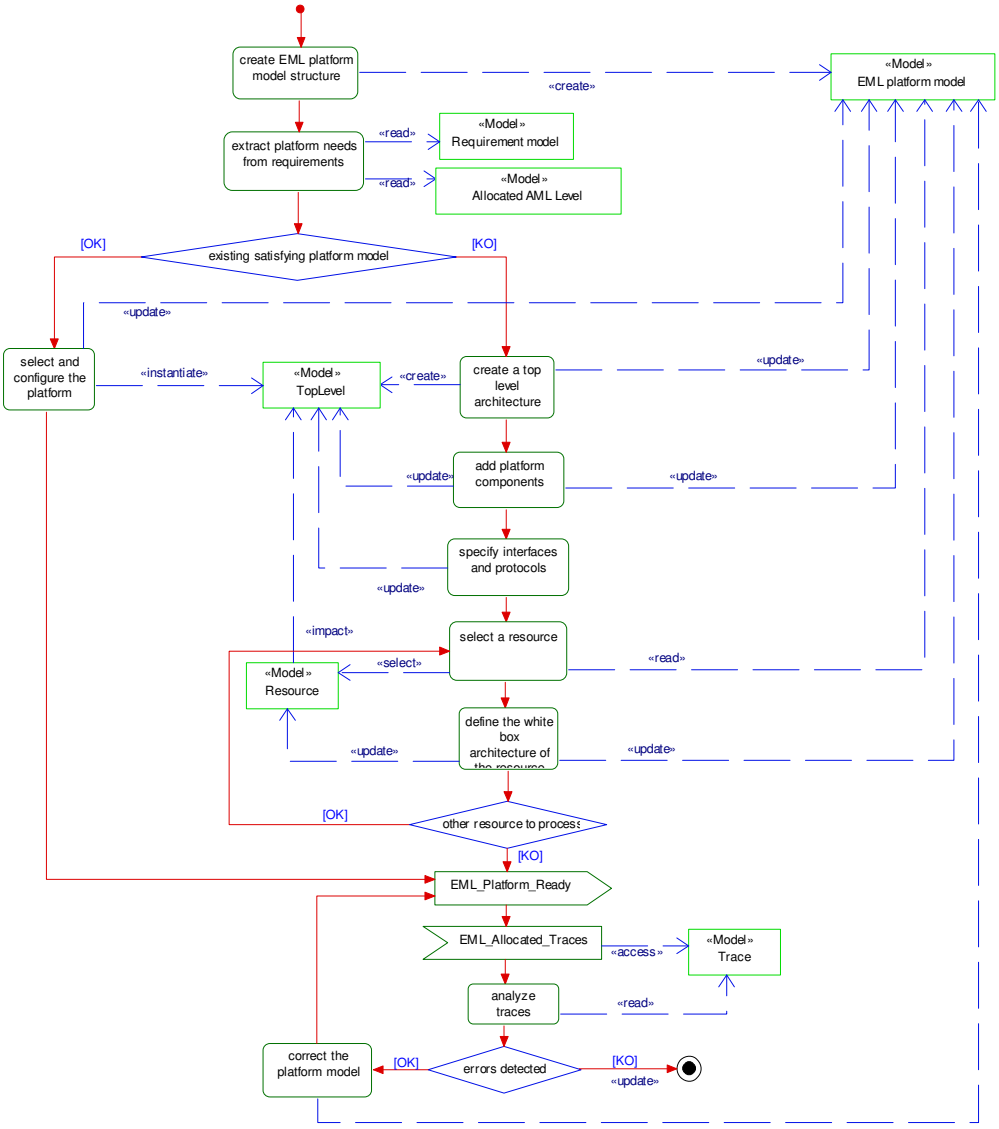


Figure 40. Diagramme d'activités illustrant la stratégie à suivre pour aboutir au modèle PM du niveau EML.

De plus, le modèle PM s'abstrait de détails superflus, pour ce niveau de modélisation et d'analyse, comme la description détaillée des protocoles de communications (bit-accurate), le type de ressources de calcul (micro-processeur, FPGA, etc) et des ressources de stockage (RAM, ROM, etc) utilisées dans le modèle de la plate-forme physique. En effet, le principal intérêt d'une telle modélisation est de représenter les nœuds de calcul, de stockage, de communication et les services offerts par la plate-forme à l'application. Les interfaces, représentation des services de communication entre les ressources de calculs et les ressources de stockage, sont modélisées à haut niveau : les données sont « bit-true » et les temps de transaction sont non bit-accurate. Le principal concept utilisé dans ce modèle PM du niveau EML est une modélisation de niveau transactionnel, comme promu par Gajski et la communauté SystemC en général. Ainsi les communications entre les composants de la plate-forme sont représentées par des appels de fonctions, et non pas par une modélisation détaillée des protocoles et des connectivités comme pour le modèle RTL.

Du fait que ce modèle PM ne fasse apparaitre qu'une topologie virtuelle de la plate-forme d'exécution, seuls les éléments du sous-profil GRM de MARTE peuvent être utilisés pour définir ce modèle. GRM met à disposition les éléments nécessaires et suffisants pour modéliser une plate-forme d'exécution à haut niveau. Les principaux éléments de GRM utilisés sont les suivants :

- « *ComputingResource* » : élément qui permet de modéliser une ressource de calcul ;
- « *StorageResource* » : élément qui permet de modéliser une ressource de stockage ;
- « *CommunicationMedia* » : élément qui permet de modéliser une ressource de communication ;
- « *TimingResource* » : élément qui permet de définir une horloge logique nécessaire pour l'analyse de performances du niveau EML ;
- « *DeviceResource* » : élément qui permet de modéliser une ressource externe au système qui communique avec le système modélisé, élément caractérisé par ses interfaces.

Note : les éléments du sous-profil HRM de MARTE ne doivent pas être utilisés. Pour ce faire, l'outil de vérification ModelChecker, cité dans la section 3.1.4, vérifie la conformité du modèle avec le méta-modèle du processus MOPCOM et indiquera une erreur si un élément du sous-profil HRM est utilisé.

3.2.2.3 Définition du modèle PSM

Une fois l'étape de définition du modèle PM (modèle de la plate-forme d'exécution) terminée, il convient de passer à l'étape intitulée « *Specify EML allocation* ». Le méta-modèle MOPCOM formalise les étapes, sous la forme d'un diagramme d'activités (Figure 41), à suivre pour aboutir au PSM du niveau EML.

Cette étape consiste à allouer les éléments du modèle PIM sur les composants (ressources matérielles virtuelles) du modèle PM. Cette étape met en jeu à la fois l'architecture applicative, l'architecture matérielle et les caractéristiques non-fonctionnelles du système. Par conséquent, celle-ci est menée conjointement par les équipes systèmes, logicielle et matérielle.

Le mécanisme d'allocation est semblable à celui mis en œuvre pour le niveau AML. A savoir, les éléments de l'application (PIM) sont alloués sur les ressources de la plate-forme d'exécution via des liens de dépendances stéréotypés « *Allocate* », issu du profil MARTE. De plus, les attributs *kind* et *nature*, associés à ce stéréotype, doivent être spécifiés afin de caractériser l'allocation. Ceux-ci permettent d'indiquer si l'allocation est spatiale ou temporelle, mais aussi si celle-ci est comportementale, structurelle ou hybride (comportementale et structurelle).

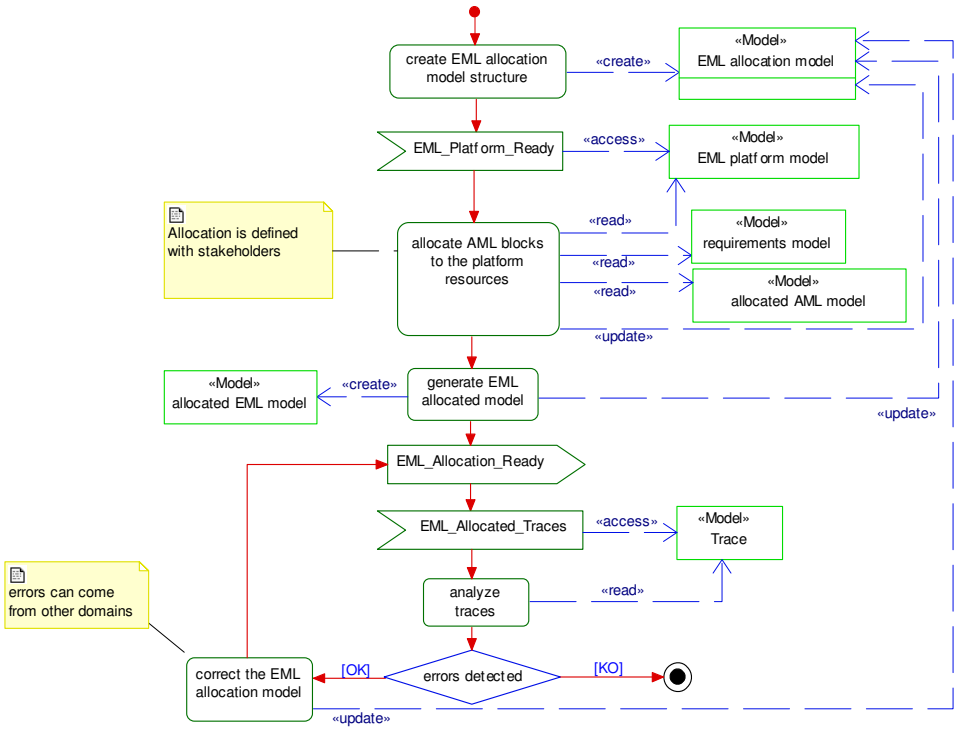


Figure 41. Diagramme d'activités illustrant la stratégie à suivre pour aboutir au modèle PSM du niveau EML.

Une fois cette étape réalisée, une transformation permet de générer le modèle PSM du niveau EML. La phase d'allocation et de transformation pour aboutir au modèle PSM doit respecter la sémantique des MoC régissant les communications entre les composants virtuels du PIM (du niveau EML). Pour cela, la transformation mise en place permet de gérer les cas de conflit en appliquant des transformations sur les machines d'état régissant le mécanisme des MoCs. La sémantique des MoCs, utilisés au niveau AML, est basée sur des communications point-à-point entre les ressources de calcul virtuelles. Lorsque deux composants applicatifs, communiquant en point-à-point, sont alloués sur la même ressource de calcul du PM, du niveau EML, la sémantique du MoC n'est pas modifiée. Par contre, si deux composants applicatifs communiquant entre eux point-à-point sont alloués sur deux ressources de calcul du PM, du niveau EML, la sémantique de communication du MoC qui régit cette communication n'est plus garantie. En effet, si les deux ressources de calcul du PM communiquent par le biais d'un bus, et non pas en point-à-point, il est impératif de garantir la sémantique du MoC même si la communication physique s'effectue par un bus partagé. C'est ici le point crucial de la transformation pour générer le PSM du niveau EML.

Bien qu'étant un des éléments du niveau EML cette transformation et tout particulièrement la gestion des MoCs a été développée par un de nos partenaires dans le développement du processus MOPCOM, Ali KOURDI, [104] associées au méta-modèle CoMeta [103].

Le modèle PSM obtenu ne se suffit pas à lui-même pour pouvoir en faire une simulation et effectuer des analyses d'ordonnancement et de performances, nécessaires pour valider la topologie de la plate-forme d'exécution et ainsi le niveau EML.

3.2.3 Validation du niveau EML

Comme nous l'avons déjà indiqué précédemment dans ce chapitre, la validation du niveau EML passe par une analyse de l'ordonnancement des tâches et une analyse des performances temporelles du PSM. L'analyse d'ordonnancement permet de fournir la capacité d'évaluer des contraintes temporelles et de garantir le bon fonctionnement du système dans les pires cas. Les traces fournies par cette analyse permettent de valider ou non la topologie de la plate-forme matérielle. Si les traces résultantes ne permettent pas de valider le respect des contraintes fonctionnelles et non-fonctionnelles il est nécessaire de modifier la topologie de la plate-forme d'exécution virtuelle, voire restructurer également l'architecture fonctionnelle (PIM). Pour cela, les traces de la simulation sont réinjectées dans le modèles EML pour guider les équipes à apporter les modifications nécessaires. Cette étape est itérative jusqu'à obtenir une topologie de la plate-forme d'exécution et un modèle PSM qui supportent et respects les contraintes du système. Ainsi, le modèle PM du niveau EML servira de point d'entrée du niveau DML et sera raffiné comme le montre la Figure 36.

Le diagramme d'activités, illustré par la Figure 42, spécifié dans le méta-modèle du processus MOPCOM, illustre la stratégie adoptée pour mener le cas d'utilisation « *validate EML allocation* » (cf. diagramme du cas d'utilisation Figure 39).

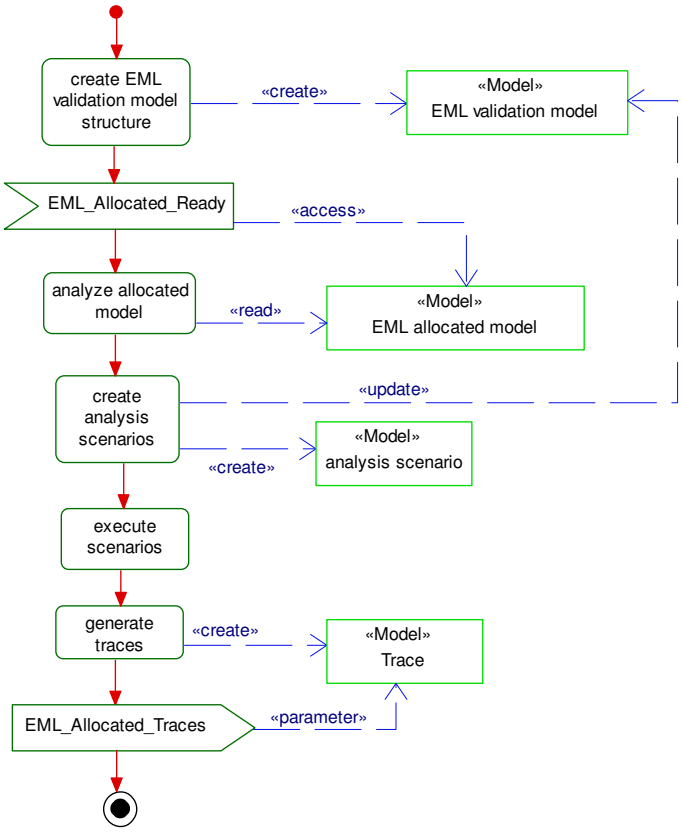


Figure 42. Diagramme d'activités illustrant la stratégie à suivre pour mettre en œuvre le cas d'utilisation « validate EML allocation ».

Les caractéristiques déterminantes permettant de valider les performances du système au niveau EML sont :

- la taille des bus et leur débit (bande-passante) ;
- la vitesse relative des ressources de calcul ;
- la taille des mémoires.

La validation du PSM du niveau EML consiste à annoter le modèle d'allocation avec des informations portant sur les aspects temps réel du système (contraintes d'ordonnancement) et des contraintes temporelles associées. Le modèle annoté sert de base à l'élaboration de scénarios de test. De plus, les annotations sont exploitées afin de générer des vecteurs de test pertinents.

Nous préconisons d'utiliser les éléments du sous-paquetage GQAM de MARTE associé aux sous-paquetages SAM et PAM.

Pour l'analyse d'ordonnancement, le sous-paquetage SAM est recommandé. SAM offre les éléments pour ajouter des annotations dans les différentes vues du modèle afin d'évaluer l'ordonnancement. Ci-après nous décrivons les étapes à suivre pour ce sous-paquetage et ainsi aboutir à un modèle EML près pour l'analyse d'ordonnancement :

- Dans le modèle PIM, les descriptions comportementales sont annotées par des contraintes de temps et par la taille des messages échangés avec les stéréotypes du sous-paquetage SAM. Le diagramme de séquence est bien adapté pour ajouter ces annotations. Les différents scénarios de description comportementale forment le modèle « Workload behavior » ;
- Dans le modèle PSM, le diagramme d'objets est annoté pour indiquer le type des ressources d'ordonnancement de chaque élément du modèle. Les entités applicatives sont stéréotypés «schedulableResource», les nœuds de calcul du PIM sont stéréotypés «SaExectHost», les nœuds de communication (bus) sont stéréotypés «SaCommHost» ;
- La dernière étape consiste à définir des scénarios d'analyses. Pour chaque scénario, stéréotypé «gaScenario» issu du paquetage SAM, le contexte et les paramètres doivent être modélisés. Le modèle représentant le contexte d'analyse, stéréotypé «saAnalysisContext», précise les conditions de début et d'arrêt des différents scénarios. La modélisation des paramètres d'un scénario est une instance du modèle du contexte d'analyse dans lequel les valeurs sont initialisées pour simuler l'ordonnancement.

Ces analyses ne peuvent se faire dans l'outil de modélisation UML. En effet des outils d'analyses d'ordonnancement tel que Cheddar [108] sont dédiés à cet effet. Cependant, il est nécessaire d'ajouter des informations aux modèles du niveau EML afin de réaliser de telles analyses comme nous venons de le présenter. Une autre solution pour réaliser ces analyses est de basculer dans l'environnement SystemC, qui permet de simuler conjointement applicatif et plate-forme d'exécution et se caractérise par une analyse transactionnelle du système. Le niveau de modélisation EML est assimilable au niveau de modélisation SystemC intitulé PV-T (*Programmer View-Timed*).

3.2.4 Les éléments de MARTE utilisés au niveau EML

Le Tableau 6 récapitule les éléments de du profil MARTE utilisés dans les différents modèles du niveau EML.

Tableau 6. Les éléments du profile MARTE utilisés dans les modèles du niveau EML.

Modèle	Eléments de MARTE (Paquetage::stéréotype)
Modèle applicatif (PIM)	**HLAM::RteUnit** – élément qui caractérise des composants à caractères d'exécution concurrentes **HLAM::RteConnector** – élément qui permet de caractériser un connecteur entre éléments RteUnit
Modèle de la plate-forme matérielle (PM)	**GRM::ComputingResources** – élément de modélisation d'une ressource de calcul **GRM::StorageResources** – élément de modélisation des méméoires **GRM::CommunicationMedia** – élément de modélisation des bus à au niveau **GRM::TimingResource** – élément de modélisation d'horloge logique. **GRM::DeviceResource** – élément de modélisation d'une ressource externe au système se caractérisant par ses interfaces **VSL** – le paquetage Value Specification Langage est utilisé pour ajouter des contraintes d'allocation du PIM sur le PM
Modèle alloué (PSM)	**Alloc::Allocate** – élément qui permet de caractériser un lien de dépendance en indiquant l'allocation d'un composant applicatif sur une ressources du PM **VSL** – le paquetage Value Specification Langage est utilisé pour ajouter des contraintes d'allocation du PIM sur le PM
Modèle d'analyses	**GQAM** – les éléments de ce paquetage permettent aux concepteurs de représenter les scénarios d'analyses focalisés sur l'ordonnancement des tâches et des performances temps temporelles **SAM** – ce paquetage met à disposition des éléments pour l'analyse d'ordonnancement des tâches du système **PAM** – ce paquetage met à disposition des éléments pour l'analyse des performances d'exécution du système

3.2.5 Les contraintes de développement

Comme pour les deux autres niveaux de modélisation, la définition du modèle EML est soumise à un certain nombre de contraintes en plus de celles imposées par l'utilisation d'UML et du profil MARTE (à chaque élément du profil sont associées des contraintes d'utilisation). Celles-ci sont automatiquement vérifiées par l'outil ModelChecker (présenté succinctement dans la section 3.1.4). Si le modèle EML ne respecte pas toutes les contraintes, celui-ci doit être modifié en conséquence pour passer ce test et ensuite poursuivre le développement et notamment pouvoir générer du code à partir de ce niveau. Le Tableau 7 présente les principales contraintes de ce niveau : certaines de ces contraintes sont liées à l'organisation du modèle, d'autre sont liées à la sémantique du modèle.

L'ensemble des contraintes associées au niveau EML sont décrites dans l'Annexe 2 (extrait du document intitulé « *Metamodels specification - Annex 1 - Constraints added by Mopcom* » du projet ANR MOPCOM SoC/SoPC). Ce document fait également apparaître la mise en œuvre Kermeta (car l'outil ModelChecker est basé sur l'outil Kermeta) de ces contraintes.

Tableau 7. Les principales contraintes du niveau EML, extrait de l'annexe 2.

Nom de la contrainte	Description
existsEMLPackage	Un projet basé sur la méthodologie MOPCOM doit contenir un paquetage EML
existsApplicationPackageInEMLPackage	Le paquetage EML doit contenir un sous-paquetage Application dans lequel sera modélisé le PIM
existsPlatformPackageInEMLPackage	Le paquetage EML doit contenir au moins un sous-paquetage Platform. Il doit y avoir autant de sous-paquetage Platform que de modèle de plate-forme d'exécution différentes
existsAllocationPackageInEMLPackage	Le paquetage EML doit contenir au moins un sous-paquetage Allocation. Il doit y avoir autant de sous-paquetage Allocation qu'il y a d'allocation différentes
existsComputingResourceInEMLPlatformPackage	Les modèles définis dans les sous-paquetages Platform doivent avoir un composant de top-niveau stéréotypé « *ComputingResource* »
existsLogicalClockInEMLPackage	Le niveau EML doit contenir une horloge logique, nécessaire pour l'analyse
noHwPhysicalLayoutElem	Le modèle EML ne doit en aucun cas faire apparaître une élément du sous-paquetage HRM::HwPhysical

3.2.6 Des métriques d'évaluation

Outre l'analyse d'ordonnancement et de performances, la génération de métriques issues du modèle permettent de donner des indications sur la qualité du modèle tant sur le plan quantitatif que qualitatif. Pour cela, nous proposons une liste non-exhaustive (cf. Tableau 8) de métriques quantitatives générées à partir du modèle EML. Celles-ci permettront d'évaluer la complexité des modèles UML en terme d'éléments d'UML et de MARTE utilisés. Pour chaque métrique, il est précisé, si nécessaire, le méta-type auquel elle fait référence.

Tableau 8. Métriques extraites des modèles du niveau EML.

Métrique EML	Description
EML Diagrams Number	Number of diagrams in model
EML Transitions Number	Number of transition in the model
EML State Number	Number of states in model
EML Allocation Number	Number of allocation links in model
EML Associations Number	Number of associations in model
EML Realizations Number	Number of realizations in model
EML Inherance Depth	Average depth of inheritance in model
EML Inheritance Number	Number of inheritance in model
EML Depth of Hierarchy Packages	Depth hierarchy of packages in model
EML Operations Number	Number of operations in model
EML FSM Number	Number of FSM in model
EML Attributs Number	Number of attributs in model
EML Port Number	Number of ports in model
EML Signal Types	Types of Signal in model
EML Data Types	Types of Data in model
EML Event Types	Type of events in model
EML Interface Number	Number of Interfaces in the model
EML Packages Number	Number of packages in model
EML MARTE stereotypes used Number	Number of stereotypes of MARTE used in model

A partir de ces métriques il est possible d'évaluer la qualité du modèle développé, mais aussi de détecter de mauvaises pratiques mises en œuvre dans le modèle. Cependant, l'analyse de celle-ci est aujourd'hui totalement manuelle et donc fastidieuse.

Une des perspectives que nous envisageons est de proposer des comparaisons de plusieurs métriques entre elles, des comparaisons de métriques par rapport à des seuils critiques (préalablement fixés par l'utilisateur) et des comparaisons avec combinaisons multiples (métriques et seuils) automatiques. Il existe déjà de nombreux travaux autour de l'évaluation de modèle UML par le biais de métriques. Nous pouvons notamment citer les travaux de BOOCH, MOOD, et CHIDAMBER [109] sur les métriques d'évaluation des systèmes orientés objets. De plus, il existe également des outils permettant d'évaluer les modèles UML tel que l'outil UMLQUALITY développé par la société Qualixo [110].

Dans le cadre du développement d'un modèle avec la méthodologie MOPCOM, l'outil MDWorkbench [107] est utilisé pour extraire automatiquement les métriques des modèles UML, capturés dans Rational Rhapsody [67], à travers le connecteur On-Demand.

Note : nous retrouvons également certaines des métriques extraites du niveau EML dans les deux autres niveaux de modélisation du processus (AML et DML).

3.3 Mise en œuvre et évaluation de la méthodologie

Comme nous l'avons indiqué dans le chapitre 1, pour valider la définition de la méthodologie proposée et éprouver celle-ci nous nous sommes appuyés sur le développement d'un système de transmission sans-fil bidirectionnel, basé sur la technologie MIMO, et s'exécutant sur une plate-forme matérielle de type SoPC. Ce système est issu des travaux de recherche dans le domaine des communications de Technicolor. Le système en question est présenté dans le chapitre 1. Le système étant complexe, nous nous sommes concentrés sur le décodage MIMO de la couche physique du système (cf. Figure 10). Nous rappelons ci-dessous (cf. Figure 43) le synoptique du décodage qui comporte principalement 4 parties : un prétraitement de la matrice canal, un prétraitement des données IQ réceptionnées, l'algorithme de décodage MIMO en tant que tel (stack decoder) et un demapper.

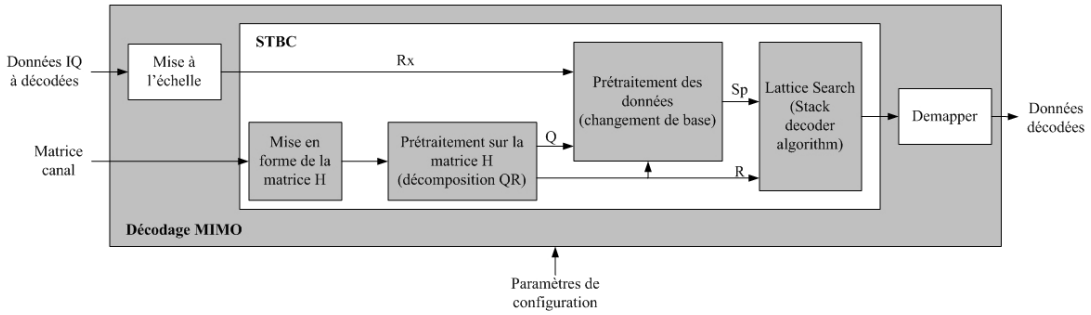


Figure 43. Synoptique du décodeur MIMO de la couche physique du système de transmission sans-fil.

Les données d'entrée du système sont de trois ordres :

- Matrice canal H : modélisation du canal de transmission ;
- Données IQ en réception à décoder ;
- Paramètres de configuration de la modulation (QPSK, 16QAM, 64QAM, etc).

Nous avons donc procédé à la modélisation UML du système décodeur MIMO en suivant la méthodologie MOPCOM.

3.3.1 Le niveau AML

Comme le stipule la méthodologie MOPCOM, le premier niveau de modélisation consiste à modéliser une architecture fonctionnelle du système et choisir une sémantique de communication entre les différentes entités applicatives.

3.3.1.1 Le modèle fonctionnelle : PIM

Nous avons tout d'abord développé le modèle fonctionnel du niveau AML à partir du cahier des charges. La Figure 44 illustre une des vues du PIM du niveau AML. Il s'agit du diagramme de classe de plus haut niveau hiérarchique du système. Cette vue fait apparaître le système en lui-même, le décodeur MIMO, et les acteurs qui interagissent avec celui-ci.

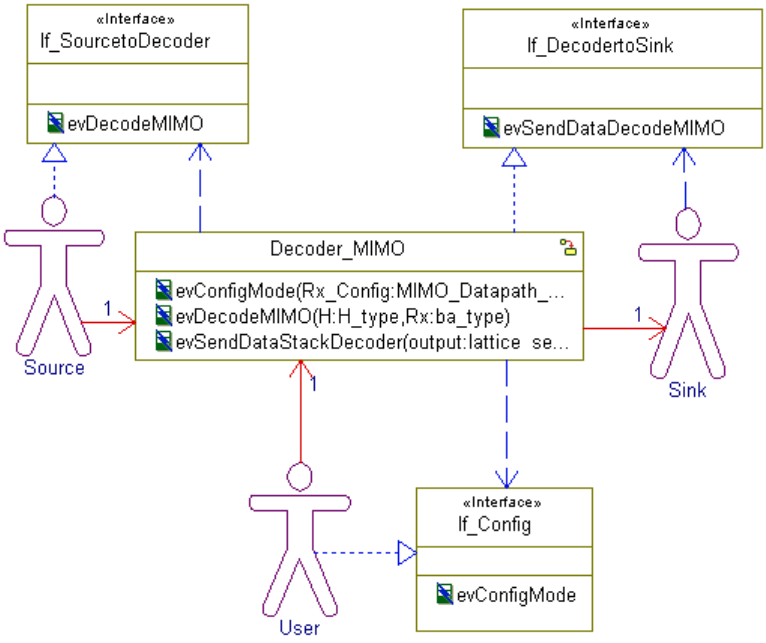


Figure 44. Vue hiérarchique de plus haut niveau du système, diagramme de classes du décodeur MIMO illustrant les interfaces avec les acteurs du système.

Le diagramme de classe fait évidemment apparaître comme acteurs les entrés/sorties identifiées sur le schéma fonctionnel de la Figure 43 :

- L'acteur *Source* : modélise la matrice canal H et les données Rx (au formant IQ) et réalise l'interface *IfSourcetoDecoder* qui permet au décodeur de réceptionner ces deux entrées ;
- L'acteur *Sink* : modélise les donnés Rx décodées en sortie du système. Il est associé à l'interface *If_DecodertoSink*.
- L'acteur *User* : modélise l'utilisateur qui peut jouer sur le système en paramétrant celui-ci à l'initialisation. Cet acteur réalise l'interface *If_Config* qui modélise l'action de configuration par un évènement.

Le décodeur MIMO est lui modélisé par la classe *Decoder_MIMO*. Cette dernière se caractérise par des événements de réception et d'émission des signaux allant et venant vers les acteurs et par une machine d'état (symbolisée en haut à droite de la classe) qui régit le séquencement des étapes du décodage.

Le découpage architectural du décodeur MIMO fait apparaître plusieurs fonctionnalités distinctes. Le diagramme d'objets, illustré par la Figure 45, une des vues du modèle PIM représentant l'architecture fonctionnelle du décodeur MIMO, fait apparaitre la modélisation des différentes fonctionnalités du système.

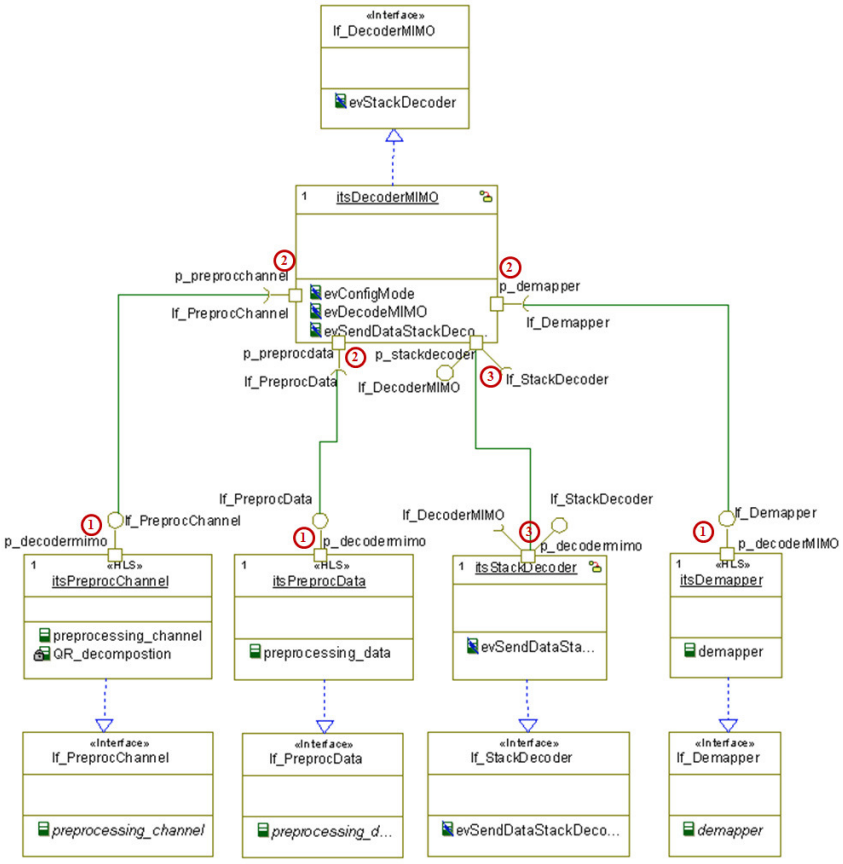


Figure 45. Le diagramme d'objet du PIM, une des vues illustrant la modélisation de l'architecture fonctionnelle du décodeur MIMO.

Le diagramme fait apparaître des objets modélisant les 4 principales fonctionnalités du décodage MIMO. Les objets *itsPreprocChannel*, *itsPreprocData* et *itsDemapper* offrent des services de calcul algorithmique par le biais de leurs interfaces associées qu'ils implémentent. Les communications entre l'objet *itsDecoderMIMO*, instance de la classe *Decoder_MIMO*, représentation hiérarchique la plus haute du système, et ces 3 objets sont unidirectionnelles au travers de ports. Les ports des 3 objets en questions offrent les services (1) de leurs interfaces associées et les ports de l'objet *itsDecoderMIMO* requièrent ces mêmes interfaces (2).

Au contraire des 3 autres fonctionnalités l'objet *itsStackDecoder* n'offre pas directement de service au décodeur MIMO. En effet, cet objet est composé de sous-fonctionnalités qui elles-mêmes sont représentées par des objets offrant des services de calcul algorithmique. La communication entre l'objet *itsSackDecoder* et *itsDecoderMIMO* est bidirectionnelle comme le montrent les ports *p_decodermimo* et *p_stackdecoder* (3), chacun offrant un service et en requérant un. Il s'agit dans les deux cas d'événements avec des paramètres.

La Figure 46 illustre la vue hiérarchique de la fonctionnalité stack decoder sous la forme d'un diagramme d'objets. Le principe est le même que le niveau hiérarchique supérieur illustré précédemment (cf. Figure 45).

L'algorithme de décodage MIMO (stack decoder) utilisé est décomposé en 3 sous-fonctions modélisées respectivement par les objets *itsSonordering*, *itsChildComputation* et *itsStackInsertion* offrant chacun un service par le biais de leur interface à l'objet *itsStackDecoder*. Ce dernier régit le séquencement de l'algorithme de décodage par une machine d'état illustrée par la Figure 47.

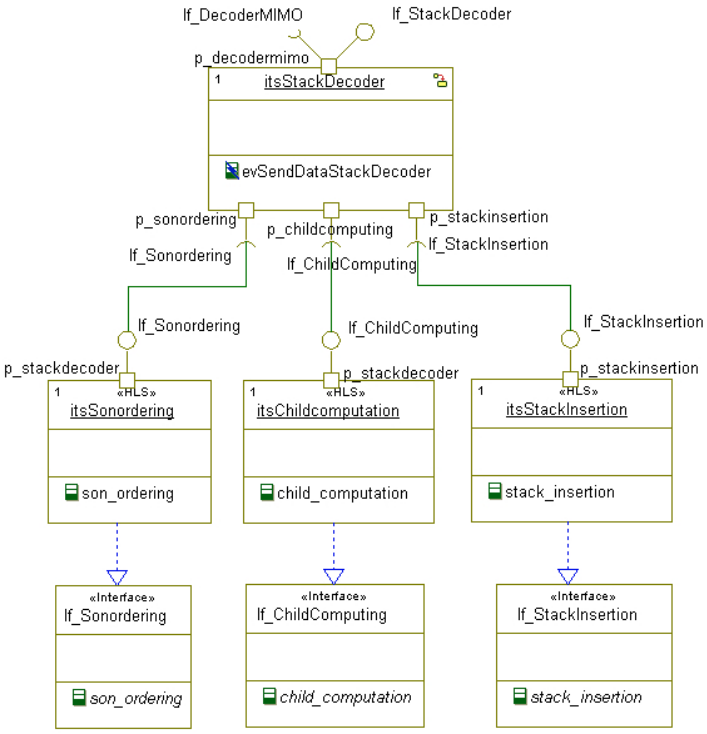


Figure 46. Diagramme d'objets illustrant la modélisation de la fonctionnalité Stack decoder.

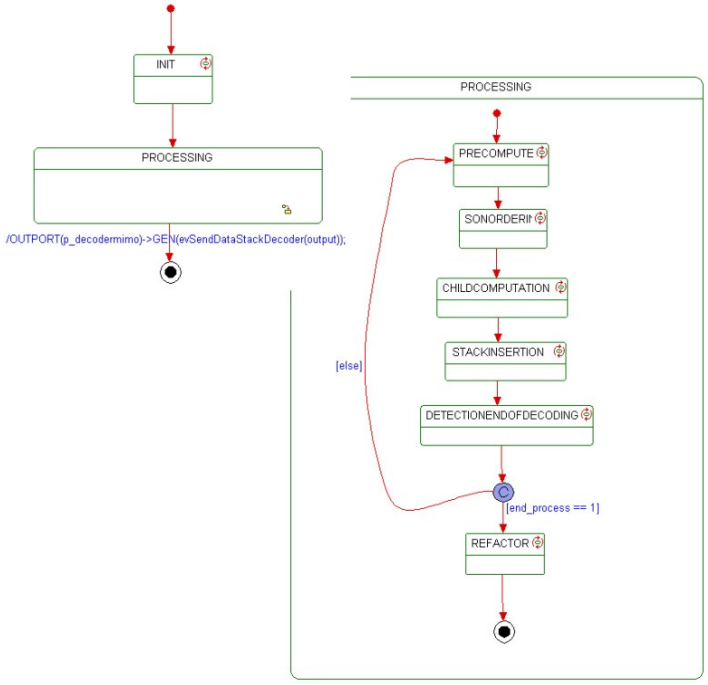


Figure 47. Diagramme d'état représentant la machine d'état séquençant l'algorithme de décodage Stack decoder.

L'état *PROCESSING* est activé en réception de l'événement *evStackDecoder* provenant de l'objet *itsDecoderMIMO*. Le sous-état *PRECOMPUTE* se définit par le déclenchement d'une action une fois entrée dans cet état. Le comportement de cet état est décrit par le langage d'action de l'outil Rational Rhapsody, sous-ensemble du C++ (cf. Figure 48).

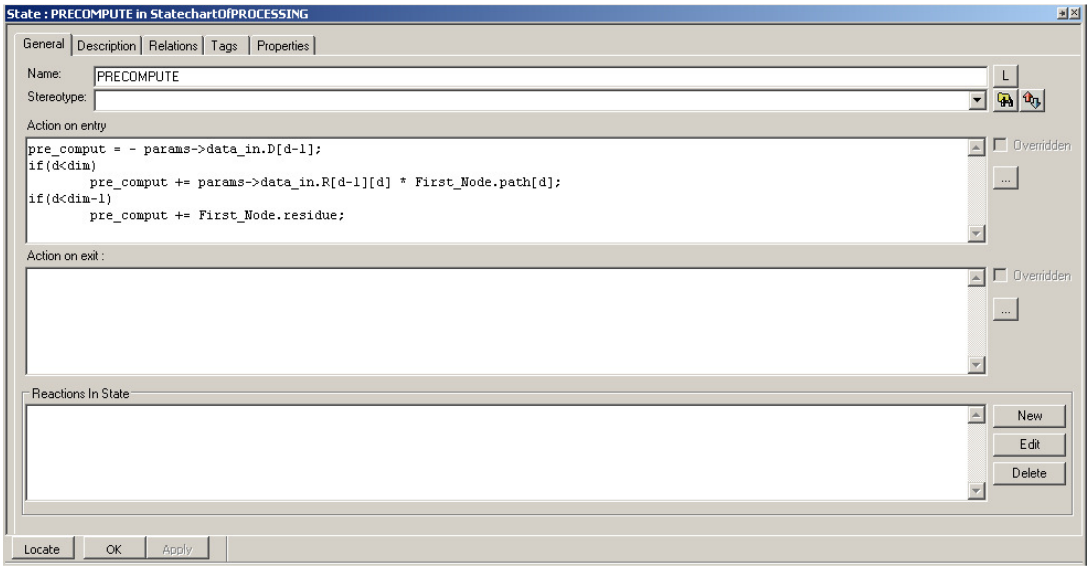


Figure 48. Description de l'algorithme d'un état à l'aide du langage d'action de l'outil de modélisation Rational Rhapody.

La Figure 49 illustre l'appel d'un service depuis un objet au travers d'un de ses ports à un autre objet offrant le service en question.

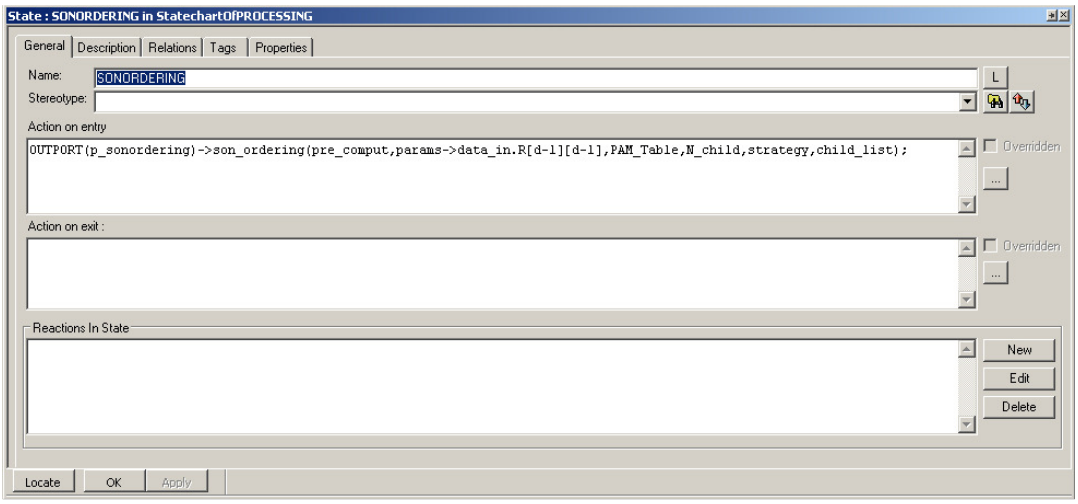


Figure 49. Description de l'appel d'une fonction au travers d'un port d'un objet, appel effectué dans l'objet itsStackDecoder, fonction dont le service est fourni par l'objet itsSonordering.

L'exemple de la Figure 49 montre l'appel du service *son_ordering* définit dans l'objet *itsSonOrdering*. Cet appel est effectué dans l'état *SONORDERING* de la machine d'état de l'objet *itsStackDecoder*. Et cet appel se fait par le biais du port *p_sonordering*, de l'objet *itsStackDecoder*, connecté au port *p_stackdecoder* de l'objet *itsSonOrdering*. Le service *son_ordering* est modélisé par une méthode dont la mise en œuvre est décrite à l'aide du langage d'action (sous ensemble du C).

Note : toute classe réagissant à un événement est une classe active, alors que les classes offrant des services (modéliser par des méthodes) sont des classes passives.

3.3.1.2 La plate-forme d'exécution virtuelle : PM

A ce niveau de modélisation, le modèle de la plate-forme d'exécution est totalement virtuel et fait totalement abstraction de données physiques de la plate-forme finale. L'objectif est ici de définir un cadre d'exécution pour l'applicatif et de choisir la sémantique des communications entre les entités du PIM. Les concepts appropriés sont nécessaires pour modéliser les communications uniquement point-à-point à ce niveau : les modèles de calcul (MoC). Nous rappelons qu'un MoC définit la sémantique des communications entre deux entités concurrentes. MOPCOM propose une librairie de MoCs (CSP et KPN), modélisés à l'aide du méta-modèle CoMeta [103]. De plus, le profil MARTE fournit tous les éléments nécessaires à la modélisation d'entités concurrentes ayant des caractéristiques temps réel : le stéréotype «RtUnit» issu du paquetage HLAM. Les communications entre les entités stéréotypées «RtUnit» sont point-à-point et modélisées par des connecteurs, stéréotypés «RteConnector», élément du paquetage HLAM. Les ports et les connecteurs mettent en œuvre les primitives de haut niveau qui supportent les mécanismes de communication.

La figure Figure 50 illustre le PM du niveau AML qui permet d'offrir au PIM un cadre d'exécution pour sa simulation et sa validation.

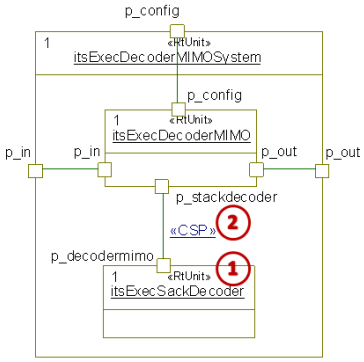


Figure 50. Diagramme de composant du PM du niveau AML.

Dans le cas de notre système, le décodeur MIMO, le PM du niveau AML est simplement constitué de deux entités d'exécution concurrentes *itsExecDecoderMIMO* et *itsExecStackDecoder*, spécifiées par le stéréotype «RtUnit» (1) communiquant entre eux. Cette communication est régie par le MoC CSP, spécifié par le stéréotype CSP (2) du profil MOPCOM.

3.3.1.3 Le modèle alloué : PSM

Le troisième modèle MDA, du niveau AML, est le modèle alloué, modèle résultant de l'allocation des entités du PIM sur les composants d'exécution concurrent du PM. Aujourd'hui, dans la méthodologie proposée, l'allocation est manuelle, mais aidée par un outil de type GUI (*Graphical User Interface*). Figure 51 illustre l'allocation des objets du PIM sur les différents composants d'exécution concurrents du PM. Cette allocation est indiquée par un lien de dépendance stéréotypée «*Allocate*», stéréotype du paquetage Alloc du profil MARTE.

Nous précisons qu'il s'agit d'une allocation spatiale (par opposition à temporelle qui permet de spécifier une allocation dynamique) car l'application n'a pas de caractère reconfigurable. La Figure 51 met également en évidence le fait que le composant d'exécution «RtUnit» ne peut contenir un seul et unique objet actif (classe active), tel que le sont les objets *itsExecDecoderMIMO* et *itsExecStackDecoder* mais plusieurs objets passifs, tel que le sont les objets *itsPreprocChannel*, *itsPreprocData* et *itsDemapper*.

Pour valider l'architecture fonctionnelle et les fonctionnalités du système, le modèle PSM doit être simulé. Pour cela nous proposons de générer un modèle équivalent SystemC non-timé. Nous reviendrons sur cette étape dans le chapitre 5.

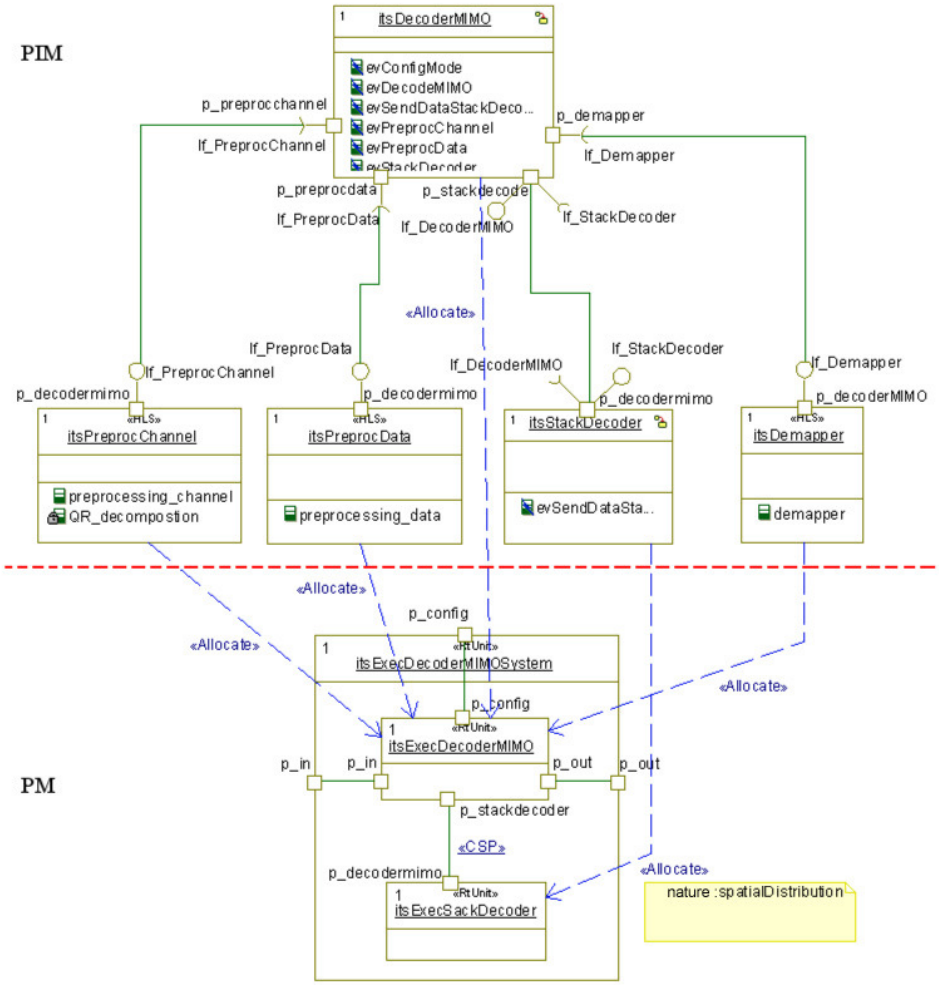


Figure 51. Vue du modèle AML illustrant l'allocation des objets du PIM sur les composants d'exécution concurrent du PM.

3.3.2 Le niveau EML

Une fois le niveau AML validé, il s'agit à présent de déterminer l'architecture d'une plate-forme matérielle supportant l'applicatif et les contraintes temps réel du système. En effet, comme nous l'avons expliqué dans la section 3.2.2.2 l'objectif du niveau EML est de modéliser la topologie d'une plate-forme d'exécution. Deux approches sont possibles :

- soit la plate-forme matérielle est déjà existante et imposée. Il s'agit alors simplement de modéliser celle-ci à haut niveau ;
- soit la plate-forme matérielle n'existe pas. Il est alors nécessaire de procéder à une exploration d'architecture pour aboutir au meilleur compromis, respectant le cahier des charges et supportant l'applicatif.

3.3.2.1 Le modèle fonctionnel : PIM

Le modèle fonctionnel PIM du niveau EML n'est autre que le résultat du niveau AML (PSM). La Figure 52 illustre le résultat de l'allocation des entités fonctionnelles du PIM du niveau AML sur les composants d'exécution concurrents du PM du niveau AML. Nous rappelons que ce modèle peut-être retouché suivant les résultats d'analyse du niveau EML.

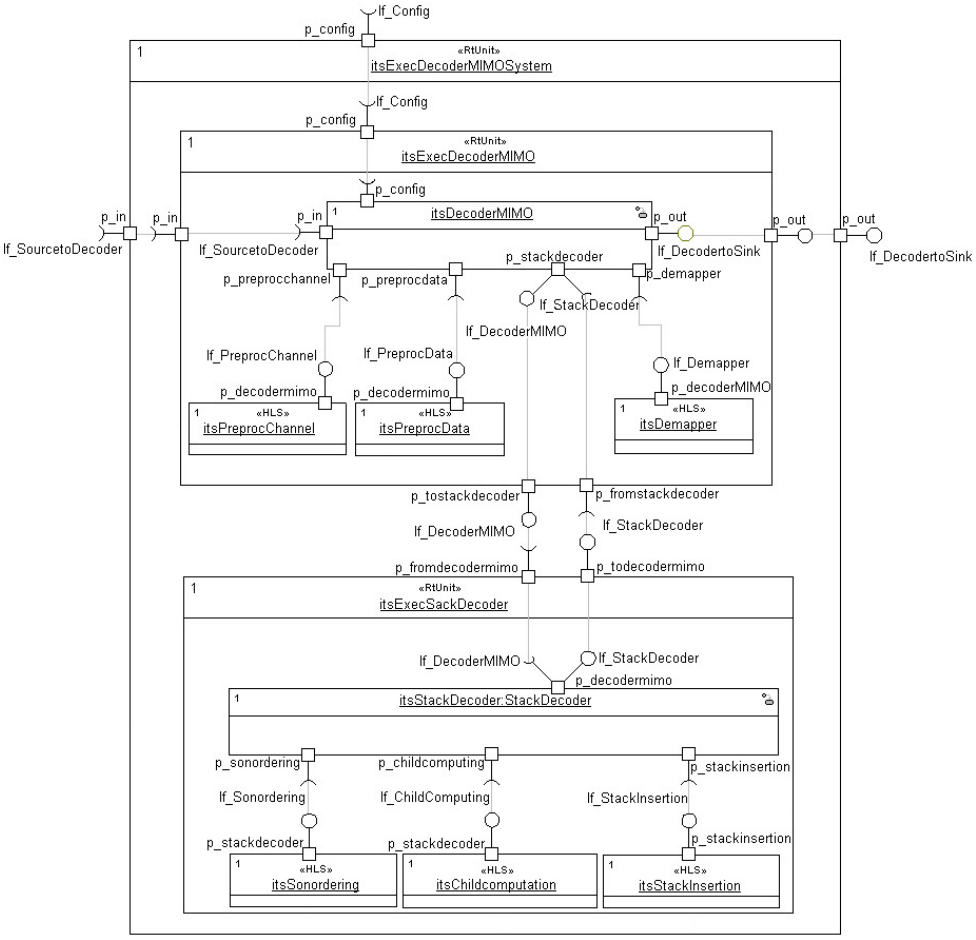


Figure 52. PSM du niveau AML et PIM du niveau EML.

3.3.2.2 Le modèle de la plate-forme matérielle : PM

Dans le cadre du système de transmission-sans fil de Technicolor, la plate-forme d'exécution est déjà spécifiée et existante : elle est basée sur un composant programmable FPGA. De ce fait, nous nous sommes trouvés dans le premier cas de modélisation du PM du niveau EML. Nous nous sommes donc concentrés à modéliser à haut niveau l'architecture de la plate-forme matérielle en s'abstrayant des éléments superflus pour le niveau EML : le type des composants, leurs caractéristiques physiques et technologiques, et la modélisation bit-accurate des protocoles supportés par les bus. Les protocoles sont modélisés à ce niveau par des événements du type REQUEST/RESPONSE avec ou sans paramètres comme nous le retrouvons dans le niveau PV-T du SystemC.

La Figure 53 illustre la modélisation de l'architecture de la plate-forme matérielle à haut niveau, architecture dédiée à exécuter l'ensemble du système et pas uniquement le décodeur MIMO. L'utilisation d'un diagramme de composants d'UML, associé aux éléments du paquetage GRM du profil MARTE, est tout particulièrement bien adaptée pour représenter cette topologie. Les composants offrant des ressources de calcul sont stéréotypés «ComputingResource», les composants offrant des ressources de stockages sont stéréotypés «StorageResource» et les composants de communication sont stéréotypés «CommunicationMedia» lorsqu'ils sont partagés par plusieurs ressources de calcul et/ou de stockages (maitres/esclaves).

Le composant *computer* représente une abstraction du composant programmable FPGA. Le stéréotype «ComputingResource» permet de préciser qu'il s'agit d'un composant offrant des services de calcul. La Figure 53 fait également apparaître l'architecture interne du FPGA. Ce dernier est composé des unités de calculs, certaines communiquant en point-à-point, d'autres au travers de bus stéréotypés «CommunciationMedia», de bus et d'unité de gestion d'accès mémoire stéréotypé «Storage Resource», mémoire externe au composant programmable stéréotypé également «Storage Resource». A ce niveau la distinction entre mémoire de stockage (SDRAM, DDR, ROM, etc) et un composant gérant l'accès mémoire (DMA) n'est pas faite.

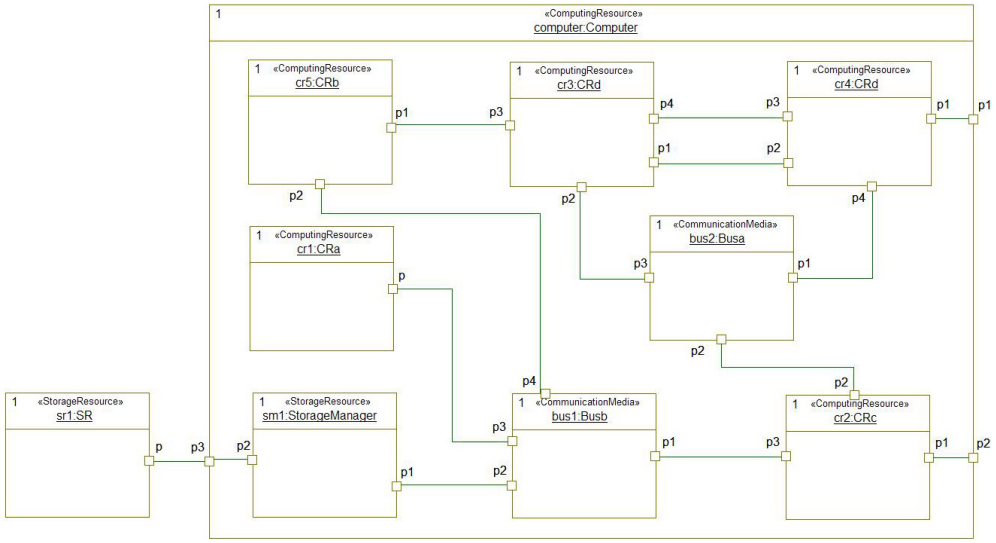


Figure 53. Diagramme de composants illustrant la topologie de la plate-forme matérielle à haut niveau, extrait du modèle PM du niveau EML.

3.3.2.3 Le modèle alloué : PSM

Le troisième modèle MDA du niveau EML, est le modèle alloué, modèle résultant de l'allocation des entités du PIM sur les composants du PM. La méthode d'allocation est similaire à celle mise en œuvre pour le niveau AML.

La Figure 54 illustre un extrait du PSM annoté avec des éléments du paquetage SAM en vue d'une analyse d'ordonnancement.

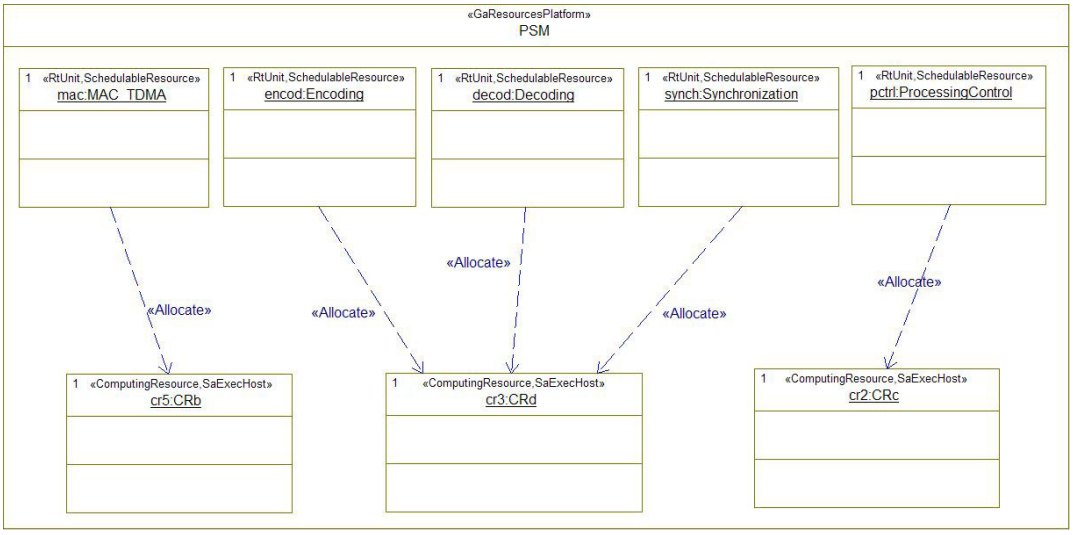


Figure 54. Diagramme de composant du modèle alloué annoté avec des stéréotypes de MARTE pour l'analyse d'ordonnancement.

D'un côté, la vue fait apparaître des entités applicatives (couche MAC/TDMA, encodage, contrôleur du système) du système de transmission sans-fil autre que le décodeur MIMO. Tous ces éléments sont non-seulement stéréotypés «RtUnit» mais aussi «SchedulableResource», stéréotype du paquetage SAM, du profil MARTE, qui indique qu'il s'agit d'élément ordonnançable et nécessitant d'être alloué sur un composant offrant un service d'exécution.

De l'autre côté, cette vue fait apparaitre 3 des ressources de calcul du PM du niveau EML. Celles-ci sont non-seulement stéréotypées «ComputingResource» mais aussi «SaExectHost», ce qui permet d'indiquer qu'il s'agit de ressource supportant l'exécution de tâches et son ordonnancement.

Note : il manque la définition de scénarios et de contextes associés pour avoir l'ensemble des éléments pour mettre en œuvre une analyse d'ordonnancement. Cependant, bien que nous ayons étudié les éléments de MARTE a utilisé, nous n'avons pas poussé plus loin leur utilisation dans le cadre du décodeur MIMO. Cet aspect est en dehors des travaux de cette thèse et est un thème de thèse en lui-même. Il s'agit donc d'une piste de travail pour le futur.

Comme pour le niveau AML, une simulation du PSM du niveau EML est nécessaire pour valider la topologie de la plate-forme matérielle. Pour cela, une transformation en un modèle équivalent SystemC (niveau PV-T) est proposée.

3.3.3 Le niveau DML

Une fois le niveau EML validé il s'agit de passer au dernier niveau de modélisation avant de générer du code RTL : le niveau DML. Nous rappelons que l'objectif principal du niveau de modélisation DML est de raffiner la plate-forme matérielle en spécifiant notamment le type des composants qui constituent la plate-forme matérielle et leurs caractéristiques associées (caractéristiques physiques notamment). A ce niveau, les protocoles sont modélisés au bit près. Cependant, il nous semble bien trop lourd de modéliser tous les protocoles existants au bit près. De ce fait, nous considérons que des modèles au bit près des différents protocoles existent sous forme de librairies. Nous rappelons que le modèle applicatif (PIM) du niveau DML est équivalent au celui du niveau EML : PSM du niveau AML (moyennant un refactoring si nécessaire).

Nous souhaitons mettre en avant l'utilisation des stéréotypes du paquetage HRM du profil MARTE, éléments utilisés pour spécifier les composants de la plate-forme matérielle (PM).

3.3.3.1 Le modèle de la plate-forme matérielle : PM

Dans notre cas d'utilisation, le PM du niveau DML est constitué de plusieurs vues modélisant l'architecture hiérarchique de la plate-forme matérielle du système de communication sans-fil. Le diagramme d'objets de la Figure 55 illustre la vue de top niveau hiérarchique plate-forme matérielle du système.

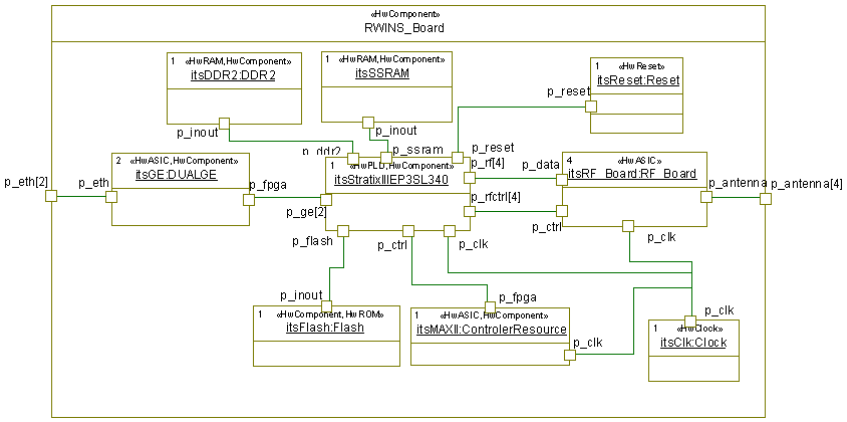


Figure 55. Diagramme d'objets représentant une vue de la plate-forme matérielle du système de communication sans-fil.

Le diagramme d'objets de la Figure 55 modélisant le top niveau de la plate-forme matérielle fait apparaître plusieurs types de composants :

- un composant programmable de type FPGA, tel que l'objet *itsStratixIIIEP3SL340,* stéréotypé «HwPLD», (raffinement du composant *computer*, stéréotypé «ComputingResource», du PM du niveau EML, cf. Figure 53) spécifie qu'il s'agit d'un composant programmable caractérisé entre autre par sa technologie, le nombre de LUTs, de bloc mémoire RAM, etc. De plus, cet objet est également stéréotypé «HwComponant», ce qui permet de spécifier ces caractéristiques physiques (dimension, consommation énergétique, etc) ;

- des mémoires externes au FPGA de différents types (RAM, stéréotype «HwRAM», ou ROM, stéréotype «HwROM»), caractérisées par leurs capacités de stockage, leurs technologies pour les mémoires ROM, etc ;
- des nœuds de calcul de type ASIC (circuit dédié à une fonctionnalité) stéréotypé «HwASIC», tel que l'objet *itsGe* qui modélise le composant qui gère la couche physique Ethernet ;
- une horloge physique globale modélisée par l'objet *itsClk* stéréotypé «HwClock» ;
- un composant de mise à zéro, modélisé par l'objet *itsReset*, stéréotypé «HwReset», issu du profil MOPCOM, qui permet de caractériser le reset du système (actif niveau haut/bas, synchrone/asynchrone). Ce composant doit impérativement apparaître sur le modèle PM du niveau DML pour la génération de code VHDL à partir de ce niveau. Si le modèle PM ne contient pas de composant stéréotypé «HwReset», la génération de code ne pourra être effective. Il s'agit d'une contrainte de modélisation du niveau DML non présentée dans ce mémoire.

La Figure 56 fait apparaître l'architecture interne du FPGA. Le diagramme de composition fait apparaître le type de chaque entité du FPGA. Nous retrouvons une architecture raffinée par rapport au PM du niveau EML (cf. Figure 53).

La vue interne du FPGA fait apparaître différents types de composants matériels :

- des bus stéréotypés «HwBus» (stéréotypés «CommunicationMedia» dans le PM du niveau EML), tel que l'objet *itsBAS*, modélisation de communication point à point unidirectionnelle entre nœud de calcul/ressource mémoire, l'objet *itBAM* (bus avec plusieurs maitres/esclaves entre nœud calcul/ressource mémoire) stéréotypé «HwBus» (objet *bus1*, stéréotypé «CommunicationMedia», du PM du niveau EML) ;
- des élément(s) stéréotypé(d) «HwProcessor» (stéréotypé(s) «ComputingResource» dans le PM du niveau EML), tel que l'objet *itsSystemCtrl* qui est un processeur logiciel de type NIOS II. Un grand nombre d'attributs du stéréotype «HwProcessor» permettent d'indiquer notamment le type d'architecture du processeur, le nombre de cœurs, le nombre d'ALU, la taille de la mémoire cache, etc ;
- des élément(s) de gestion de mémoire (DMA) stéréotypé(s) «HwDMA» (stéréotypé(s) «StorageResource» dans le PM du niveau EML) ;
- des élément(s) stéréotypé(s) «HwComputingResource» (stéréotypé(s) «ComputingResource» dans le PM du niveau EML), tel que l'objet *itsPHY_MIMO_IP*, nœud de calcul dans lequel le décodeur MIMO se trouve.

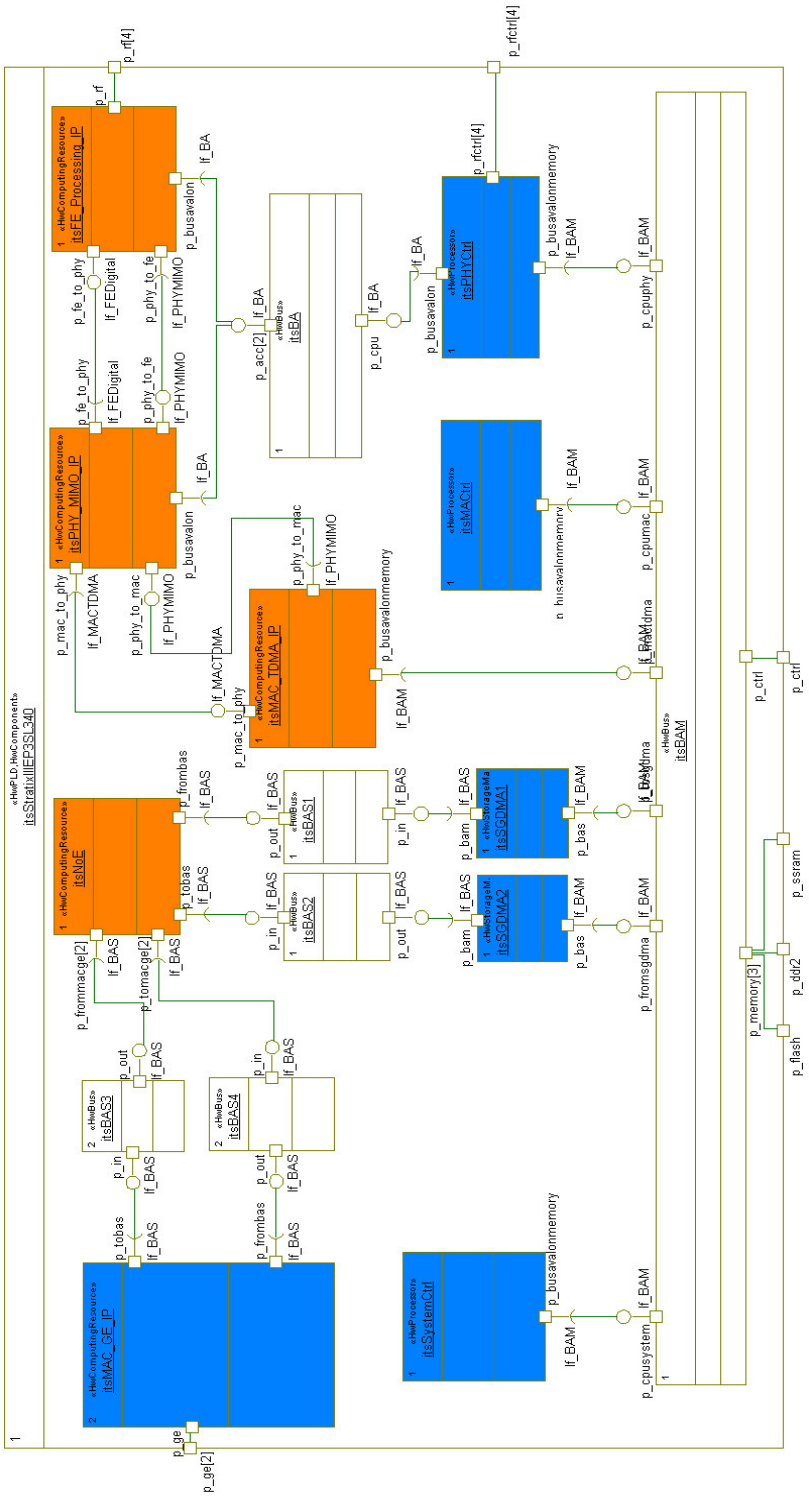


Figure 56. Diagramme de composition représentant une vue interne de l'architecture du FPGA avec les types des composants spécifiés.

3.3.3.2 Le modèle alloué : PSM

Enfin, il convient de réaliser l'allocation des entités du modèle applicatif (PIM) sur les composants de la plate-forme matérielle (PM). Le processus est identique à celui des niveaux supérieurs AML et EML.

La Figure 57 illustre un extrait de l'allocation d'entités du PIM sur des composants de la plate-forme matérielle PM.

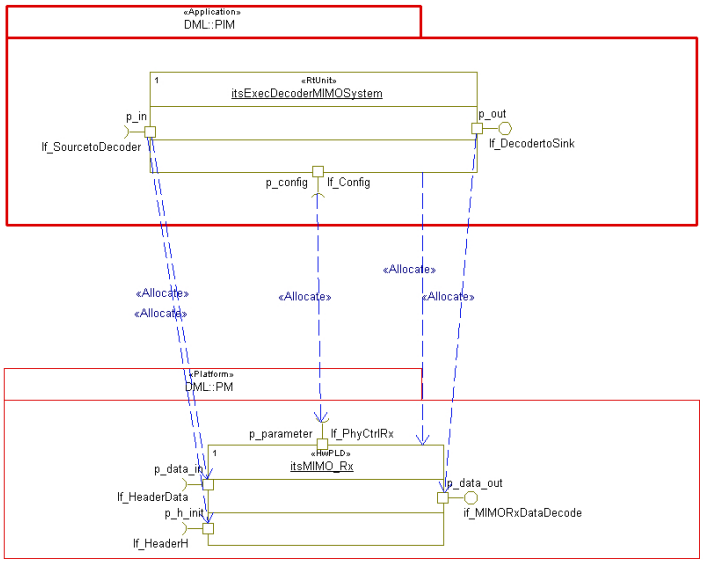


Figure 57. Vue illustrant l'allocation du décodeur MIMO sur le composant matériel supportant son exécution, extrait du PSM du niveau DML.

Le décodeur MIMO, modélisé au niveau du PSM par l'objet *itsExecDecoderSystemMIMO* (cf. Figure 50), est alloué sur le composant matériel modélisé par l'objet *itsMIMO_Rx*, ressource de calcul inclue dans le nœud de calcul de la couche PHY MIMO, représenté par l'objet *itsPHYMIMO_IP* (cf. Figure 56).

3.3.4 Analyses et évaluation de la méthodologie proposée

Nous présentons dans cette section notre analyse et évaluation sur l'aspect modélisation UML/MARTE de la méthodologie MOPCOM présentée et illustrée dans ce chapitre. Nous reviendrons sur les générations de code dans les chapitres 4 et 5. Dans un premier temps, nous présentons notre retour d'expérience sur l'utilisation de la méthodologie en elle-même au travers du système du décodeur MIMO, sous-ensemble d'un système de communication sans-fil de Technicolor. Notre analyse et notre point de vue exposé ci-après se place dans un contexte industriel. Dans un second temps, nous exprimons notre analyse de l'environnement d'outillage utilisé pour mettre en œuvre la méthodologie.

3.3.4.1 La méthodologie et la modélisation UML/MARTE

La modélisation du décodeur MIMO, modélisation réalisée en respectant la méthodologie MOPCOM, nous a permis de faire évoluer celle-ci par itérations successives et d'en proposer une première évaluation. En effet, les avantages, par rapport aux méthodologies de co-conception actuellement utilisées, d'une méthodologie dirigée par les modèles semblent nombreux sur le papier mais il est impératif de les prouver afin de faire accepter une telle approche auprès des ingénieurs de développement et tout particulièrement des ingénieurs hardware dont la modélisation n'est pas une des principales préoccupations, voire même totalement inconnue.

Aujourd'hui, la maturité la méthodologie MOPCOM et notre expérience de la modélisation du décodeur MIMO ne permettent pas de la comparer avec les méthodes actuelles. Cependant, nous pouvons donner notre analyse suite à notre expérimentation. Notre point de vue se place dans un contexte où nos travaux de recherche se sont effectués au sein d'une équipe de recherche du laboratoire Imaging & Network Experience de la division Research & Innovation de Technicolor et de l'équipe SCEE de Supélec, équipes dont les principales compétences sont le traitement du signal et le développement de système de type SoPC dans le domaine des télécommunications.

Le niveau AML de MOPCOM

Le niveau de modélisation se focalise principalement sur la modélisation de l'architecture fonctionnelle de l'applicatif et sur la sémantique de communication entre les entités fonctionnelles. Le point délicat de la mise en œuvre de ce niveau de modélisation n'est pas tant sur le choix de la découpe fonctionnelle mais plutôt sur la modélisation UML. Ce que nous voulons dire par là c'est qu'il n'existe pas une seule et unique modélisation possible du PIM d'un système au niveau AML. Ce modèle doit être le plus clair possible pour être compréhensible à la fois par les ingénieurs systèmes, les ingénieurs logiciels et enfin les ingénieurs hardware. Ceci passe par une utilisation intelligible des différents diagrammes UML, ce qui est d'autant plus vrai que le système à modéliser est complexe. L'autre point important de ce niveau est le choix de la sémantique de communication entre les entités du PIM. Comme nous l'avons présenté et illustré cela fait appel au domaine de compétence lié au MoC au niveau de la modélisation de la plate-forme d'exécution totalement virtuelle (PM) au niveau AML. Cette problématique nous semble très éloignée des problématique des équipes dans lesquelles ces travaux se sont déroulés.

Ainsi donc, nous avons constaté une difficulté notoire pour un ingénieur hardware à appréhender le niveau de modélisation AML de la méthodologie MOPCOM. En effet les préoccupations et objectifs de ce niveau sont très éloignés de la mise en œuvre matérielle. De toute évidence la modélisation du niveau AML n'est pas à développer par un ingénieur matériel.

Le niveau EML de MOPCOM

Le niveau de modélisation EML se focalise principalement sur la modélisation de la topologie de la plate-forme matérielle supportant l'applicatif. Dans notre expérimentation, la plate-forme matérielle du décodeur MIMO étant déjà existante nous n'avons pas eu à faire une véritable exploration d'architecture. Toutefois, cela nous a permis de manipuler des éléments du profil MARTE pour la modélisation de haut niveau du matériel. De plus, nous n'avons pas poussé suffisamment loin notre utilisation des paquetages GQAM, SAM et PAM utilisés pour annoter les modèles en vue d'analyse d'ordonnancement et de performances. Par conséquent, le niveau EML est clairement aujourd'hui le niveau qui manque le plus de maturité dans la méthodologie. Pour valider définitivement les choix fait à ce niveau (cf. section 3.2) il est impératif de le mettre en œuvre dans un contexte où la plate-forme matérielle est inexistante et nécessite impérativement une validation par le biais d'analyse d'ordonnancement et de performances. Nous pouvons toutefois prédire qu'une attention toute particulière doit-être menée dans le développement du niveau EML car il permettra certainement de détecter des erreurs très tôt dans le processus de développement : nous pensons notamment à des sous-dimensionnements, voire surdimensionnements, de la plate-forme matérielle.

Le niveau DML de MOPCOM

Le niveau de modélisation DML se focalise sur le raffinement du modèle de la plate-forme matérielle définie au niveau EML (PM). La principale difficulté de ce niveau est l'utilisation à bon escient des concepts du paquetage HRM du profil MARTE. Toutefois, celui-ci étant complet et proche des compétences matérielles, le niveau est certainement le plus facile à cerner et à développer pour un ingénieur hardware. En effet, l'utilisation du paquetage HRM permet notamment de spécifier la nature des composants (type de mémoire, type de processeur, etc.) qui constituent la plate-forme matérielle et leurs caractéristiques (fréquence de fonctionnement d'un processeur, nombre d'ALU d'un processeur, taille des bus d'adresses et de données pour les mémoires, nombre de portes logiques d'un FPGA, etc.).

Note : L'utilisation massive de librairie de composants nous semble nécessaire, notamment pour les bus et protocoles, afin de faciliter et accélérer le développement du PM du niveau DML et ainsi se focaliser sur d'autres préoccupations.

Généralités

D'une manière générale, nous constatons que la formalisation d'une méthodologie de conception de SoC/SoPC dirigée par les modèles permet de mieux appréhender ce type d'approche. En effet, la formalisation des différentes étapes de modélisation des 3 niveaux AML, EML et DML sous la forme d'un méta-modèle associé à des contraintes de modélisations (contraintes appliquées à l'utilisation d'UML et du profil MARTE) permet de mieux cerner les enjeux de chacun des niveaux de modélisation et de guider les équipes de conception dans la prise en main de la méthodologie en elle-même.

Ensuite, les compétences de l'équipe de Technicolor, au sein de laquelle j'ai effectué mes travaux de recherche, ayant mis au point le système de transmission sans-fil sont, outre de forte compétence en traitement du signal orienté télécommunication, le développement de systèmes de type SoPC. Dans ce contexte, l'utilisation d'une méthodologie de conception basée sur MDA, utilisant les concepts de modélisation de haut niveau et de transformation de modèle, est très éloignée des préoccupations des ingénieurs matériels. Par conséquent, l'utilisation de la méthodologie MOPCOM, ou toute autre méthodologie basée sur des modèles (cf. chapitre 2), doit passer nécessairement par une formation sur les concepts orientés objets et une prise en main du langage UML dans le cas de notre méthodologie. Toutefois, nous avons constaté que l'utilisation du profil MARTE dans la méthodologie permet aux ingénieurs hardwares de mieux appréhender la méthodologie, notamment la compréhension du modèle PM du niveau EML et DML. En effet, dans cette extension du langage UML, l'ingénieur peut se raccrocher aux éléments de modélisation du matériel du profil MARTE (processeur, mémoire, bus, contrôleur de mémoire, etc.). Le niveau AML étant le niveau plus dur à appréhender pour un ingénieur hardware, à l'opposé, le niveau DML se rapproche des préoccupations de ces ingénieurs.

De plus, dans la mise en œuvre de notre application nous avons noté un point critique dans l'approche MDA : l'allocation du modèle PIM sur un modèle PM pour obtenir un modèle PSM. En effet, aujourd'hui l'allocation est manuelle et ne garantit pas l'intégrité des différents concepts mis en jeu (par exemple, garantir la sémantique des MoC lorsque l'on passe du niveau AML au niveau EML). C'est actuellement aux équipes de développement utilisant la méthodologie MOPCOM de garantir cette intégrité, source d'erreurs potentielles importantes.

Pour conclure notre analyse, nous souhaitons mettre en évidence que les différentes activités de modélisation de la méthodologie MOPCOM seront affectées selon les compétences des ingénieurs. Les ingénieurs matériel se concentreront plus particulièrement sur les modèles PM du niveau EML et DML alors que les ingénieurs système et logiciel se concentreront sur les modèles PIM.

3.3.4.2 Les outils associés à la méthodologie

Notre analyse se concentre sur l'outil de modélisation UML Rational Rhapsody [67]. En effet, nous n'avons pas directement eu à manipuler les outils MDWorkbench [107] et Kermeta [43], respectivement l'outil de transformations de modèles et l'outil utilisé pour mettre en œuvre les contraintes de la méthodologie MOPCOM.

Tout d'abord, l'outil de modélisation est le cœur des outils associés à la méthodologie MOPCOM. Cependant, au vue des compétences des équipes dans lesquelles ces travaux se sont déroulés, l'utilisation d'outil de modélisation UML est loin d'être familière, voire fut totalement nouvelle. Ainsi, il était donc primordial que l'outil de modélisation UML soit mature, fiable et accessible (ergonomie).

Dès les premières manipulations de Rational Rhapsody, nous avons constaté que l'outil possédait ces caractéristiques. Par contre, un inconvénient majeur est la mise œuvre quelque peu propriétaire de la norme UML 2.1 dans l'outil. En effet, l'éditeur s'est octroyé le droit de faire certaines interprétations dans la mise en œuvre de la norme.

Par exemple, il n'y a pas de différence faite entre diagramme de classes et diagramme d'objets. Toutefois, l'adaptation est rapide. Cependant, cette caractéristique rend hypothétique toute compatibilité avec un autre modeleur UML, comme nous le verrons par la suite. De plus, cette mise en œuvre propriétaire de la norme UML ne permet pas d'utiliser tous les éléments du profil MARTE. En effet, certains stéréotypes sont simplement inapplicables car les concepts UML sur lesquels ils sont censés s'appliquer ne sont pas présents dans le framework de Rational Rhapsody. Il est donc nécessaire d'être au fait de ces informations lors de la capture des différents modèles du processus MOPCOM avec cet outil, car ce n'est pas sans conséquence. Par exemple, dans la méthodologie MOPCOM la notion d'allocation est très importante quelle que soit le niveau de modélisation. En effet c'est lors de cette phase que sont faits les choix d'implantation (partitionnement logiciel/matériel), Cependant, du fait de cette mise en œuvre propriétaire d'UML, l'outil présente un certain nombre d'inconvénients pour exprimer les allocations :

- Un seul niveau de hiérarchie des modèles est réellement géré ;
- La notion d'association n'a pas été mise en œuvre telle quelle, celle-ci est traduite en pointeur ;
- L'allocation lie des « classifiers » alors qu'elle devrait relier les propriétés de l'application à ceux de plate-forme.

Du fait qu'il propose une mise en œuvre de la norme UML 2.1 propriétaire, l'exportation des modèles vers d'autres outils de modélisation UML est très délicate, voir impossible. En effet, l'exportation de modèle capturé avec Rational Rhapsody, au format XMI (format d'échange standardisé des modèles UML), vers un autre outil de modélisation UML est très délicate. Le fichier XMI du modèle est le plus souvent mal interprété par un autre outil de modélisation (le test a été effectué avec deux outils de modélisation Open Source : Papyrus[12] [71] et BOUML [111], implémentant la norme UML 2.1 sans aucunes interprétations). Par conséquent une partie du modèle est tout simplement inexploitable voire perdue lors du passage d'un outil vers un autre. Ce problème rencontré ne va pas dans le sens prôné par le MDA : interopérabilité, portabilité des modèles développés avec des outils différents. Nous avons donc constaté que le choix d'un outil de modélisation, Rational Rhapsody dans notre cas, n'est pas anodin. Notre méthodologie est fortement liée à cet outil, par conséquent la portabilité des modèles développés et de la méthodologie en elle-même ne sera pas directe.

Dans le cadre de MOPCOM, un effort tout particulier a été réalisé pour intégrer la méthodologie MOPCOM et les générateurs de code (VHDL, SystemC et C/C++ dans l'outil Rational Rhapsody). En effet, une configuration par défaut des différents générateurs de code, associés à la méthodologie MOPCOM, est proposée à l'utilisateur comme le montre la Figure 58. Ainsi l'utilisateur n'a pas besoin de basculer vers l'outil de transformation génération de code MDWorkbench, outil de transformation de modèles intégré à Rational Rhapsody sous le nom RulesComposer). De plus, la phase de vérification des modèles développés est automatique lorsque l'utilisateur appelle un générateur de code (vérification des contraintes de modélisation faite par l'outil ModelChecker).

[12] Papyrus : outil de modélisation UML, intégrant le profile MARTE, développé par le CEA-LIST.

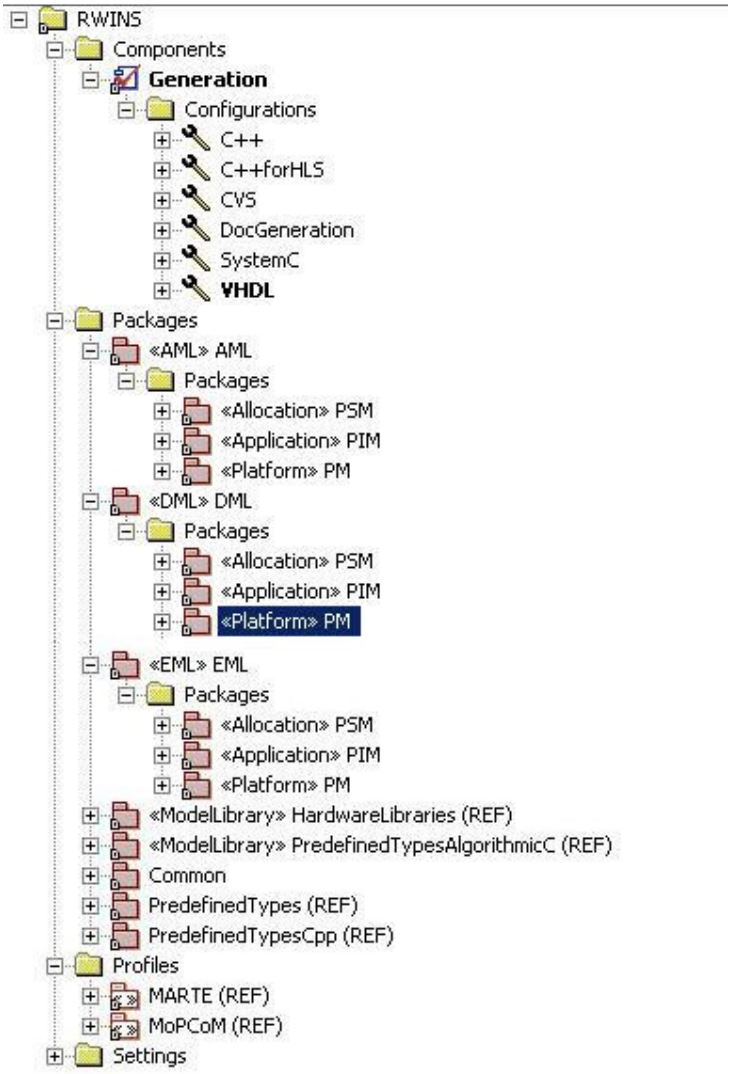


Figure 58. Illustration de l'organisation d'un projet MOPCOM dans l'outil de modélisation UML.

De plus, lors de la création d'un nouveau projet de type MOPCOM, le profil MARTE, le profil MOPCOM, et une organisation du projet respectant les 3 niveaux de modélisation MOPCOM sont automatiquement chargés.

Pour conclure, cette intégration de la méthodologie facilite grandement la prise en main du processus. L'utilisateur peut alors se concentrer sur la modélisation des différents modèles. Toutefois, cette intégration n'est pas indépendante de l'outil Rational Rapsody. Cependant l'outil de transformation de modèles (MDWorkbench basé sur le framework Eclipse), l'outil de vérification des contraintes de modélisation (ModelChecker basé sur Kermeta, lui-même basé sur le framework Eclipse également), le profil MARTE et MOPCOM sont eux indépendants de l'outil. Il est donc facilement envisageable de les mettre en œuvres avec un autre outil de modélisation UML, exercice qui n'a pas été mené mais qui serait nécessaire pour valider définitivement l'environnement d'outillage associé à la méthodologie MOPCOM.

3.4 Conclusion

Tout d'abord, dans ce chapitre, nous venons de présenter la méthodologie MOPCOM basée sur l'approche MDA et sur le profil MARTE pour tenter d'améliorer sensiblement la productivité lors de la conception de systèmes de télécommunications, et plus largement de systèmes électroniques temps réel embarqués. Nous avons participé à la définition de l'ensemble de la méthodologie proposée mais notre principale contribution s'est effectuée sur le niveau EML, niveau de modélisation dédié à l'exploration de l'architecture de la plate-forme matérielle. Nous avons présenté les différentes étapes de modélisation UML associées au profil MARTE, chaque étape étant soumise à des contraintes de modélisation formalisées et vérifiées avant le passage à l'étape suivante. Un méta-modèle formalise l'ensemble des étapes de la méthodologie.

Une mise en œuvre de celle-ci au travers du développement d'un système de communication sans-fil a été menée afin de valider la définition de la méthodologie mais aussi afin d'évaluer ses apports, ses manques, ses avantages et ses inconvénients par apport aux méthodologies de co-conception habituellement utilisées. Les premiers retours d'expérience sont encourageants mais dans l'état actuel de la maturité de la méthodologie proposée il est difficile d'indiquer un gain en terme de productivité par rapport aux méthodologies de co-conception actuellement utilisées. En effet pour proposer une comparaison exhaustive, il nous semble impératif de mettre en œuvre la méthodologie sur un système encore non-existant et développer par une équipe n'ayant pas contribué à sa formalisation afin de mettre en exergue de manière totalement indépendant les plus et les manques de celle-ci.

Pour conclure nous avons principalement parlé de modélisation UML/MARTE dans ce chapitre. Mais la méthodologie MOPCOM propose également des générations de code à partir des modèles développés. L'une d'entre elle est la génération de code RTL. Nous présentons dans le chapitre suivant nos travaux menés sur ce point.

Chapitre 4 Génération de code/mise en œuvre matérielle

Sommaire

Dans un processus de co-conception descendant (dit top-down en anglais) de systèmes embarqués, l'étape de mise en œuvre du logiciel embarqué et du code matériel intervient après les étapes de spécifications et de modélisation fonctionnelle. Dans le projet MOPCOM et ces travaux de thèse, nous nous sommes focalisés sur la mise en œuvre matérielle, celle-ci présentant les défis les plus importants. Cette étape consiste à décrire la partie matérielle du système au niveau RTL (*Register Transfer Level*) avec des langages de description matériel tels que le VHDL ou le Verilog. Aujourd'hui cette étape est manuelle. L'un des intérêts des méthodologies de co-conception basées sur les modèles est de proposer une génération de code matérielle à partir des modèles UML. Nous présentons tout d'abord succinctement la génération de code matériel directement depuis le modèle UML telle qu'elle est mise œuvre dans la méthodologie MOPCOM. Afin de compléter cette génération, nous proposons dans le cadre de cette thèse de coupler la méthodologie MOPCOM avec le concept de synthèse comportementale (ou synthèse de haut niveau). Nos nous sommes appuyés sur le décodeur MIMO, présenté dans le chapitre 1 dont la modélisation avec MOPCOM est présentée dans le chapitre précédent, pour illustrer ce couplage.

4.1 Génération de code UML vers HDL

4.1.1 Code matériel ou langage de description matériel : définition

Au début de l'air du numérique, la conception de circuits numériques passait par la description du comportement et de l'architecture du système sous forme de portes logiques. Les circuits devenant de plus en plus complexes, la description de tels systèmes sous forme de portes logiques devenait plus en plus complexe et coûteuse. Depuis un peu plus d'une quinzaine d'années, les langages de description matériel (HDL – *Hardware Design Language*) tel que le VHDL (ou le Verilog) ont permis d'accroitre la productivité en élevant le niveau de représentation des circuits numériques. Cela signifie que les équipes de conception ont accepté de perdre un peu en performance (surface, consommation, fréquence) par rapport à la saisie de schéma manuelle. Toutefois, on estime que le gain de productivité (programmation, simulation, tests) compense largement ces inconvénients. L'apparition des langages HDL a permis également de simuler le comportement des circuits et de générer automatiquement (synthèse logique) le schéma logique résultant. Comme nous l'avons déjà expliqué dans ce document, la complexité des systèmes embarqués d'aujourd'hui nécessite à nouveau d'élever le niveau de description, le développement du code HDL devenant de plus en plus coûteux dans la phase de conception d'un système. De ce fait, la synthèse comportementale (ou synthèse de haut niveau – HLS) est apparue, permettant de générer du code HDL depuis une description C, C++ voire SystemC. Partant de ce fait, nous proposons dans nos travaux de coupler une telle approche avec la méthodologie de co-conception MOPCOM basée sur les techniques MDA.

4.1.2 Mise en œuvre de la génération de code dans MOPCOM

A partir du niveau de modélisation DML du processus MOPCOM, une génération de code UML vers un code VHDL a été mise en œuvre. Cette génération est une transformation dite de modèle-à-modèle (M2M). Le jeu de règles pour la transformation a été défini à l'aide de la plate-forme MDWorkbench [107], développée par la société Sodius. Ce jeu de règles se base, d'un côté sur le méta-modèle du langage UML et de l'autre côté sur la sémantique du langage VHDL comme illustré par la Figure 59. La transformation est principalement basée sur le modèle UML. Les différents diagrammes UML ne sont pas exploités, exceptés les diagrammes d'états (Statecharts diagram dans l'outil de modélisation Rational Rhapsody d'IBM), et les diagrammes de classe/objet (Object Model Diagram dans Rational Rhapsody) pouvant contenir des informations impactant la transformation.

Ce générateur de code permet, à partir du modèle UML, de générer l'entité des composants matériels (ports d'entrée/sortie), les connexions entre les différents composants et toute la partie de contrôle d'un composant (issue d'une description sous forme de diagramme d'état). Par contre, la description des algorithmes ne donnera pas lieu à une génération de code matériel par le biais de ce générateur, mais sera considéré dans la suite de ce chapitre.

Il est à noter qu'à ce stade le code généré n'est pas synthétisable car seul le squelette de l'architecture est généré.

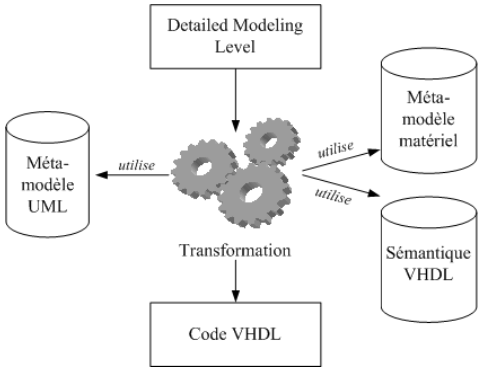


Figure 59. Synoptique de la génération de code UML vers VHDL.

Nous ne donnerons pas dans ce mémoire une illustration de cette génération de code. D'une part, la génération de code UML vers VHDL ne fut pas directement le cœur de des travaux de cette thèse, et d'autre part, le code VDHL généré en tant que tel n'apporte que peu d'informations intéressantes dans le cadre de cette thèse. Nous avons toutefois apporté notre contribution lors de la phase de validation et d'évolution du générateur de code UML vers VHDL par le biais de nos retours d'expérimentations auprès de la société Sodius qui a développé le transformateur. Nos retours d'expériences se sont effectués sur la génération des machines d'état et la génération des entités du décodeur MIMO présenté dans la section 1.2.2.3 du chapitre 1.

4.2 Synthèse de haut niveau

Comme nous l'avons déjà précisé dans le chapitre précédent, le générateur de code UML vers VHDL, développé dans le cadre du projet MOPCOM, ne permet pas de générer l'ensemble du code matériel à partir du niveau DML de MOPCOM. En effet, l'objectif premier du projet MOPCOM ne fut pas de développer un générateur de code UML vers VHDL complet. De plus, le développement d'un générateur de code matériel performant (générant du code matériel synthétisable) aurait nécessité une attention toute particulière dans un autre projet portant uniquement sur ce thème. Afin de fournir une méthodologie de co-conception complète, nous proposons dans cette thèse de coupler à la méthodologie MOPCOM des outils dédiés à la génération de code matériel synthétisable pour générer le corps des composants matériels. Ainsi, le flot de conception devient complet depuis les spécifications de haut niveau jusqu'à la génération de code exécutable sur la plate-forme réelle. Le choix s'est porté sur les techniques de synthèse comportementale, plus communément appelée synthèse de haut niveau (HLS – *High Level Synthesis*).

4.2.1 Les origines de la synthèse de haut niveau

La synthèse de haut niveau [112] est une transformation M2M. Le point d'entrée de cette transformation est un modèle comportemental d'une fonction (exécuté de manière séquentielle) et le point de sortie est un modèle décrit à l'aide d'un langage de description du matériel, tel que le VHDL ou le Verilog (modèle qui peut-être parallélisé). Cette approche n'est pas récente. Les premiers travaux sur la synthèse comportementale sont apparus, au début des années 80, sous la forme de compilateurs capables de transformer un code logiciel en un code matériel (Silicon Compiler de David L. JOHANNSEN [113] mais ce fut un échec. Quelques années plus tard, Synopsys, acteur majeur dans le

domaine des fournisseurs d'outils EDA, met sur le marché en 1994 son outil Behavioral Compiler [114] (ayant pour point d'entrée du VHDL/Verilog et pour point de sortie un code RTL synthétisé) : celui-ci n'est que très peu utilisé. Malgré ces échecs, un regain d'intérêt apparaît au début des années 2000 pour ce type d'approche. Après la société anglaise Celoxica, ce sont les succès rencontrés par de société Forte Design Systems et surtout Mentors Graphics qui vont positionner durablement ces solutions sur le marché de l'EDA. Aujourd'hui, une dizaine d'outils occupent ce marché tel que Cynthesizer [115], Pico Extrem [116] et Catapult C Synthesis [117][118][119] de Mentor Graphics, ce dernier capte à lui seul plus de 60% du marché.

La montée en puissance de ces technologies s'inscrit dans une logique d'élévation du niveau d'abstraction de la conception des circuits électroniques. Les outils HLS peuvent donc être vus comme une des composantes de l'ESL. Mais, malgré l'intérêt qu'engendre de tels outils au niveau de la productivité, le marché des outils de l'ESL reste encore très marginal (environ 200M$/an selon l'EDAC – *Electronic Design Automation Consortium* – [120]) par rapport au chiffre d'affaire de tous les outils de la CAO électronique (entre 4 à 5 milliard de dollars selon l'EDAC). De plus, les freins sont encore trop nombreux, notamment pour générer un SoC complet par le biais d'un tel outil. Cela reste donc encore un défi de recherche.

Prenant en compte ces constats, il nous est paru intéressant de coupler à la méthodologie de conception MOPCOM un outil HLS dédié à la génération de code d'IP matériel. Effectivement, la complémentarité de ces deux solutions est évidente, et ce choix nous permet de bénéficier à la fois des avantages de la modélisation UML et d'un outil de génération de code dédié : il nous est apparu évident qu'un tel outil pouvait être un bon complément au générateur UML vers VHDL de MOPCOM ainsi qu'aux librairies d'IP matérielles (quand elles existent).

La première réflexion menée avec les partenaires du projet MOPCOM fut le choix de l'outil HLS et son intégration dans le flot MOPCOM. Le choix de l'outil s'est très vite porté sur Catapult C Synthesis [117][118][119] de Mentor Graphics du fait de sa maturité.

4.2.2 Couplage d'un outil HLS avec la méthodologie MOPCOM

Notre démarche générale se voulant générique et évolutive, nous souhaitions proposer un mécanisme d'intégration opérationnel quel que soit l'outil HLS utilisé. Le point commun de la plupart de ces outils HLS est le langage supporté comme point d'entrée : ANSI C, C++, voire SystemC. Par conséquent, afin de connecter des outils de générations de code HDL dédiés, tels que les outils HLS, à la méthodologie MOPCOM, un générateur de code UML vers C++ a été développé dans le cadre du projet MOPCOM SoC/SoPC par la société Sodius. La Figure 60 illustre l'intégration d'un outil HLS dans l'environnement MOPCOM. Cette intégration est réalisée au niveau de modélisation le plus proche de la mise en œuvre réelle : niveau de modélisation intitulé DML du processus MOPCOM (cf. section 3.1.2.3 du chapitre 3).

Le générateur de code permet de transformer un modèle UML en un code C/C++ équivalent. Le générateur de code UML2CPP s'appuie sur un jeu de règles qui définit les règles de transformation d'un modèle UML vers un modèle C++. Ce jeu de règles s'appuie sur le méta-modèle UML d'une part et sur la sémantique C++ d'autre part.

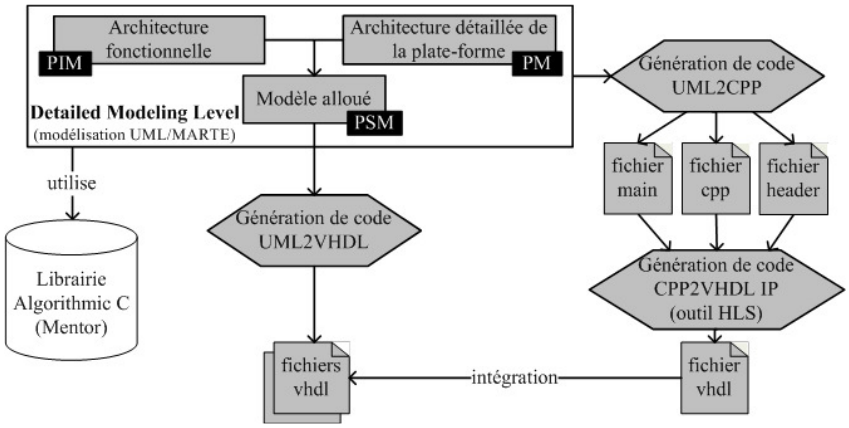


Figure 60. Intégration d'un outil HLS dans le flot MOPCOM.

De plus, intégré dans l'environnement d'outillage associé à MOPCOM, le générateur UML2CPP est adapté pour s'appuyer sur le langage d'action textuel de l'outil de modélisation Rational Rhapsody d'IBM [67] : celui-ci étant un sous ensemble du langage ANSI C. Ce langage d'action permet d'exprimer, entre autre, des actions, des opérations, des triggers, des conditions ainsi que déclarer des données.

Note : Cette extension n'est bien évidemment utilisable uniquement qu'à partir d'un modèle UML développé avec Rational Rhapsody. Cependant, sans cette extension, le générateur UML2CPP peut-être couplé à un autre outil de modélisation UML, car il est basé sur des standards.

4.2.2.1 Mise en œuvre d'un nouveau générateur : quel intérêt ?

Pourquoi avoir développé un générateur UML vers C++ alors qu'un certain nombre d'outil de modélisation UML intègre nativement déjà un générateur de code C/C++ ? L'intérêt d'un nouveau générateur est motivé principalement par deux points importants.

D'un côté, le code C++ généré n'est pas toujours exploitable directement. En effet, par exemple, le code C++ obtenu avec le générateur intégré de l'outil Rational Rhapsody est surchargé avec des éléments liés à son noyau OXF (*Object Execution Framework,* ensemble des éléments et règles qui régit le fonctionnement des services proposés par l'outil).

D'un autre côté, l'utilisation d'outil HLS ne permet pas d'avoir une totale liberté sur l'utilisation des éléments du langage C++. En effet, ces outils imposent des restrictions sur la manière dont est décrit le modèle d'entrée C++. Un grand nombre de ces contraintes sont identiques d'un outil à un autre telles que :

- Interdiction d'utiliser l'allocation dynamique de mémoire pourtant bien utile en programmation objet ;
- Utilisation restrictive des pointeurs (il est impératif de connaître la taille mémoire de l'objet sur lequel on pointe).

Or, si les générateurs de code intégrés aux outils de modélisation UML ne prennent pas en compte toutes ces caractéristiques la génération de code n'est plus fonctionnelle, d'où l'intérêt de proposer un générateur UML vers C++ indépendant de l'outil de modélisation par souci de généricité et pouvant être paramétré selon l'utilisation faite des modèles de sortie.

Note : Rappelons qu'une approche de co-conception de haut niveau a notamment pour but de maintenir la conception le plus longtemps possible indépendante de la cible matérielle afin de réduire les difficultés liées à la mise en œuvre pour passer d'une plate-forme à une autre.

4.2.2.2 Comment utiliser le générateur de code UML2CPP ?

Avant de pouvoir générer du code C++ avec le générateur UML2CPP présenté précédemment, les équipes de conception doivent :

- Ajouter des d'informations sur les modèles UML, du niveau DML de MOPCOM, afin de sélectionner les éléments du modèle destinés à la génération de code ;
- Paramétrer le générateur de code UML2CPP.

Ajout d'informations dans les modèles du DML

L'intérêt de coupler une méthodologie de co-conception MDA avec un outil HLS dédié à la génération de code matériel n'est pas de générer tout le code matériel par le biais d'un tel outil mais de se concentrer sur certaines fonctionnalités du système embarqué, dont la version HDL synthétisable n'est pas disponible.

Par conséquent la génération UML vers C++ ne s'effectuera pas sur l'ensemble du modèle mais uniquement sur un certain nombre d'éléments de celui-ci. Pour cela un mécanisme permet d'identifier les éléments du modèle sur lesquels sera appliquée la génération de code. En pratique, l'équipe ou l'ingénieur en charge de la mise en œuvre matérielle dispose d'un stéréotype «HLS», proposé dans le profile MOPCOM associé à la méthodologie du même nom. Ce stéréotype est applicable sur les méta-types *Class* et *Package* du méta-modèle UML. Une illustration de l'utilisation de ce stéréotype est donnée en Figure 61.

Aucune restriction n'est imposée sur la nature des classes stéréotypées. Par conséquent, l'utilisateur doit faire attention à ne pas stéréotyper automatiquement toutes les classes du modèle fonctionnel : par exemple la génération de code C++ pour des outils HLS depuis une classe abstraite, de par sa signification sémantique (pas d'implémentation des méthodes déclarées), n'a aucun sens. De même, les classes/objets du modèle fonctionnel chargé(e)s de modéliser des machines d'état ne seront pas traités par le générateur de code UML2CPP car celui-ci n'a pas été développé dans cet objectif. Il n'est donc pas nécessaire de les stéréotyper «HLS». Ce cas est lié non pas à la sémantique de l'UML mais aux caractéristiques ainsi qu'aux performances intrinsèques des outils HLS qui ne sont pas particulièrement performants avec la gestion du contrôle. De plus, la génération de code matériel (VHDL) depuis un diagramme d'état associé à la méthodologie MOPCOM étant performante, nous ne voyons pas d'intérêt à faire appel à d'autres outils pour générer le code matériel de cette partie du modèle UML.

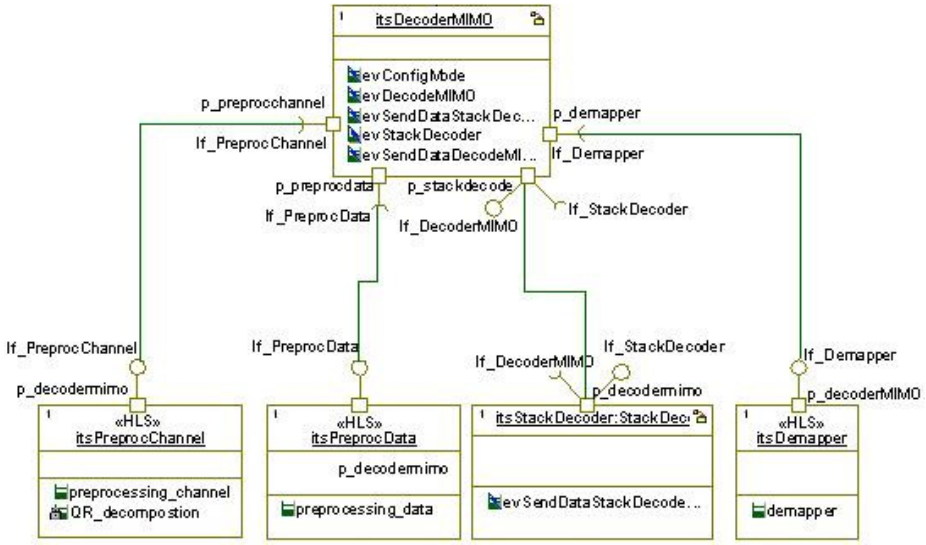


Figure 61. Modèle PIM du niveau DML du décodeur MIMO dont certaines classes sont stéréotypées «HLS», afin de générer le code C++ équivalent.

Paramétrage du générateur de code UML2CPP

Une fois fait le choix des classes du modèle, sur lesquelles sera appliquée la génération de code C++, l'étape suivante consiste à paramétrer le générateur de code C++ (UML2CPP) suivant l'utilisation qui sera faite des fichiers générés ; plusieurs versions sont disponibles à l'heure actuelle :

- une version « nue » : aucune contrainte n'est requise sur l'écriture du code C++ ;
- une version pour l'outil GAUT [121] ;
- une version pour l'outil Catapult C Synthesis.

La Figure 62 illustre le paramétrage du générateur de code UML2CPP pour répondre aux contraintes de l'outil HLS Catapult C de Mentor Graphic.

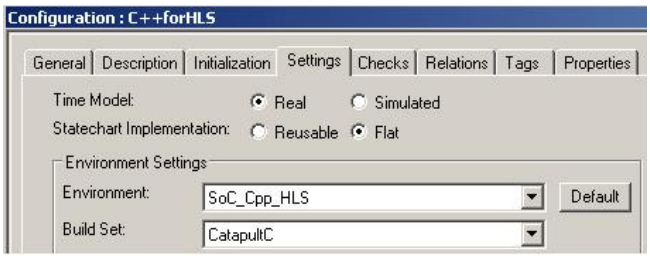


Figure 62. Fenêtre de configuration du générateur UML2CPP.

Le générateur en version nue est par défaut non restrictif et donc totalement générique. Le mécanisme de paramétrage permet d'adapter ce dernier à telle ou telle utilisation du code généré. De plus, le générateur ne se limite pas aux trois cas cités, il peut être personnalisé (évolution) afin de proposer une génération de code pour d'autre outil HLS. Il faut pour cela adapter le jeu de règles à l'outil utilisé.

Génération du code

Avant la phase effective de génération de code, le générateur vérifie la conformité des classes concernées par la génération de code avec le langage C++ et les contraintes imposées par l'outil HLS choisi (ou pas le cas échéant). Une fois cette étape effectuée, les fichiers d'entête programme (extension .h) et de définition des méthodes (extension .cpp) sont générés pour chacune des classes stéréotypées «HLS». Les attributs et le corps des méthodes de chaque classe seront générés. Par contre, tous les éléments issus des diagrammes d'états ne donneront lieu à aucune génération équivalente en C++ (rôle du générateur UML vers VHDL). De plus, un troisième fichier (ayant pour extension .cpp) est généré pour chaque classe. Celui-ci contient une fonction *main*, dont le corps contient une instance de la classe en question, ainsi que l'appel de la méthode principale de cette classe. De plus, toutes les données échangées par le biais des ports de la classe en question sont passés en paramètres de la fonction *main*. L'utilité de ce fichier se comprend aisément : une classe non instanciée (objet) ne permet pas de fixer les paramètres. Or un outil de synthèse comportementale doit connaître toutes les valeurs de paramètres afin de générer du code. En effet, selon la valeur des paramètres, le résultat de la génération de code HDL peut être différent (ce qui revient à dire : deux instances d'une même classe peuvent fournir une réalisation différente). En effet, les outils HLS ne peuvent pas mettre en œuvre une génération de code HDL directement à partir d'une classe. Il est nécessaire de créer un objet de cette classe pour initialiser le contexte et ainsi connaître la taille mémoire, entre autre, nécessaire à cet objet.

De plus, dans le cas de l'outil HLS Catapult C Synthesis uniquement, un pragma spécifique est ajouté afin d'indiquer qu'il s'agit de la fonction de plus haut niveau (important dans le cas d'une architecture hiérarchique). Un modèle de cette fonction main est donné en Figure 63.

```
#include « Classe.h »

#pragma hls_design top // pour CatapultC uniquement

Main_nomclasse(paramètres/données d'entrées/sorties)
{
      // Instance statique d'une classe
      nom_classe objet;

      // Appel de la méthode principale de la classe
      objet.nom_méthode_principale(paramètres/données d'entrées/sorties);
}
```

Figure 63. Modèle de la fonction main générée pour chaque classe stéréotypée «HLS».

4.2.2.3 Spécificités de la génération pour l'outil Catapult C Synthesis

Comme indiqué précédemment, chaque outil HLS impose des contraintes de codage sur les modèles d'entrées. Pour cela le générateur est spécialisé pour un outil. Dans le cadre du projet MOPCOM et de nos travaux nous rappelons que nous nous sommes concentrés sur le couplage de l'outil HLS Catapult C Synthesis de Mentor Graphics. Nous avons choisi cet outil pour sa maturité mais également car il intègre une chaîne de test automatisée, ce qui facilite grandement la phase de validation du composant matériel généré.

Les restrictions de codage C++

Tout d'abord, le générateur de code UML2CPP est spécialisé de manière à prendre en compte les restrictions de modélisation imposées par l'outil Catapult C. Le Tableau 9 récapitule les principales restrictions imposées sur le code C++ [117][118][119].

Tableau 9. Restrictions sémantique sur le modèle C++ en entrée de l'outil HLS Catapult C.

Catégorie de règle	Règles	Restriction(1) Catapult C
Type	Utilisation de la librairie Algorithmic C ac_int pour spécifier la dimension des vecteurs de bits	Warning
	Pas de type réel, représenté au format virgule flottante (type C++ float et double),→ utilisation du format virgule fixe avec la librairie Algorithmic C ac_fixed	Erreur
	Utilisation du qualificatif static uniquement pour la déclaration de variable constante	Erreur
	Utilisation de types signé (signed) et non-signé (unsigned) dans la même expression sans utilisation d'un «cast»	Erreur
Tableau	Pas de tableau avec une taille de mémoire indéfinie après compilation (utilisation de la déclaration directe au profit de la notation pointeur)	Erreur
	Utilisation d'allocation dynamique proscrite (instructions malloc, realloc, free, etc.)	Erreur
	Indexation d'un tableau avec une expression signée	Warning
Boucle	Boucle avec un nombre d'itérations indéfinie	Erreur
	Variable de sortie de la boucle doit être initialisée et de valeur d'incrémentation constante	Erreur
	Multiplication de boucles imbriquées	Erreur
	Utilisation de plusieurs instructions de sortie de boucle « *exit* » dans une même boucle	Warning
	Utilisation des instructions « *continue* » et « *goto* »	Erreur
	Boucle sans label d'identification	Warning
Sélection conditionnelle	Utilisation d'une instruction de condition « *if* » non-suivie d'une instruction « *else* »	Erreur
	Utilisation d'une instruction « *break* » inconditionnelle pour chaque branche d'un « *switch/case* »	Erreur
Fonction	Utilisation de multiples instructions return dans une même fonction	Erreur
	Récursivité de la fonction (à éviter)	Warning
Expression mathématique	Utilisation des opérateurs division / et modulo % (couteux en ressources)	Warning
	Utilisation des parenthèses pour améliorer le parallélisme de l'opération	Warning
	Utilisation des fonctions mathématiques telles que cosinus, sinus, racine carré, etc. (utilisation de la librairie Algorithmic C math)	Warning
Divers	Utilisation de flux d'entrée/sortie ou de fichiers (utilisation de la librairie stdio.h)	Erreur
	Utilisation de plusieurs horloges pour une même fonction (multiple domaine d'horloge)	Erreur

(1) Deux niveaux de restriction sont proposés :

- une restriction dure (Erreur) qui avorte la génération de code ;
- une restriction souple (Warning) qui ne bloque pas la génération de code.

Que ce soit l'une ou l'autre des restrictions détectées, une information sur la nature de celle-ci est indiquée à l'utilisateur. Dans le premier cas, une modification du modèle est impérative, par contre dans le second cas une modification du modèle n'est pas imposée mais recommandée afin de faciliter le travail de génération de code HDL par l'outil HLS.

A noter que nous n'avons pas recensé dans le Tableau 9 la notion de pointeur. En effet, l'utilisation de ce mécanisme dans un modèle C/C++ d'entrée de Catapult C, et de tout autre outil HLS, est à prendre avec précaution et tout particulièrement dans le cadre de passage de données à la fonction (ou une méthode) principale de la classe. En effet, dans un certain nombre de cas la taille mémoire nécessaire à la variable sur laquelle le pointeur pointe ne peut être connue de manière statique à la compilation du code. Dans le cas d'une mise en œuvre matérielle il est impératif de connaître la taille/dimension mémoire nécessaire pour chaque variable tout au long de la durée de vie du système. De ce fait, les pointeurs peuvent être utilisés dans un modèle d'entrée HLS si et seulement si la taille mémoire nécessaire à la variable pointée est connue et statique une fois la phase de compilation effectuée. Cette remarque est évidente pour un ingénieur matériel mais un ingénieur logiciel ne se poserait même pas la question.

De plus nous pouvons également indiquer des précisions sur le passage de données à une fonction C (ou une méthode C++), qui se traduira par des ports matériels dans le cas où cette fonction est désignée comme la fonction de plus haut niveau hiérarchique. Le passage de données peut se réaliser de deux manières : par variable ou par pointeur. Néanmoins ces méthodes donnent lieu à deux résultats différents après synthèse, ce qu'il convient de prendre en compte dans le cadre du fonctionnement global du système. En effet, l'un ou l'autre des résultats obtenus peut ne pas convenir au fonctionnement du système global.

Au niveau des entrées :

Le passage d'une donnée par variable engendre la création d'un registre, interne au composant, comme illustré sur la Figure 64.

Exemple : *void fonction(ac_int<12,1> entrée, ac_int<16,1> sortie)* ; se traduira par le schéma de la Figure 64.

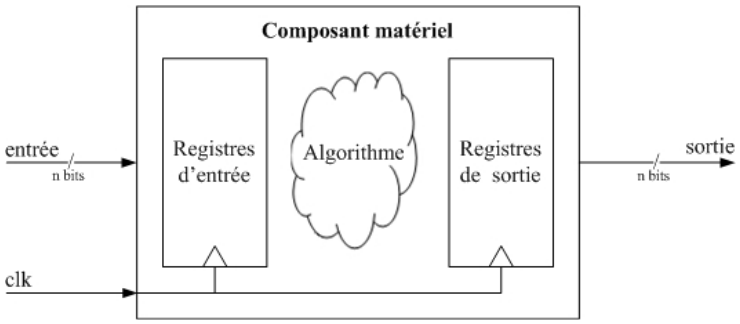


Figure 64. Illustration du résultat de la génération de code HDL dans le cas de données passées par variables.

Par contre, le passage d'une donnée par pointeur implique que l'information soit stockée d'une manière ou d'une autre à l'extérieur du composant (le plus souvent dans une mémoire de type RAM, une FIFO ou dans un registre) comme illustré sur la Figure 65. De plus, il est impératif que celle-ci soit stable à l'instant précis de sa prise en compte.

Exemple : *void fonction(ac_int<12,1> *entrée, ac_int<16,1> *sortie)* ; se traduira par le schéma de la Figure 65.

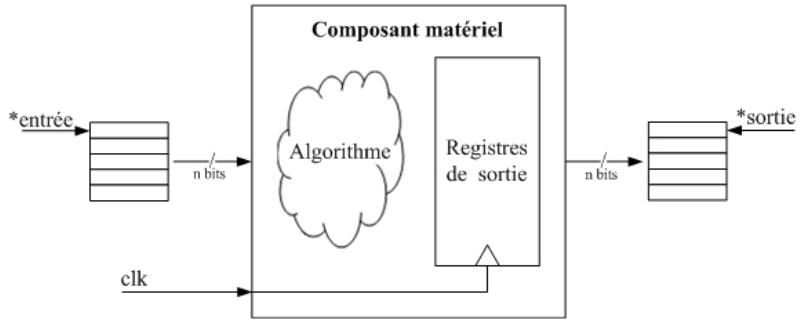


Figure 65. Illustration du résultat de la génération de code HDL dans le cas de donnée passé par pointeur.

Au niveau des sorties :

Dans les deux cas illustrés en Figure 64 et Figure 65, le passage par variable ou par pointeur engendre la création d'un registre de sortie. Il est à noter toutefois, dans le cas de l'utilisation d'un pointeur, qu'il faut faire attention à l'accessibilité de la variable externe au moment de l'écriture.

Pour conclure sur les pointeurs, il est fortement conseillé de limiter l'utilisation de ceux-ci dans le cadre d'une démarche de génération de code HDL avec un outil HLS pour faciliter au mieux le travail de ce dernier. En effet, l'outil connaît l'environnement extérieur à la fonction d'entrée uniquement par le biais de l'interface de celle-ci.

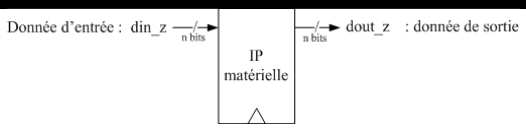
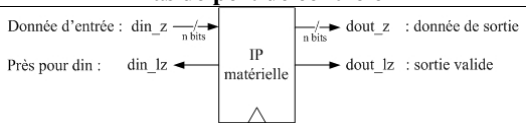
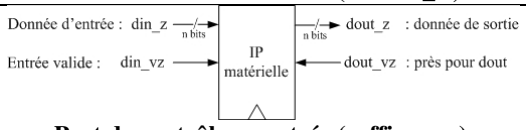
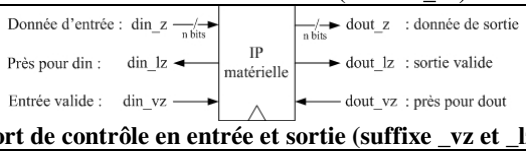
Caractéristiques des interfaces

Un des points importants dans la conception de SoC est la cohérence des interfaces des différents composants communiquant entre eux. Pour cela il est impératif que l'interface des composants générés par le biais de l'outil HLS soit compatible avec le code généré par le générateur UML vers VHDL afin de faciliter la phase d'intégration. Le cas échéant il sera alors indispensable de développer manuellement un « wrapper » pour faire une adaptation de l'interface avec celle de l'IP générée par l'outil HLS. Nous reviendrons plus en détails sur la phase d'intégration dans la section 4.3.1 de ce chapitre.

Le Tableau 10 répertorie les différents modes de contrôle d'un port de données et met en corrélation ceux-ci avec les modes de contrôle disponibles dans l'outil HLS Catapult C. De plus, la notation des différents ports reprend celle utilisée dans l'outil HLS afin de faciliter l'intégration des IP générées par Catapult C.

Note : le cas d'un port commun d'entrée/sortie (inout) n'est pas répertorié dans le tableau. Le cas existe mais il est fortement recommandé de ne pas le mettre en œuvre pour la synthèse HLS.

Tableau 10. Les différents modes de contrôle des ports de données disponibles dans l'outil HLS Catapult C.

Type de contrôle	Représentation schématique de l'interface d'une IP matérielle	Désignation dans Catapult C
Aucun contrôle	 **Pas de port de contrôle**	mgc_in_wire mgc_out_xxx
Contrôle interne au composant	 **Port de contrôle en sortie (suffixe _lz)**	mgc_in_wire_en mgc_out_xxx_en
Contrôle externe au composant	 **Port de contrôle en entrée (suffixe _vz)**	? ? (1)
Contrôle interne & externe au composant	 **Port de contrôle en entrée et sortie (suffixe _vz et _lz)**	mgc_in_wire_wait mgc_out_xxx_wait

(1) Ce cas n'est pas proposé dans l'outil HLS Catapult C.

De notre point de vue, la modélisation des ports de contrôle n'a pas un grand intérêt. Il nous semble peu pertinent de rentrer autant dans les détails. En effet, il faut éviter de surcharger les modèles UML pour ne pas les rendre illisibles. Cependant la méthodologie MOPCOM n'interdit pas au concepteur de les modéliser.

Toutefois pour que les ports de contrôle apparaissent sur les entités matérielles, une fois la génération de code UML vers VHDL effectuée, il faut offrir un moyen d'indiquer au générateur une information permettant de générer ces ports. Pour cela un stéréotype nommé «Handshake» applicable sur le méta-type *Interface* est disponible dans le profile MOPCOM. Associé à ce stéréotype, un tag nommé *control* permet de spécifier le type de mode de contrôle souhaité. A noter que le mode de contrôle choisi s'applique sur l'ensemble de l'interface (pouvant être constituée de plusieurs signaux distincts), ce qui peut-être contraignant. En effet, dans de nombreux cas le signal de données n'est pas associé à un (des) signal (aux) de contrôle. De ce fait, il pourrait donc être envisagé de faire évoluer le procédé en ce sens en appliquant le stéréotype «Handshake» non plus uniquement sur le méta-type *Interface* mais également sur les méta-types *Event* et *Attribut*. Cependant cette solution nécessite plus de travail pour l'utilisateur et peut engendrer une surcharge d'informations sur le modèle.

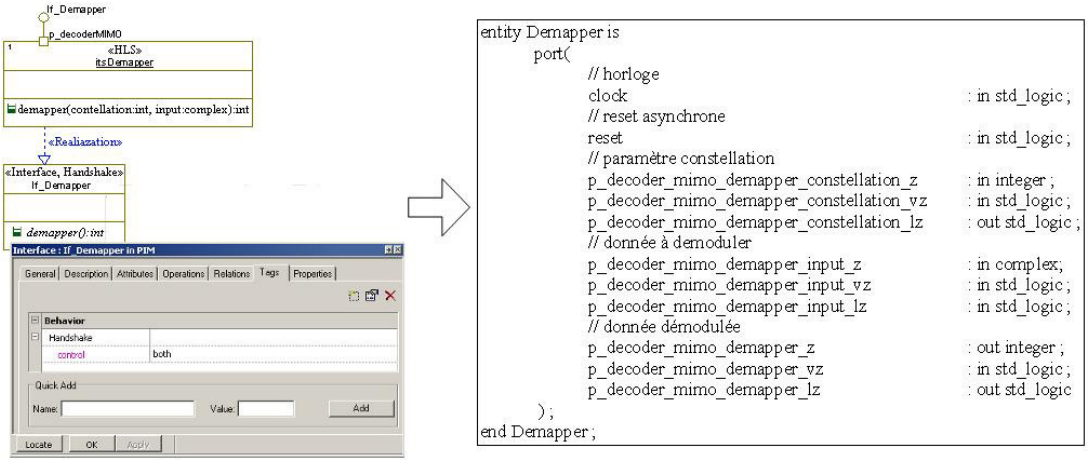


Figure 66. Exemple de sélection du type de mode de contrôle de l'interface du « demapper ».

La classe étant stéréotypée «HLS», seule l'entité de celle-ci sera générée par le biais du générateur UML vers VHDL. Le résultat de cette génération, tenant compte du mode de contrôle sélectionné, est donné en Figure 66 sachant que le type « complex » est déclaré comme suit :

```
type complex is
record
        i : integer ;
        j : integer ;
end record ;
```

Figure 67. Déclaration du type complex.

Nous montrerons par la suite que cette entité est cohérente avec l'interface de l'IP VHDL générée par l'outil HLS Catapult C.

Directive

Enfin, une des autres caractéristiques du générateur de code UML2CPP pour l'outil Catapult C concerne l'utilisation de pragma (directive pour « aider » la synthèse) intégré dans le code C++ et donc pouvant être également directement intégrée dans le modèle UML. La principale directive utilisée est celle qui désigne la fonction de plus haut niveau : « *hls_design* ». Cette directive n'est pas directement présente dans le modèle UML, elle est générée dans le fichier *main* comme suit (cf. Figure 63) : *#pragma hls_design top*.

Attention, toutes les directives proposées avec Catapult C [117] ne sont pas supportées par le générateur de code.

4.2.3 Mise en œuvre et résultats de synthèse de haut niveau

Afin de valider le couplage de la méthodologie MOPCOM avec un outil HLS (Catapult C dans notre cas) nous avons procédé en trois étapes :

- Validation du générateur de code UM2CPP ;
- Génération d'IP matérielle (VHDL) par le biais de l'outil HLS Catapult C de Mentor Graphic à partir du modèle C++ d'une fonction ;
- Intégration de l'IP VHDL générée dans le système complet (dans notre cas le décodeur MIMO) pour évaluer à la fois ses performances et sa facilité d'intégration.

Au-delà des performances des outils HLS, notre objectif principal est de démontrer l'intérêt de coupler une méthodologie de co-conception MDA, telle que MOPCOM, avec un outil HLS. Nous souhaitons également mettre en exergue à quel point l'utilisation d'un tel outil peut influer en amont sur la phase de modélisation. Enfin, malgré les contraintes d'utilisation des outils HLS, nous souhaitons mettre en évidence que l'utilisation d'un tel outil contribue à proposer une méthodologie de co-conception plus générique et indépendante de la plate-forme cible.

4.2.3.1 Génération de code UML vers C++

Comme nous l'avons déjà indiqué dans la section 4.2.2, le développement du générateur de code UML2CPP a été assuré par la société Sodius. Nous avons apporté notre contribution lors de la phase de validation et d'évolution par le biais de nos retours d'expérimentations. Ainsi pour valider celui-ci nous avons procédé en deux étapes :

- validation et comparaison du code généré, depuis le modèle UML, avec un code C/C++ écrit manuellement, développé sur la base d'un modèle Matlab du décodeur MIMO, présenté dans le chapitre 1;
- test du code C/C++ en entrée de l'outil HLS Catapult C, compilation de celui-ci dans l'outil.

Validation et comparaison du code C++ généré

Cette première étape peut paraître anodine mais il est impératif de valider la fonctionnalité du modèle C++ généré avant d'exploiter celui-ci dans un outil HLS. Comme dans tout processus de développement chaque étape est validée avant le passage à la suivante afin d'éviter la propagation d'erreurs tout au long du processus de développement. Cependant, pourquoi faut-il valider le code C++ généré, pourquoi ne pas valider le modèle UML ? Comme indiqué dans la chapitre 2, le langage UML ne peut être simulé nativement (même si certains outils de modélisation UML, tel que Rational Rhapsody d'IBM, intègrent un mécanisme de simulation des modèles), il s'agit à la base d'un langage de représentation. De plus, en considérant le modèle UML valide, rien ne prouve que la génération de code UML vers C/C++ soit dénuée d'éventuels bogues. Certes la génération de code est automatique mais le jeu de règles est développé par un ingénieur, être humain qui n'est pas infaillible. Il se peut donc que dans la phase de développement du générateur des erreurs subsistent tant que le générateur ne sera pas généralisé et utilisé à grande échelle pour valider tous les cas. Il est donc impératif de

valider le code généré. Pour cela, chacune des classes donnant lieu à une génération de code C++ a été validée par le biais d'un vecteur de test issu des résultats d'un modèle Matlab.

Lors de cette première étape nous avons relevé quelques imperfections au niveau de la génération de code de tableaux de valeurs.

Tout d'abord, l'initialisation des tableaux telle qu'elle est écrite suite à la génération de code UML vers C++ ne respecte pas la sémantique du langage. La Figure 68 illustre un extrait de la déclaration de la classe *Demapper* du décodeur MIMO issue du modèle UML.

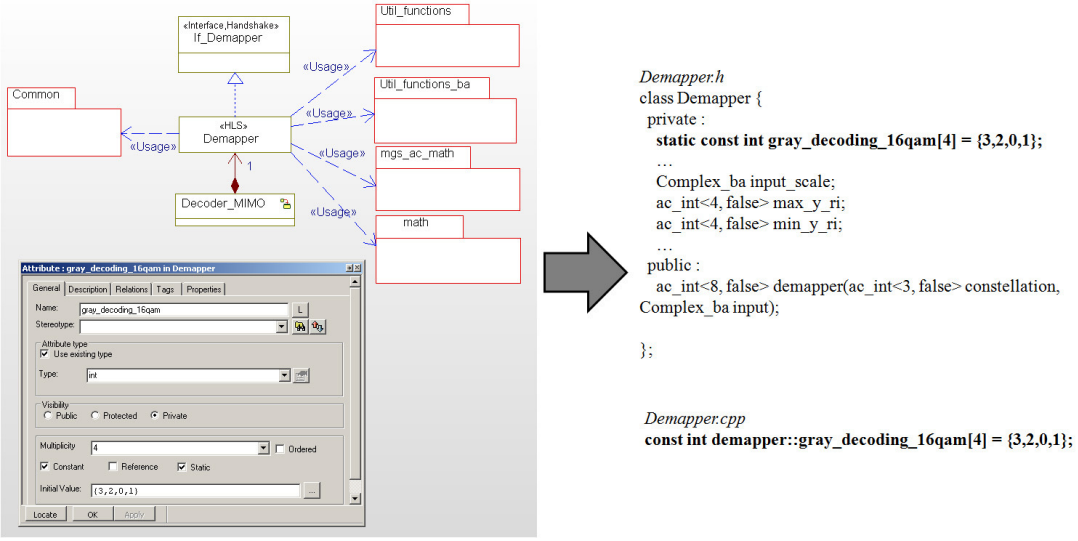


Figure 68. Exemple de génération de code de la classe « Demapper », stéréotypée «HLS», ayant entre autre un attribut de type tableau dont les valeurs initiales sont des constantes statiques.

A gauche de la Figure 68, la fenêtre illustre la manière dont est saisie la déclaration et l'initialisation de l'attribut *gray_decoding_16qam* de la classe *Demapper* dans le modèle UML. La partie droite de la Figure 68 illustre un extrait du résultat de la génération de code UML vers C++. Dans la sémantique C++, l'initialisation d'un tableau ne peut-être mise en œuvre dans la déclaration de la classe (ici décrite dans un fichier d'en-tête). Seule la déclaration du tableau doit être présente dans la déclaration de la classe, et l'initialisation de celui-ci, si il y a lieu d'être, doit être réalisée dans le ficher de définition dans lequel on retrouve les définitions des méthodes de la classe, comme illustré en bas à droite de la Figure 68.

Ensuite, l'écriture de la déclaration de tableau de dimension 2 et plus ne répond pas à la sémantique C++, ce problème est illustré par la Figure 69.

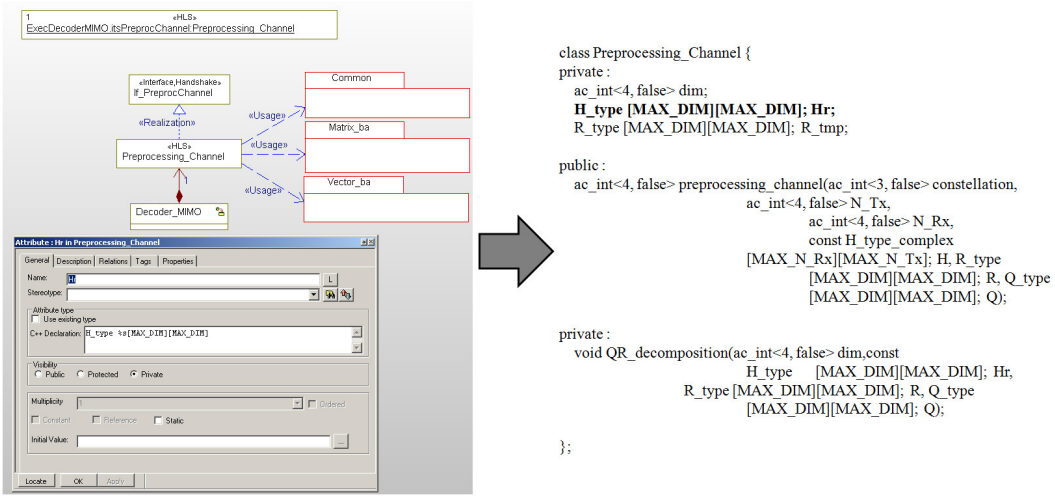


Figure 69. Exemple de génération de code de la classe « Preprocessing_Channel », stéréotypée HLS, du décodeur MIMO, ayant entre autre, un tableau à 2 dimensions comme attribut.

A gauche de la Figure 69, la fenêtre illustre la manière dont est saisie la déclaration de la matrice *Hr* (matrice de canal H constituée de valeurs réelles) de la classe *Preprocessing_Channel* dans le modèle UML. La partie droite de la Figure 69 illustre un extrait du résultat de la génération de code UML vers C++. Dans la sémantique C/C++, le nom du tableau doit figurer avant la dimension de celui-ci exprimée entre crochets.

Ces deux imperfections doivent être corrigées manuellement dans l'état actuel de développement du générateur UML2CPP. Ces erreurs proviennent en partie de la manière dont sont saisis les tableaux dans l'outil de modélisation UML et non pas dû à une mauvaise implémentation des règles permettant d'extraire l'information du modèle UML et de générer le modèle C++ équivalent d'une classe.

Une fois les modifications apportées, la fonctionnalité du code C++ est validée à l'aide d'un vecteur de test écrit en C/C++.

Test du code C/C++ en entrée de l'outil HLS

La seconde étape de vérification du code généré consiste à vérifier que toutes les restrictions de codage imposées par l'outil HLS ont bien été prises en compte lors de la génération de code. Cette étape s'effectue directement dans l'outil HLS. La démarche présentée ci-dessous a été réalisée avec l'outil de synthèse Catapult C. Malgré les différences entre les outils HLS, la démarche générale reste la même.

Une fois le projet créé, les fichiers d'entrées (.h et .cpp) correspondant au modèle d'entrée sont chargés. Le passage à l'étape suivante (choix de la cible technologique) passe par une phase de compilation des fichiers d'entrées. L'outil donne la main à l'utilisateur pour l'étape suivante uniquement si la compilation ne révèle pas d'erreurs. Le compilateur intégré prend en compte la sémantique ANSI C et les contraintes que l'outil impose.

Ces deux étapes de vérification de code sont nécessaires et impératives. Toutefois pour alléger le travail des équipes de conception, nous pourrions envisager une automatisation de ces deux étapes : lancer par le biais de scripts par exemple.

4.2.3.2 Synthèse de haut niveau

Une fois le code C++ validé, il convient de l'exploiter pour générer du code matériel. Dans le cadre de nos travaux, nous nous sommes concentrés sur la sous fonction de prétraitement et non sur l'ensemble du décodeur MIMO (cf. Figure 11) qui se compose :

- D'une mise en forme et d'un prétraitement de la matrice de canal H ;
- D'un prétraitement des données réceptionnées IQ (Rx).

La Figure 70 illustre la complexité des traitements effectués sur le vecteur Rx et sur la matrice de canal H.

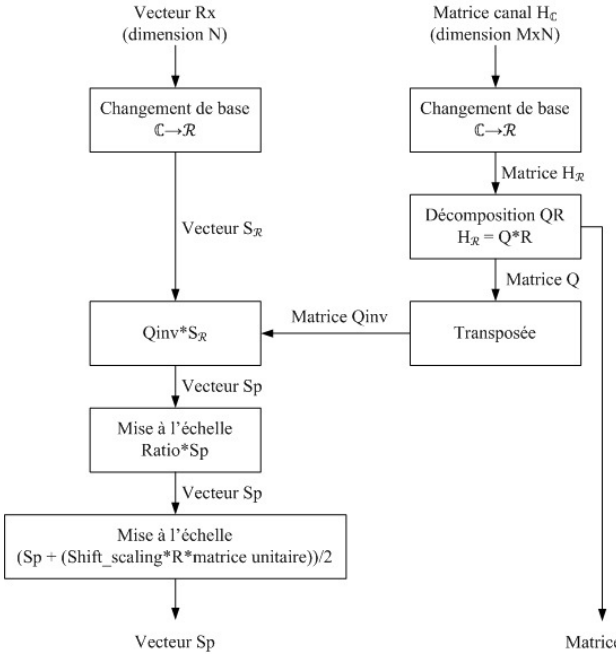


Figure 70. Décomposition fonctionnelle de la fonction de prétraitement du décodeur MIMO.

L'opération la plus complexe de cette fonction est la décomposition QR. Comme nous l'avons précisé dans la section 1.2.2.3 du chapitre 1 plusieurs algorithmes existent. Nous avons opté pour l'algorithme de Gram-Schmidt (cf. Annexe 1), car nous disposions d'une version VHDL optimisée manuellement, ce qui permettra de réaliser une comparaison avec la version générée par Catapult C, dont le code C++ est issu du modèle UML du décodeur MIMO.

Dans un but de comparaison, certains paramètres de l'outil HLS Catapult C sont identiques pour toutes les itérations de génération de code présentées par la suite :

- la cible technologique visée est un FPGA Stratix III de chez Altera (référence EP3SL340F1517C) ;
- la fréquence souhaitée du design est fixée à 100MHz ;
- le signal de remise à zéro est synchrone à l'horloge et actif au niveau bas ;
- l'outil HLS est paramétré afin de générer une IP VHDL optimisée en fréquence et non en surface.

Ces caractéristiques sont issues du décodeur MIMO développé à la main par le laboratoire Network de Technicolor. Nous pourrons ainsi comparer les résultats obtenus.

Dans un premier temps, nous avons modélisé la fonction de prétraitement en une seule classe, nommée *Preprocessing* sur la Figure 71, regroupant la mise en forme de la matrice canal H, le prétraitement de cette dernière et le prétraitement des données IQ à décoder.

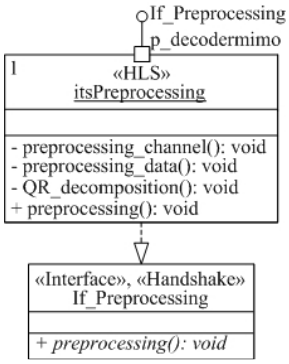


Figure 71. Modélisation UML de la fonction de prétraitement du décodeur MIMO.

La génération de code UML vers C++ fournit principalement trois fichiers comme expliqué dans la section 4.2.2. A partir de ces fichiers (modèle C++ de la fonction de prétraitement) nous avons procédé pas-à-pas pour observer l'impact des différentes contraintes d'architecture appliquées dans l'outil HLS sur les performances du composant matériel généré.

Cas 1 : génération de code HDL sans contrainte architecturale

Afin d'avoir un premier point de comparaison nous avons effectué une première génération de code sans imposer la moindre contrainte architecturale (telle que l'agencement mémoire, le déroulage de boucle, le parallélisme, le pipelining, etc.) à l'outil HLS Catapult C. Cependant, l'interface du composant matériel doit absolument être paramétrée, opération primordiale comme cela a été expliqué précédemment (cf. section 4.2.2). Le type de contrôle est choisi (Figure 72) en cohérence avec celui associé à l'interface du port de la classe *Preprocessing* (dans le modèle UML, cf. Figure 71) afin de faciliter l'intégration futur du composant matériel généré. Ce choix est également en cohérence avec le type de contrôle utilisé dans la version développée manuellement de cette même fonction.

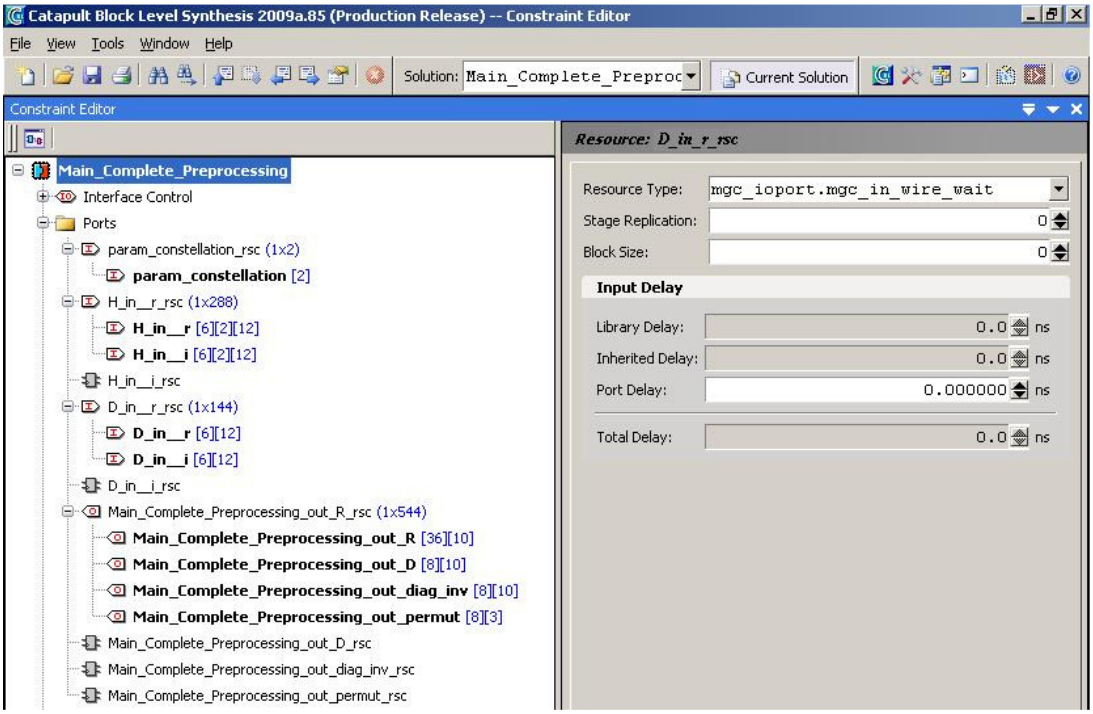


Figure 72. Fenêtre de l'outil HLS Catapult C illustrant l'étape de paramétrage de l'interface du composant « Preprocessing ».

La Figure 72 met en évidence le type de contrôle choisi pour le signal d'entrée *D_in_r_rsc* (Rx, données IQ réceptionnées à décoder) : un contrôle à la fois externe et interne (cf. Tableau 10. Les différents modes de contrôle des ports de données disponibles dans l'outil HLS Catapult C.) indiqué par le paramètre *mgc_ioport.mgc_in_wire_wait*, ce qui est cohérent avec le choix fait au niveau de la modélisation UML. Le paramétrage de l'interface du composant sera identique tout au long de notre étude.

N'appliquant aucune autre contrainte à l'architecture du composant, l'IP matérielle est générée en quelques minutes. Pour valider la fonctionnalité de l'IP générée, l'outil HLS Catapult C intègre un flot de vérification automatique. A partir d'un vecteur de test décrit en C/C++, l'outil met en place, de manière totalement transparente pour l'utilisateur, un environnement de test qui compare les données en sortie du modèle C avec celles en sortie de l'IP VHDL (qu'il a générée à partir de ce même modèle C). Ce flot de vérification intégré offre un gain de temps considérable et permet également d'éviter des erreurs de codage, d'autant plus que cela ne nécessite pas de développer un vecteur de test décrit en VHDL.

Certes avec ce flot de vérification la fonctionnalité de l'IP VHDL est assurée mais cela n'indique en aucun cas si les performances temporelles respectent les contraintes du cahier des charges et cela ne donne pas d'indication sur la surface occupée par le composant matériel généré. Pour cela l'outil HLS propose principalement deux valeurs, l'une pour évaluer les performances temporelles et l'autre pour évaluer la surface du composant matériel généré. L'outil offre également d'autres outils de statistiques récapitulant le nombre de boucles, le pourcentage de temps de calcul de chaque boucle par rapport au nombre de cycles complet de l'IP, le nombre de registres, la taille mémoire utilisée, etc.

Les résultats obtenus pour cette architecture sont les suivants : 2 474 cycles de latence et un score en surface de 43 641 (valeur relative donnée par l'outil pour comparer les différentes architectures produites). Au vue de ces valeurs il est clairement impossible de se prononcer sur les performances de l'IP. Il convient donc de faire une simulation de l'IP pour valider les performances temporelles et faire une synthèse de l'IP afin d'obtenir le nombre réel de portes logiques utilisées. Les résultats de simulation et de synthèse sont exposés dans la section 4.3.2. Nous verrons que les performances de cette première architecture sont très éloignées du cahier des charges.

Cas 2 : génération de code avec contraintes architecturales

Après une première génération de code sans contraintes architecturales (en dehors de l'interface du composant), nous proposons d'observer l'impact des contraintes architecturales imposées à l'outil sur les performances du composant matériel généré. Outre le choix de l'interface, l'utilisateur dispose de plusieurs clés pour orienter la génération de code HDL dans l'outil HLS Catapult C : il peut intervenir principalement au niveau des boucles et de l'agencement de la mémoire interne nécessaire au composant.

Au niveau des boucles, Catapult C offre la possibilité de les dérouler, ou de les pipeliner. De plus, l'outil permet également d'interdire le regroupement d'une boucle avec une autre, afin d'interdire la parallélisation des traitements. Par contre, le parallélisme ne peut directement être imposé par l'utilisateur mais sera détecté par l'outil et mis en œuvre dès que cela est possible (l'outil est capable de regrouper deux boucles si le pas d'incrémentation et le nombre d'itérations sont compatibles et ainsi paralléliser les calculs effectués dans ces deux boucles par exemple). Ce dernier point est important : cela impacte la manière dont le modèle d'entrée est écrit. En effet, il faut anticiper le travail effectué par l'outil et donc penser au parallélisme possible de la fonction lors de l'écriture des différentes boucles.

En reprenant la fonction de prétraitement, tout en utilisant la même interface que précédemment, l'outil HLS détecte 34 boucles (cf. Figure 73). Certaines boucles sont interdites de regroupement (parallélisme interdit), d'autres sont mises à plat (déroulées partiellement ou totalement) et d'autres sont pipelinées.

Il n'est pas question ici d'expliquer le choix pour chacune des boucles de la fonction mais plutôt d'observer l'impact que peut avoir ce choix sur l'architecture de l'IP générée et les performances de celle-ci. Ainsi, un diagramme de GANT[13] permet d'observer plus ou moins aisément l'impact des choix. Pour obtenir une solution optimale, une exploration d'architecture itérative est nécessaire. Toutefois, la facilité avec laquelle l'utilisateur peut modifier les contraintes sur les boucles et comparer le résultat avec les versions précédentes offre un confort indéniable.

Note : Le choix de dérouler ou de pipeliner une boucle ne doit pas se faire automatiquement mais de manière réfléchie. De ce fait, seul un ingénieur ayant une expérience en développement matériel sera capable de jouer au mieux avec ces contraintes.

[13] Diagramme de GANT : diagramme illustrant l'ordonnancement et la durée de chaque opération élémentaire

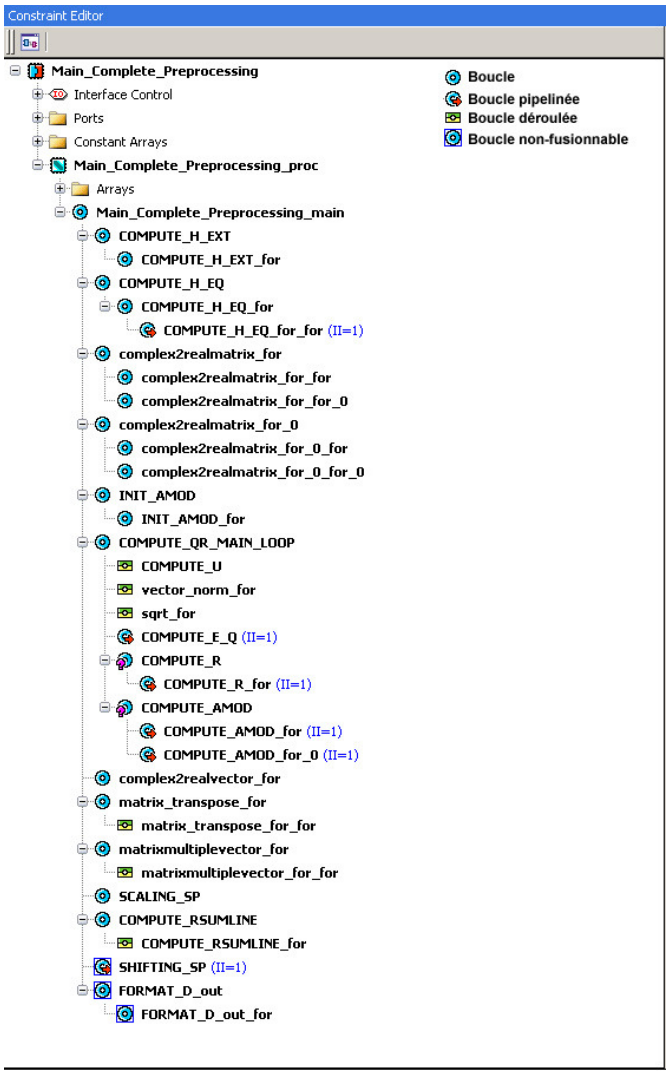


Figure 73. Illustration des contraintes appliquées sur chacune des boucles de la fonction de prétraitement.

Par exemple, la boucle nommée *COMPUTE_U* sur la Figure 73, qui calcule une matrice intermédiaire, est déroulée dans sa globalité. La boucle nommée *SCALING_SP* sur la Figure 73, qui applique un coefficient au vecteur SP, peut être fusionnée avec une autre boucle (décision prise par l'outil) mais ne sera ni déroulée ni pipelinée. La boucle nommée *SHIFTING_SP* sur la Figure 73 est à la fois pipelinée et ne peut être fusionnée avec une autre boucle.

Les résultats obtenus pour cette seconde architecture sont les suivants : 1 852 cycles de latence et un score en surface de 38 747 (valeur relative donnée par l'outil pour comparer les différentes architectures produites). Nous pouvons constater un gain de 25%, par rapport au premier cas, au niveau du nombre de cycles nécessaire à la fonction pour générer une donnée en sortie. Un gain de 12% est également obtenu sur la surface occupée par l'IP par rapport au premier cas.

Malgré ces résultats encourageant, seule une simulation temporelles et une synthèse de cette IP nous permettront de valider l'architecture choisie (cf. section 4.3.2).

Nous pouvons déjà annoncer que le nombre de cycles nécessaires est toujours trop important par rapport à l'objectif. Par conséquent, il faut jouer sur d'autres éléments que les boucles pour améliorer sensiblement une nouvelle fois les performances de l'IP.

Cas 3 : génération de code avec contraintes sur l'agencement de la mémoire interne

Outre l'interface du composant, et les boucles de traitement, l'utilisateur peut intervenir sur un dernier élément pour obtenir une IP matérielle performante. En effet, l'utilisateur peut choisir le type de la mémoire interne et son agencement nécessaire au fonctionnement du composant matériel (cf. Figure 74). L'utilisateur dispose notamment de mémoire de type ROM (issue de la librairie de la technologie cible choisie) pour stocker les attributs constants (tel que des paramètres, des tables de conversion), de mémoire RAM simple ou double port(s) également issue de la librairie de la technologie cible choisie, et de mémoire de type registre.

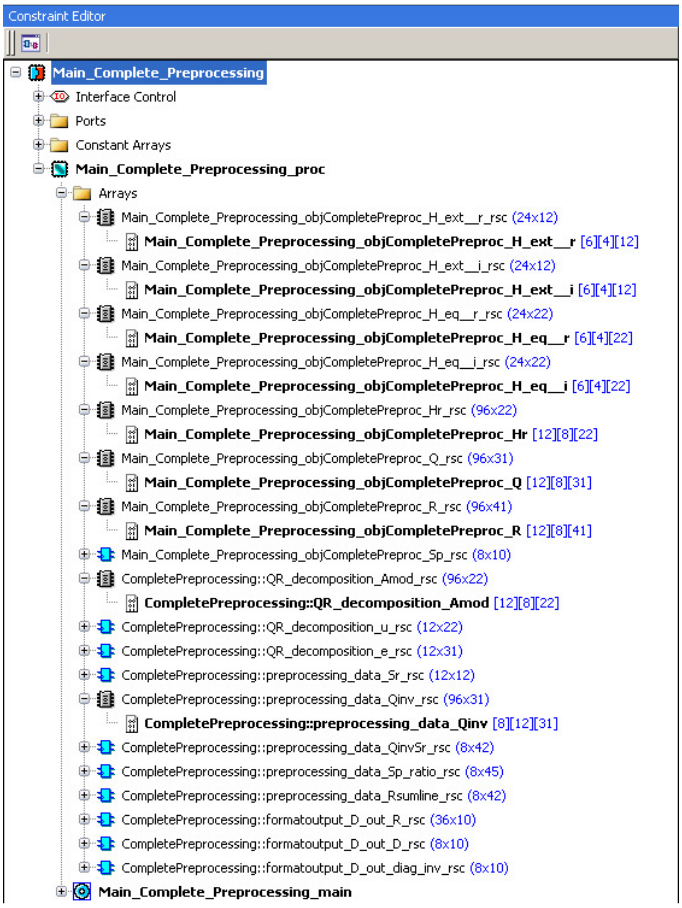


Figure 74. Illustration de la mémoire interne nécessaire à l'IP de prétraitement.

Par exemple, le vecteur *Sp* (cf. Figure 11 et Figure 70), désigné par le terme *Main_Complete_Preprocessing_objCompletePreproc_Sp_rsc* dans la Figure 74, est stocké sous la forme de 8 mots de 10 bits dans un registre. La matrice Q conposée de 12 par 8 éléments (cf. Figure 11 et Figure 70), désignée par le terme *Main_Complete_Preprocessing_objCompletePreproc_Q* dans la Figure 74, est quant à elle stockée dans un bloc mémoire de type RAM sous la forme de 96 mots de 31 bits.

Une fois le type de mémoire sélectionné, l'utilisateur peut agencer les différentes mémoires, en regroupant plusieurs attributs dans un même élément mémoire. Pour cela, il choisit la taille du mot (profondeur de la mémoire), ce qui fixe la dimension globale de l'élément mémoire. Il peut également choisir l'adresse de début pour chaque attribut stocké dans un même élément mémoire.

Lors des deux premiers cas présentés précédemment les attributs internes de la fonction de prétraitement étaient tous stockés, par défaut, dans des registres. Avec un tel choix, l'accès aux données est rapide mais la surface du composant matériel généré est d'autant plus importante notamment dû aux fils d'accès à chaque registre.

Afin de réduire la surface de l'IP, les attributs les plus consommateurs en mémoire, tels que les matrices, peuvent être stockés dans des blocs mémoires RAM (simple ou double port(s)).

Dans le cas de notre fonction de prétraitement, l'organisation de ces blocs mémoires RAM peut être ajustée pour optimiser l'accès aux données (par exemple, pour obtenir l'accès immédiat à toute une ligne ou colonne d'une matrice et non pas un accès valeur par valeur).

Note : l'agencement de la mémoire et donc l'accès aux données stockées dans cette mémoire peut avoir un impact sur les contraintes imposées sur les boucles.

L'impact au niveau des résultats obtenus est flagrant : tandis que la surface occupée par le composant est nettement plus faible, 14 302 comparé au 43 641 et 38 747 des cas précédents, soit un gain respectivement de 68% et de 63%, le nombre de cycle est quasiment doublé, passant de 1 852 à 3 478 cycles, soit une différence de 47%. Le gain en surface est net mais engendre une dégradation des performances temporelles du composant.

Par conséquent, malgré un compromis entre les contraintes sur les boucles et le type de mémoires internes choisies, le résultat obtenu ne garantit pas que les performances de l'IP générée respectent le cahier des charges.

Cas 4 : retouche du modèle d'entrée

Lors de la phase de modélisation, en dehors de l'ajout du stéréotype «HLS» et du stéréotype «Handshake» pour spécifier l'interface du composant, l'équipe en charge de développer le modèle fonctionnel ne tiendra pas nécessairement compte des contraintes d'utilisation d'un outil HLS pour générer du code matériel. Par conséquent le modèle UML et plus particulièrement la description des algorithmes (principalement décrits en C++) de chacune des méthodes des classes n'est pas nécessairement optimisée. En effet, à un haut niveau de modélisation (niveau AML de MOPCOM), les performances, telles que l'optimisation mémoire, le temps de calcul d'une tâche ne sont pas une priorité. Au contraire, dans une phase de conception matérielle ces éléments sont très importants pour optimiser les performances du composant matériel. Nous rappelons que l'un des objectifs du niveau EML, et plus encore le niveau DML de MOPCOM, est de proposer une architecture matérielle du système optimisée, optimisation obtenue par raffinement successif.

Partant de ce constat, le modèle UML du décodeur MIMO a été retouché, et plus particulièrement la classe *PreProcessing* modélisant la fonction de prétraitement. La structure de celle-ci n'a pas été modifiée mais plusieurs modifications ont pu être apportées au niveau de ses différentes méthodes. Nous ne décrirons pas ici en détails les modifications apportées au modèle. Toutefois nous pouvons indiquer comme modifications majeures :

- La suppression d'attributs locaux temporaires, ce qui réduit le nombre de blocs mémoire nécessaires et le temps de calcul ;
- Le fractionnement de la matrice canal H en sous-matrice, afin d'optimiser le parallélisme des calculs basés sur cette matrice, ainsi que l'accès mémoire ;
- L'utilisation d'opérateurs de la librairie Algorithmic C en place des opérateurs de base (par exemple l'opérateur « / » ou encore l'opérateur de calcul de la racine carré), ce qui permet d'optimiser ces calculs ;

Le résultat a été obtenu par un raffinement successif entre le modèle UML et le code HDL obtenu avec l'outil HLS.

Les contraintes appliquées sur l'interface, les boucles et l'agencement de la mémoire interne sont identiques à la version précédente (cas 3), sauf bien évidement pour les modifications apportées, ce qui entrainent l'apparition de boucles supplémentaires (une dizaine de plus par rapport au modèle C++ précédent).

Les résultats obtenus, pour ce quatrième cas, pour la génération de code après optimisation du modèle d'entrée C++ sont les suivants : 962 cycles de latence et un score en surface de 38 764. Le constat est immédiat, par rapport au second cas le nombre de cycles est divisé par 2 (1 852 → 962) alors que la surface est similaire. L'augmentation du nombre de boucle (fragmentation de certaines boucles de l'algorithme de la fonction de prétraitement en plusieurs boucles moins complexes) facilite le travail de l'outil HLS, notamment pour paralléliser et pipeliner ces dernières.

Ce quatrième cas de génération de code HDL, que ce soit par la synthèse comportementale, ou que ce soit depuis le modèle UML directement, montre clairement l'importance de prendre en compte dès la phase de modélisation l'utilisation d'un tel outil, et au delà d'un outil HLS, de prendre en compte la mise eu œuvre matérielle. De plus cela montre également que le niveau de modélisation le plus proche de l'implémentation finale de la méthodologie MOPCOM (DML) doit certainement être modélisé par une équipe d'ingénieurs matériels.

4.2.4 Conclusion

Pour évaluer l'impact des différentes contraintes architecturales applicables sur les performances de l'IP VHDL générée, une évaluation pas à pas à été réalisée. Le Tableau 11 récapitule les résultats obtenus pour la génération de code VHDL avec l'outil HLS Catapult C de la fonction de prétraitement du décodeur MIMO.

Tableau 11. Récapitulatif des résultats de synthèse comportementale pour la fonction de prétraitement du décodeur MIMO.

	Cas 1	Cas 2	Cas 3	Cas 4
Nombres de cycles	2 474	1 852	3 478	962
Surface	43 641	38 704	14 302	38 764

Le meilleur résultat est obtenu en trouvant le meilleur compromis architectural par un raffinement successif. Nous notons que, malgré le résultat encourageant obtenu dans le quatrième cas, celui-ci ne satisfait pas le cahier des charges. De fait, nous pouvons tirer les conclusions suivantes :

- Les performances de l'IP générée dépendent non seulement du code d'entrée mais aussi d'une utilisation optimale de l'outil HLS utilisé. Or, dans le cadre de cette thèse, nous ne prétendons pas avoir exploré l'ensemble des possibilités offertes par l'outil Catapult C ;
- La découpe fonctionnelle du modèle UML du décodeur MIMO n'est pas optimale. Il faudrait alors revoir l'architecture fonctionnelle et proposer une découpe plus fine. Ainsi donc le code C++ des fonctions sera plus simple à interpréter par l'outil HLS. Par exemple, nous pourrions revoir l'architecture de la fonction de prétraitement traitée avec l'outil HLS pour la scinder en plusieurs sous-fonctions plus simples.

4.3 Intégration et validation

L'outil HLS permet de valider la fonctionnalité de l'IP générée, de comparer les performances de différentes architectures générées mais ne permet pas de valider clairement les performances temporelles et surfacique de l'IP matérielle générée. De ce fait, une étape supplémentaire est nécessaire pour valider l'IP générée.

4.3.1 Intégration

Une fois l'étape de génération de code C++ depuis le modèle UML effectuée, puis le composant matériel (IP VHDL) généré par le biais de l'outil HLS, celui-ci doit-être intégré dans l'architecture du système complet. Pour cela, deux cas de figure sont envisageables :

- intégration de l'IP générée dans l'architecture globale du système généré depuis le modèle UML (flot de développement MDA) ;
- intégration de l'IP générée dans l'architecture globale du système développé dans le cadre d'une méthode de co-conception actuelle (i.e. développée principalement manuellement).

Note : L'effort à fournir pour intégrer cette IP dans l'un ou l'autre de ces deux cas n'est pas le même.

4.3.1.1 Intégration de l'IP dans une architecture générée depuis un modèle UML

Nous rappelons que le code matériel (VHDL) généré depuis le niveau de modélisation DML du processus MOPCOM est principalement constitué du squelette de chaque composant matériel (vue boite noire) et des interconnexions entre les composants. De plus, tout le contrôle du système issu d'une représentation sous forme de diagrammes d'état est également généré.

Dans ce cas, si l'équipe de conception, en charge du développement matériel, a bien suivi la procédure proposée dans la méthodologie MOPCOM, l'interface des différentes classes stéréotypées «HLS» est caractérisée à l'aide du stéréotype «Handshake», comme expliqué dans la section 4.2.2.3. De ce fait, l'entité VHDL, générée par le générateur UML vers VHDL la génération de code matériel VHDL est cohérente avec la déclaration de l'entité générée par l'outil HLS (comme illustrée par la Figure 75). En reprenant l'exemple de la fonction de démodulation (*Demapper*), nous pouvons constater que la déclaration de son entité VHDL, générée par l'outil HLS, est similaire, au nom près, de l'entité générée depuis le modèle UML (cf. Figure 66).

Le nom des ports de l'IP VHDL générée par l'outil HLS ne pouvant être changé (le code VHDL généré par l'outil HLS est assez abscond, de ce fait il est préférable de ne pas tenter de le modifier), le renommage doit être effectué au niveau du code VHDL généré directement depuis le modèle UML (avec le générateur UML vers VHDL).

Ce cas d'intégration est très certainement le moins coûteux à mettre en œuvre pour les équipes de conception et garantit une certaine fiabilité, mais cela suppose que la conception du système soit guidée par une approche MDA telle que nous le proposons. Aujourd'hui, les approches de conception par les modèles ne sont encore qu'à l'état de recherche. Par conséquent, les équipes de conception au sein des entreprises n'utilisent pas ce type d'approche.

```
ENTITY MainDemapper IS
  PORT(
        demapper_constellation_rsc_z : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
        demapper_constellation_rsc_vz : IN STD_LOGIC;
        demapper_constellation_rsc_lz : OUT STD_LOGIC;
        demapper_input_real_rsc_z : IN STD_LOGIC_VECTOR (19 DOWNTO 0);
        demapper_input_real_rsc_vz : IN STD_LOGIC;
        demapper_input_real_rsc_lz : OUT STD_LOGIC;
        demapper_rsc_z : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
        demapper_rsc_vz : IN STD_LOGIC;
        demapper_rsc_lz : OUT STD_LOGIC;
        clk : IN STD_LOGIC;
        rst_n : IN STD_LOGIC
        );
END MainDemapper;
```

Figure 75. Déclaration de l'entité « Demapper » générée par l'outil HLS Catapult C.

4.3.1.2 Intégration de l'IP dans une architecture écrite à la main

Depuis quelques années, un certains nombres d'entreprises éprouvent le besoin d'intégrer dans leur processus de développement de co-conception des outils HLS. Ces outils sont principalement aujourd'hui utilisés sur des sous-ensembles d'un système pour générer des IP matérielles en vue d'être intégrées dans l'architecture globale du système, comme peut l'être une IP issue de librairies de composants (technique de réutilisation de plus en plus utilisée pour améliorer la productivité et la fiabilité).

Partant de ce fait, l'intégration d'un composant matériel (IP VHDL) généré par un outil HLS dans une architecture matérielle écrite à la main demande un effort non-négligeable qui doit être pris en compte dans la démarche de conception en comparaison avec le cas précédent. Comme nous l'avons présenté avec Catapult C, l'interface de l'IP générée par l'outil HLS peut-être paramétrée (par exemple, dans Catapult C, le type de contrôle associé au signal, le formatage peuvent être paramétrés), mais ce paramétrage n'est pas aussi souple que dans le cas d'une description manuelle de cette interface.

L'interface de l'IP matérielle, générée par un outil HLS, n'a que peu de chance de correspondre à 100% aux interfaces des composants avec lesquelles celle-ci communique (comme illustré en haut de la Figure 76), sauf si le travail de spécification des différents composants et notamment de leurs interfaces est effectué en tenant compte des caractéristiques de l'outil HLS utilisé.

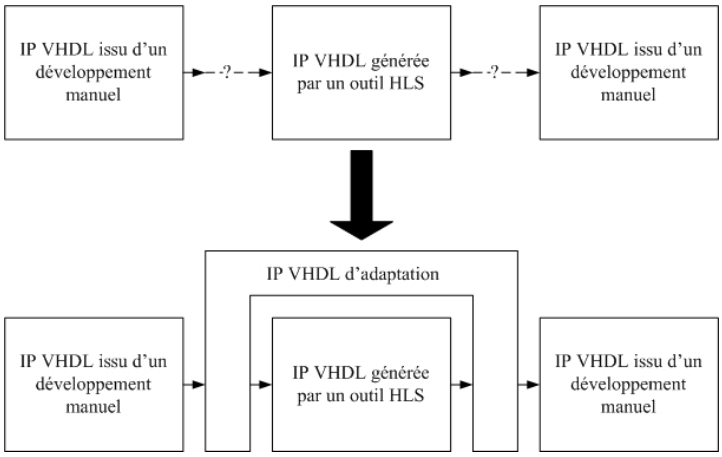


Figure 76. IP VHDL générée par un outil HLS encapsulée dans un composant d'adaptation d'interface.

Par conséquent, il est nécessaire d'adapter l'interface du composant avec ces composants communiquant. Cette adaptation est réalisée au travers d'un composant d'adaptation (wrapper en anglais). Ce composant encapsule le composant généré par l'outil HLS (comme illustré en bas de la Figure 76), et il est développé manuellement. La complexité de celui-ci est plus ou moins importante selon de degré d'adaptation nécessaire. L'adaptation des interfaces de chacun des composants communiquant entre eux peut nécessiter un reformatage des signaux (taille du signal identique mais un ordonnancement des bits différent), nécessiter une adaptation du mode de contrôle, voire également contenir des composants de temporisation (FIFO) pour adapter le débit des signaux. De plus, la complexité de ce composant d'adaptation le sera d'autant plus que l'IP communique en entrée et en sortie avec plusieurs composants.

Dans le cas de la fonction de prétraitement du décodeur MIMO, le choix du mode de contrôle des signaux d'entrées/sorties de l'IP *preprocessing*, fait dans l'outil HLS, est semblable à celui mis en œuvre dans les composants communiquant avec celle-ci (cf. Figure 11), IP matérielles développées manuellement. Par contre, le formatage des signaux d'entrées/sortie de l'IP VHDL, réalisant la fonction de prétraitement, ne correspond pas à celui des signaux d'entrées/sortie des composants avec lesquels elle communique. Le composant d'adaptation contient donc une fonction de ré-ordonnancement du signal de la matrice H (*H_in* dans Catapult C, cf. Figure 72), du signal Rx (*D_in* dans Catapult C, cf. Figure 72) et un reformatage des signaux des sorties Sp et R (regroupé dans le signal *Main_Complete_Preprocessing_out_R* dans Catapult C, cf. Figure 72). Ce dernier est nécessaire pour adapter le signal de sortie à l'architecture parallélisée choisie pour la fonction de décodage MIMO (8 décodeurs Lattice Search travaillent en parallèle).

4.3.2 Validation

Comme nous l'avons expliqué précédemment (cf. section 4.2.3) les données proposées par l'outil HLS ne permettent pas directement de valider les performances (temporelles et surfaciques) du composant matériel généré. Pour cela, il faut passer par une phase de simulation pour obtenir les performances temporelles et par une phase de synthèse pour obtenir le nombre de portes logiques réellement utilisées par l'IP matérielle générée.

4.3.2.1 Simulation

Pour valider les performances temporelles de l'IP matérielle générée avec Catapult C, correspondant à la fonction de prétraitement du décodeur MIMO, une simulation de celle-ci est effectuée. Afin de se placer dans les mêmes conditions que la version existante développée manuellement, le vecteur de test, décrit en VHDL, utilisé est identique. L'interface de l'IP développée manuellement étant différente (agencement de signaux différents, nom des ports différents et ports de débogage supplémentaires) de l'interface du l'IP générée par l'outil HLS, une encapsulation de ce dernier est nécessaire pour adapter les signaux (cf. section 4.3.1).

L'outil ModelSim [122] de Mentor Graphics est utilisé pour exécuter la simulation. Afin de comparer les performances temporelles de l'IP générée par l'outil HLS, en d'autre terme, le temps de latence (nombre de cycles x durée d'un cycle) entre deux données successives en sortie de la fonction de prétraitement est relevé sur les chronogrammes résultants de la simulation.

Le Tableau 12 récapitule les données relevées pour les quatre cas précédemment présentés en comparaison avec le résultat obtenu avec l'IP développé manuellement et respectant les spécifications du système de transmission sans-fil : 96 données R et Sp successives en sortie de la fonction de prétraitement en un temps de 12 µs, soit 0,125 µs/données Rx).

Tableau 12. Latence pour obtenir une nouvelle donnée en sortie de la fonction de prétraitement.

	Cas de référence (1)	Cas 1	Cas 2	Cas 3	Cas 4
Temps de calcul pour 1 donnée Rx décodée (en µs)	0,125	48,98	27,65	40,74	17,19

(1) Latence obtenue pour l'IP de prétraitement développée à la main.

Nous pouvons constater un gain non négligeable entre les quatre architectures de l'IP matérielle de prétraitement (Cas 1 à 4). Le troisième cas est moins bon que le second cas du fait du temps d'accès aux mémoires RAM utilisées pour le stockage de certains éléments de la fonction. De plus, ces valeurs sont cohérentes avec le nombre de cycles de latence donné par l'outil HLS (cf. section 4.2.3). Au vu de ces résultats de simulation, même dans le dernier cas, les performances temporelles du composant matériel souhaitées ne sont pas atteintes. L'effort à fournir pour s'approcher de la valeur de référence reste important.

4.3.2.2 Synthèse logique

La seconde étape pour évaluer les performances de l'IP matérielle générée par l'outil HLS passe par la phase de synthèse logique de celle-ci. Cette étape consiste à transformer le modèle HDL en un circuit logique équivalent représenté sous forme de portes logiques. A l'issue de cette étape, le nombre de portes logiques utilisées par l'IP sera connu. Toutefois pour cette phase, l'IP de prétraitement n'a pas été synthétisée seule mais intégrée dans le décodeur MIMO complet écrit manuellement. Par conséquent une encapsulation de l'IP de prétraitement générée avec Catapult C a été réalise pour adapter son interface avec celles des composants avec lesquelles elle communique (cf. section 4.3.1). Enfin l'environnement de synthèse est l'outil QUARTUS II d'Altera [123] (choix en cohérence avec la cible technologique : FPGA d'ALTERA).

Le Tableau 13 récapitule les résultats de synthèse logique obtenus pour les différents cas de génération HLS précédemment présentés en comparaison avec le cas de référence et les capacités du FPGA ciblé.

Tableau 13. Résultats de synthèse logique pour le décodeur MIMO.

	Caractéristique du FPGA (1)	Cas de référence (2)	Cas 1	Cas 2	Cas 3
Conbinational ALUTs	270 400	54 289	58 924	49 854	37 373
Memory ALUTs	135 200	2 682	2 011	2 011	2 011
Dedicated Logic Registered	270 400	41 520	41 867	38 983	40 083
Total block memory bits	16 662 528	278 994	199 424	199 424	199 424
DSP block 18-bits elements	576	180	55	59	55

(1) Le FPGA Altera ciblé a pour référence Stratrix III EP3SL340F1517C.
(2) Résultat de la synthèse logique avec l'IP de prétraitement développée à la main.

Tout d'abord, le nombre d'ALUTs (*Adapative Look Up Table*) combinatoires évolue sensiblement selon les contraintes architecturales appliquées lors de la génération de l'IP matérielle *preprocessing* par le biais de l'outil HLS Catapult C (cas 1 à 4). L'architecture pipeline utilisée dans le second cas offre un gain de 16 % entre la première (cas 1) et la seconde architecture de l'IP. L'utilisation de blocs mémoires de type RAM, dans la troisième architecture de l'IP (cas 3), au lieu de registres dans la première architecture (cas 1), permet de porter de ce gain à 37%. Par contre l'architecture parallélisée du quatrième cas réduit le gain à 30% au profit de performances temporelles meilleures (cf. section 4.3.2.1).

Ensuite, au niveau du nombre d'éléments mémoires et de blocs DSP utilisés, les choix architecturaux choisis n'ont que peut d'impact sur ces éléments.

Enfin, les résultats de synthèse du décodeur MIMO, intégrant l'IP de prétraitement générée par l'outil HLS, sont sensiblement meilleurs que la synthèse de ce même décodeur MIMO développé complètement manuellement. Le nombre d'éléments combinatoires, de blocs mémoires et d'éléments de calcul dédiés sont plus faibles. Ainsi l'espace occupé dans le FPGA par le décodeur MIMO intégrant l'IP de prétraitement générée depuis un outil HLS est plus faible que la version initiale développée manuellement. Ce qui met en évidence l'efficacité de l'outil HSL à générer une architecture compact. Cependant, il faut admettre que ce bénéfice ne compense pas des performances temporelles moins bonnes que celles de la version développée manuellement.

En conclusion, l'impact des différentes contraintes appliquées sur la génération de l'IP VHDL apparaît clairement dans les résultats de simulation et de synthèse. Il est nécessaire de choisir le meilleur compromis entre la surface et les performances temporelles évaluées lors de la phase de simulation.

4.4 Conclusion

Dans ce chapitre nous proposons un essai de couplage d'une approche de co-conception de systèmes électroniques embarqués basées sur l'approche MDA avec le concept de synthèse de haut niveau (ou synthèse comportementale) pour générer du code matériel.

Au vu des résultats obtenus avec le décodeur MIMO, nous pouvons indiquer que les performances temporelles d'une IP matérielle générée par un outil de synthèse de haut niveau n'atteignent pas celles d'une IP matérielle codée manuellement par un ingénieur expérimenté. Ce constat n'est finalement pas surprenant au regard de ce qui a pu être constaté lors du passage d'une description sous la forme de portes logiques à une description HDL du matériel : le gain de productivité s'est traduit par une perte de performances. Il en est de même avec l'utilisation des outils HLS qui monte encore d'un cran le niveau d'abstraction afin de proposer un concept d'aide à la génération automatique de code HDL. Cependant l'écart de performances des IPs générées avec les outils HLS nous semble encore trop important pour utiliser ces derniers massivement dans un flot de co-conception. Leur maturité fait que leur utilisation est plutôt ponctuelle sur un sous-ensemble d'un système. Ce constat rejoint les conclusions du rapport de l'ITRS realisé en 2009 [4] qui souligne, je cite « *[…] although behavioral synthesis is essential to system-level design, efficient behavioral synthesis is not yet realized today,*

despite having been a research topic for more than a decade […] ». Nous espérons dans les années à venir voir évoluer sensiblement les performances des outils HLS.

Nous ajoutons que les performances des IP matérielles générées avec un outil HLS dépendent non seulement de l'outil mais aussi de la complexité du code en entrée. Comme, nous l'avons déjà dit dans ce chapitre, une architecture fonctionnelle bien pensée permet de faciliter le travail de l'outil HLS. Cette remarque n'est pas anodine dans le contexte d'un couplage avec un flot de co-conception MDA. L'utilisation d'un tel outil nécessite une réflexion en amont dès la phase de modélisation UML.

De plus, les performances des IPs matérielles générées sont également liées à une utilisation intelligente des différents paramètres de l'outil HLS. L'utilisateur se doit d'anticiper le résultat que pourra générer l'outil HSL en influant sur tel ou tel paramètre (interfaces de l'IP, agencement de la mémoire, manipulation des boucles, etc.). De ce fait, que ce soit de manière indépendante ou couplé à un flot de co-conception MDA, un outil HLS doit être manipulé par un ingénieur ayant des compétences en développement matériel.

Notre principal objectif dans le cadre de cette thèse ne fut pas de démontrer ou décrédibiliser les performances des outils de synthèse de haut niveau mais de mettre en avant l'intérêt de coupler de tels outils à une de conception MDA multi-niveau telle que MOPCOM.

Ce qui nous semble intéressant à souligner ici, avec ce type d'outil, c'est la facilité et la rapidité avec laquelle un utilisateur pourra explorer et comparer différentes architectures pour une IP donnée. De plus, le flot de vérification intégré aux outils HLS offre un gain de temps considérable et permet également d'éviter des erreurs de codage, d'autant plus que cela ne nécessite pas de développer un vecteur de test décrit en VHDL. Ces caractéristiques apportent indéniablement un confort et un gain de productivité.

Avec l'utilisation d'outils HLS, nous pouvons alors parler d'exploration d'architecture microscopique tandis que le flot de co-conception MDA permet une exploration d'architecture au niveau macroscopique. Il y a donc bien une complémentarité et un intérêt à coupler une méthodologie MDA avec une approche de synthèse comportementale.

Dans tous les cas, il convient de trouver le meilleur compromis quant à l'utilisation d'outils HLS couplés à un flot de co-conception MDA. Plusieurs possibilités sont à envisager selon les besoins. D'un côté, le concepteur peut se contenter des performances de l'IP générée par l'outil HLS (qui doit tout de même respecter les contraintes minimales pour que le système global fonctionne) et intègre celle-ci dans le design généré par le biais du flot MDA (utilisation du générateur UML vers VHDL). D'un autre côté, il peut faire le choix de ne pas intégrer cette IP mais de se baser sur l'architecture de celle-ci pour écrire une version optimisée.

Chapitre 5 Modélisation des systèmes reconfigurables

Sommaire

Nous venons de présenter nos travaux permettant d'aboutir à une méthodologie de co-conception pour les systèmes électroniques embarqués basée sur des modèles. Le résultat de ces travaux est déjà en soi un défi que nous avons partiellement relevé en éprouvant les limites des solutions actuelles et en traçant les grandes lignes des futures méthodologies de co-conception.

Dans ce nouveau chapitre, nous présentons nos travaux sur la modélisation de systèmes reconfigurables, ce qui englobe reconfiguration de l'application et/ou reconfiguration de la plate-forme matérielle. Pour cela, nous proposons une extension de la méthodologie MOPCOM présentée dans le chapitre 3.

Dans ce chapitre, nous mettons l'accent sur la modélisation d'un système reconfigurable dans un contexte radio logicielle.

5.1 Un besoin de flexibilité

Comme nous l'avons déjà indiqué à de nombreuses reprises tout au long de ce mémoire, les systèmes électroniques embarqués sont de plus en plus complexes (fonctionnalités, plate-forme hétérogène, nombreuses contraintes telles que la surface, la consommation, etc.). De plus, notamment dans le contexte des systèmes de radiocommunications, le manque de souplesse de ces derniers est de plus en plus vu comme un frein à leur évolution. Ainsi le besoin de plus de flexibilité se fait ressentir. Par nature, la flexibilité engendre un surcoût de complexité (gestion de la flexibilité) : on parle de surcoût de reconfiguration (« overhead » en anglais). Seule une économie d'échelle peut apporter un gain, qui sera d'autant plus significatif que la reconfigurabilité sera bien gérée dans le système.

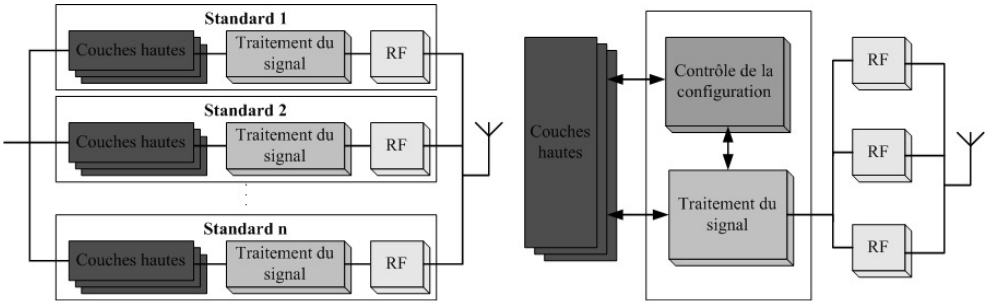


Figure 77. Architecture multistandard classique vs architecture multistandard RLR [26].

Le synoptique à gauche de la Figure 77 illustre l'architecture d'un système de radiocommunication mettant en œuvre plusieurs standards de communication (GSM, UMTS, et WiFi par exemple). Dans une telle architecture, il y a autant de chaîne de traitement du signal numérique que de standards, ce qui engendre une duplication des ressources de calcul. L'approche prônée par la RLR (*Radio Logicielle Restreinte*), présentée dans le chapitre 1, consiste à proposer une architecture matérielle commune (cf. synoptique à droite de la Figure 77) pour la chaîne de traitement du signal numérique de l'ensemble des standards de communication supporté par le système. Pour cela une ressource matérielle reconfigurable commune est utilisée pour mettre en œuvre les différents standards.

Afin de bien cerner la problématique et les enjeux de la reconfigurabilité, nous présentons les principaux cas de reconfiguration possible pour un système RL (*Radio Logicielle*) ou RI (*Radio Intelligente*).

5.1.1 Un système reconfigurable

Présentée dans le chapitre 1, la RL et la RI poussent à son extrême le principe de flexibilité. La RL se caractérise par une mutualisation de la chaîne de traitement du signal pour différents standards de communication. De plus l'ensemble des traitements est effectué de manière logicielle dès la sortie de l'antenne, ce qui lui confère une grande flexibilité. Un équipement RI est un système capable de s'adapter à son environnement tout en prenant en compte ces capacités matérielles et offrir une qualité de service (QoS) optimale à son utilisateur ou au réseau en général. Pour cela, un équipement RI permet de basculer d'un standard de communication à un autre sans rompre le service : c'est ce qu'on appelle le « vertical handhover » [29].

Un système RI se caractérise par :

- une application reconfigurable ;
- une plate-forme matérielle reconfigurable capable de supporter une application reconfigurable.
- des moyens pour activer la reconfiguration de manière autonome (capteurs et décision).

Note : Dans le contexte de système de radiocommunications, l'application déployée sur la plate-forme matérielle est la chaîne de traitement du signal radio.

5.1.1.1 Une application reconfigurable

Une application statique

Avant de considérer une application reconfigurable, rappelons ce que nous entendons par une application statique. La Figure 78 illustre l'architecture d'une chaîne de traitement du signal statique. L'ordonnancement des tâches ne varie pas dans le temps et chacune des tâches est identique à chaque appel de celle-ci.

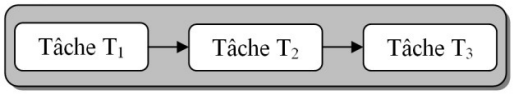


Figure 78. Architecture d'une chaîne de traitement du signal statique.

Une application reconfigurable

Une application reconfigurable se caractérise par le fait qu'au moins un des opérateurs d'une des tâches de la chaîne de traitement du signal soit reconfigurable. La reconfigurabilité peut être considérée selon trois niveaux de granularité en fonction du type de(s) changement(s) apporté sur les tâches (ou mode opératoire) devant être effectuées.

Le niveau de granularité le plus élevé, illustré par la Figure 79, consiste à reconfigurer l'ensemble de la chaîne de traitement du signal. Ce cas correspond à un basculement d'un standard de communication à un autre lorsque les tâches entre les différents standards sont sensiblement différentes. Par conséquent, la mutualisation de la chaîne de traitement du signal n'apporte que peu d'intérêt dans un tel cas.

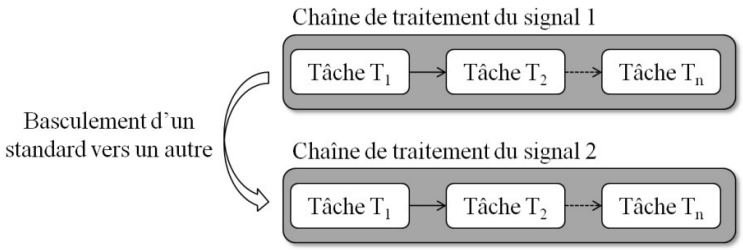


Figure 79. Représentation d'une application reconfigurable.

Le second niveau de granularité se caractérise par une reconfiguration de certaines tâches de la chaîne de traitement du signal et non l'ensemble de celle-ci. Dans ce cas, certaines des tâches restent similaires quels que soient l'environnement et l'état du système, ce qui permet de limiter l'impact de la reconfiguration et donc son « overhead ».

La Figure 80 illustre le cas où une seule des tâches de la chaîne de traitement du signal est reconfigurée : la tâche T_2 est reconfigurée en la tâche T_2', et vice-versa, suivant le contexte dans lequel se trouve le système.

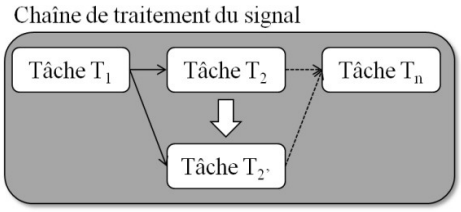


Figure 80. Spécification d'une chaîne de traitement du signal dont une tâche est reconfigurable.

Dans ce cas de reconfiguration de la chaîne de traitement du signal, la cohérence des interfaces entre les différentes tâches doit être garantie. Autrement dit, les tâches T_2 et T_2' doivent avoir les mêmes interfaces d'entrée/sortie. Le plus important est que le chemin de données soit conservé. Il faut donc réacheminer les données d'entrées/sorties aux bons endroits (aux adresses dans le cas de traitement effectué sur un processeur et sur les pistes dans le cas de traitement effectué sur FPGA).

Le troisième niveau de granularité se caractérise par une reconfiguration des opérateurs d'une tâche. Dans ce cas-là, il y a un très fort recouvrement des différentes tâches et des différentes opérations de ces tâches quel que soit le contexte dans lequel se trouve le système. Seule une reconfiguration fine est effectuée au niveau d'une (ou plusieurs) opération(s) d'une tâche (ou plusieurs tâches) de la chaîne de traitement du signal.

Note : la recherche de recouvrement maximal des opérateurs est un thème de recherche en soi, tel qu'abordé dans les études d'autres doctorants de l'équipe SCEE de Supélec [124][125][126].

La Figure 81 illustre le cas où l'un des opérateurs d'une des tâches est reconfiguré : la tâche T_2 est constituée de trois opérateurs, dont l'opérateur Op_2 qui peut être reconfiguré en l'opérateur Op_2', et inversement, suivant le contexte dans lequel se trouve le système.

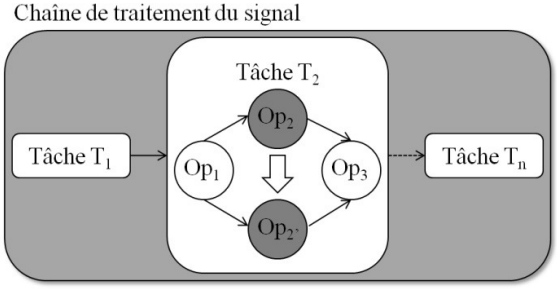


Figure 81. Spécification d'une chaîne de traitement du signal avec une reconfiguration d'opérateurs au sein d'une tâche en cours de fonctionnement.

Ces trois niveaux de reconfiguration d'une chaîne de traitement du signal (application) sont complémentaires et permettent d'exprimer tous les scénarii possibles d'un changement de modes opératoires d'un système suivant le contexte dans lequel il se trouve. Plus le niveau de granularité est fin, plus la reconfiguration de la chaîne de traitement du signal pourra être rapide et ainsi limitera l'overhead.

5.1.1.2 Une plate-forme matérielle reconfigurable

Afin de supporter une chaîne de traitement du signal (application) reconfigurable, la plate-forme matérielle doit supporter une telle caractéristique. Pour cela, la plate-forme matérielle d'un système reconfigurable doit être constituée au moins d'un processeur (GPP, DSP, voire GPU) et/ou d'un composant reprogrammable de type FPGA. Dans le cas d'un processeur, le firmware exécuté peut-être reprogrammé. Dans le cadre de cette thèse, nous nous sommes concentrés sur une plate-forme matérielle reconfigurable basée sur un FPGA, plate-forme typique dans un contexte RLR.

Comme pour l'application, un FPGA peut être reconfiguré à différents niveaux de granularité (dans sa globalité ou partiellement). Dans le contexte radio, le premier cas est principalement utilisé lorsque les chaînes de traitement du signal de deux standards sont très peu mutualisables. De plus, suivant la complexité de la chaîne de traitement du signal, la durée de reconfiguration complète d'un FPGA peut amener à une rupture du service. Dans un contexte RLR, la reconfiguration partielle est préférée.

Aujourd'hui, la reconfiguration dynamique partielle de FPGA n'est supportée que par certaines familles de FPGA de la société Xilinx [127]. La technologie proposée par Xilinx permet de reconfigurer une zone du FPGA en cours de fonctionnement du système embarqué. Lors de la phase de conception de la plate-forme matérielle, il est nécessaire de définir une unique région statique (RS) et une (ou plusieurs) région(s) reconfigurable(s) partiellement (RRP). Plus la RRP est petite, moins l'overhead (en termes de temps de reconfiguration, surcoût de consommation) dû à la reconfiguration aura un impact significatif 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). Suivant le contexte dans lequel se situe le système embarqué, la zone RRP en question est chargée par le fichier binaire du nouveau contexte. Cette technologie a été étudiée et améliorée par l'équipe SCEE [26][32]. La Figure 82 illustre un exemple d'architecture d'un FPGA constituée d'une région statique (composants en blanc et bus de communication) et de deux RRPs.

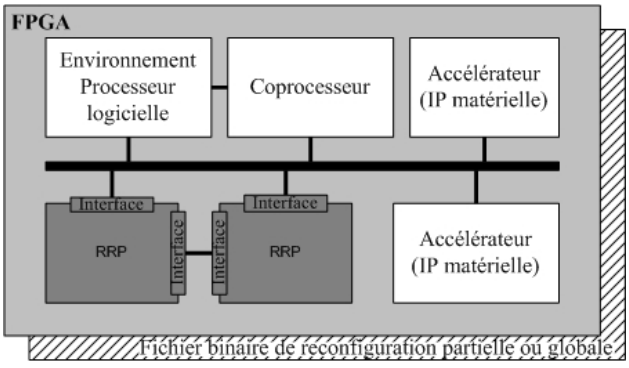


Figure 82. Exemple d'une architecture interne d'un FPGA constituée d'une région statique et de deux régions RRP.

5.1.1.3 Des scénarii de systèmes reconfigurables

De nombreux scénarii de reconfiguration sont possibles suivant les spécifications du système embarqué. Nous donnons ici deux exemples de scénario de reconfiguration de système embarqué, exemples pour lesquels la plate-forme matérielle est basée sur un FPGA.

La Figure 83 illustre le cas où le système met en œuvre une reconfiguration matérielle tout en ayant un mode opératoire statique.

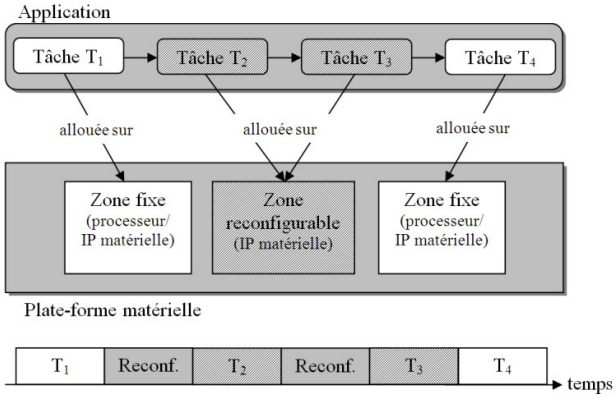


Figure 83. Scénario de reconfiguration où seule la plate-forme matérielle est reconfigurée.

Les tâches T2 et T3 sont successivement exécutées sur la même zone matérielle. Ce scénario permet un gain en termes de surface matérielle utilisée. Il faut tout de même que le temps de reconfiguration de la zone matérielle reconfigurable soit en adéquation avec les contraintes temporelles du système.

La Figure 84 illustre le cas où le système lie une reconfiguration de l'application avec une reconfiguration de la plate-forme matérielle.

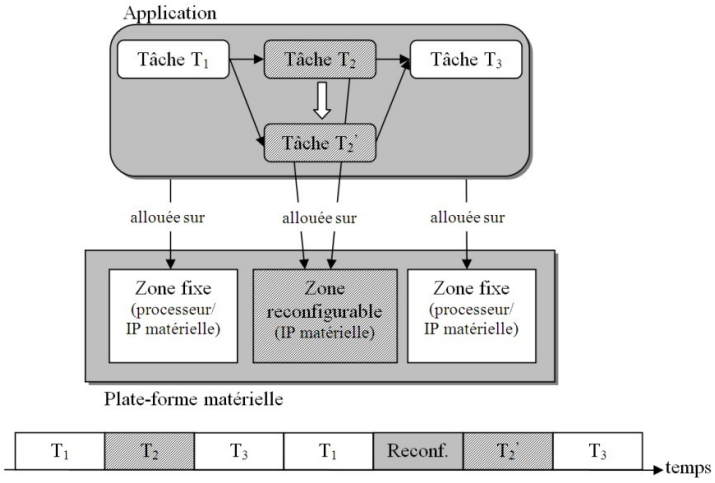


Figure 84. Scénario de reconfiguration dont l'application nécessite un changement de mode opératoire qui engendre une reconfiguration matérielle.

La Figure 84 illustre uniquement le cas où un changement de tâche applicative est effectué en cours de fonctionnement. Toutefois les deux autres cas de reconfiguration applicative sont également susceptibles d'être mis en œuvre avec une reconfiguration matérielle.

On peut bien évidement imaginer tous les degrés possibles de combinaison de reconfiguration du système embarqué.

5.1.2 Un gestionnaire de reconfiguration

La reconfiguration du système est d'autant plus complexe que la plate-forme matérielle supportant l'application est hétérogène. Par conséquent, afin de contrôler le mécanisme de reconfiguration du système et l'impact que celle-ci peut avoir sur le fonctionnement du système, il est nécessaire d'intégrer une gestion rigoureuse de celle-ci. En effet, en aucun cas, une reconfiguration du système, qu'elle soit logicielle ou matérielle, ne doit dégrader, voire interrompre le service proposé par le système embarqué. Dans un contexte de radiocommunication, la reconfiguration du système ne doit en aucun cas engendrer une perte d'information (interruption de communication). Bien que l'utilisation d'un gestionnaire de reconfiguration soit évidente, il existe encore peu de travaux aboutis sur ce sujet.

L'architecture de gestion de référence dans la communauté de la RL est le SCA (*Software Communication Architecture*) [23], évoquée dans le chapitre 1. L'utilisation de cette architecture est même requise pour les systèmes de radiocommunications militaires destinés aux armées américaines. Cependant le SCA est plus une abstraction de la plate-forme matérielle qu'une véritable architecture de gestion de la reconfiguration. Le SCA permet aux équipes de conception de s'affranchir de l'hétérogénéité de la plate-forme matérielle. Le seul aspect dynamique du SCA réside dans le déploiement d'une application (forme d'onde dans le vocabulaire militaire) sur une plate-forme matérielle hétérogène. De notre point de vue, le SCA est un peu trop réducteur par rapport aux nombreux cas de reconfiguration, évoqués précédemment dans la section 5.1.1, que nous voudrions voir supporter par une architecture de gestion de la reconfiguration d'un système de radiocommunication. En outre le SCA, basé historiquement sur la technologie CORBA [128], est inadapté à la gestion de la reconfiguration des circuits spécifiques du type DSP ou FPGA. De nombreuses études [129][130] tentent de contourner cette limitation depuis 10 ans, mais sans réel succès. Si bien qu'une refonte du SCA est envisagée sous le nom « SCA Next ».

Aujourd'hui, il existe peu d'architecture de gestion de reconfiguration qui propose, au-delà d'une simple gestion de reconfiguration, une gestion intelligente (analyse de métriques et prise de décisions de reconfiguration) de celle-ci. Nous pouvons tout de même citer les travaux de l'UPC de Barcelone [131] qui intègre la notion de prise de décision de la reconfiguration, concept qui reste encore à l'état d'embryon.

Les travaux de Jean-Philippe DELAHAYE [26] puis de Loïg GODARD [27], au sein de l'équipe SCEE de Supélec, ont abouti à une architecture de gestion de reconfiguration intitulée HDCRAM (*Hierarchical and Distributed Cognitive Radio Management*) capable de répondre aux nombreuses situations de reconfigurations.

L'originalité de l'architecture HDCRAM est fondée sur :

- D'un côté, la séparation du chemin de données et du chemin de reconfiguration afin d'offrir une meilleur flexibilité, mais aussi de proposer une portabilité et une réutilisabilité meilleure. En revanche, cela ajoute des contraintes de synchronisation entre les différents modules à reconfigurer ;
- D'un autre côté, l'instauration d'une gestion intelligente de la reconfiguration en se basant sur une gestion de prise de décision.

Evidemment, la mise en œuvre d'une telle architecture au sein d'un système reconfigurable accroît sa complexité, et peut occuper une place (en surface et/ou en temps) non négligeable. C'est le prix à payer pour disposer d'un système reconfigurable performant. Il va de soi que la « perte » de place due à l'ajout de cette architecture HDCRAM est justifiée par rapport au gain apporté par la flexibilité du système sur ses performances globales.

L'architecture HDCRAM, présentée sur la Figure 85, est composée de deux branches sur 3 niveaux hiérarchiques :

- Une branche descendante, intitulée ReM (*Reconfiguration Manager*), qui transmet l'ordre de reconfiguration depuis un niveau abstrait (connaissance de l'état global du système et de son environnement), totalement indépendant de la plate-forme matérielle cible, vers les opérateurs de traitements des données ;
- Une branche ascendante, intitulée CRM (*Cognitive Radio Management*), qui prend la décision de reconfiguration du système, depuis la capture de métriques, en passant par l'analyse jusqu'à la prise de décision.

Cette architecture est répartie suivant trois niveaux hiérarchiques afin d'offrir plusieurs niveaux de granularité de reconfiguration.

Pour permettre son utilisation par la communauté de la RI, mais aussi dans tout autre domaine, Loïg GODARD a développé un méta-modèle de cette architecture HDCRAM [28] à l'aide d'un langage de modélisation standard (issu du domaine de l'informatique) : UML. Par défaut, l'UML n'étant pas simulable, le méta-modèle a été complété en utilisant le langage Kermeta [43][132]. Déjà présenté dans le chapitre 2, nous rappelons que Kermeta est non seulement un langage, basé sur EMOF (*Essential Meta Object Facility*), mais aussi un environnement de développement et de simulation pour méta-modèle, intégré dans Eclipse sous la forme d'un plug-in. Ce langage permet non seulement de décrire la structure d'un méta-modèle mais aussi de décrire le comportement de celui-ci et d'exécuter une instance du méta-modèle. L'utilisation de Kermeta pour rendre le méta-modèle d'HDCRAM exécutable a également permis de mettre en œuvre des scénarii afin de valider différents cas d'utilisation de cette architecture.

Note : l'utilisation de Kermeta pour proposer un méta-modèle exécutable de l'architecture HDCRAM n'a en aucun cas influé sur le choix d'utiliser Kermeta dans le cadre de la méthodologie MOPCOM (cf. chapitre 2), où il est utilisé à un autre niveau.

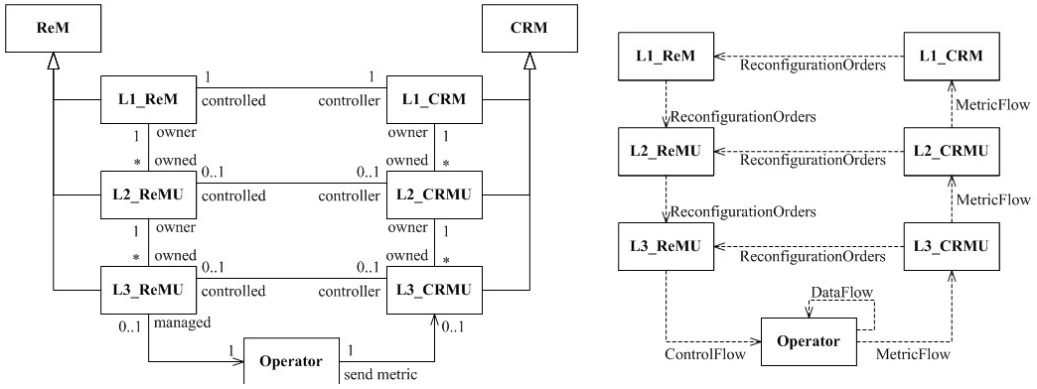


Figure 85. Méta-modèle et flot de données de l'architecture HDCRAM.

A son niveau de développement avant cette thèse, l'architecture HDCRAM permettait uniquement de capturer et de représenter des notions (structurelles et comportementales) de très haut niveau. Bien que validée par des scenarii significatifs abstraits, l'architecture HDCRAM n'a jamais été placée dans un contexte de conception d'un système embarqué réel. En outre, à l'instar du SCA, ni l'architecture HDCRAM, ni son méta-modèle, ne spécifie la manière de la mettre en œuvre ; et ceci de manière volontaire afin de ne pas imposer des contraintes irrémédiables, et ainsi de permettre son adaptation lors des futurs évolutions technologiques. Ce constat nous amène à nous poser les questions suivantes :

- Comment peut-elle être intégrée dans un flot de conception tel que MOPCOM ?
- A quel niveau d'abstraction la situer dans un flot de conception tel que MOPCOM ?

Nos travaux présentés dans la suite de ce chapitre tentent de répondre à ces questions.

5.2 Une modélisation des systèmes embarqués reconfigurables

Afin de prendre en compte le caractère reconfigurable d'un système embarqué temps réel, et plus particulièrement dans le contexte de la radio logicielle, il nous fallait d'une manière ou d'une autre intégrer cet aspect dans la phase de modélisation de systèmes électroniques embarqués. Pour cela, nous proposons une extension de la méthodologie de co-conception MOPCOM.

La spécification de systèmes électroniques embarqués reconfigurables est un problème complexe. Aujourd'hui quelques études existent ; celles-ci permettent la modélisation du caractère dynamique de l'application et de l'architecture dans le contexte des systèmes électroniques embarqués. Ces approches utilisent ou non le langage UML mais ont en commun d'exprimer la reconfigurabilité des systèmes. Actuellement, la reconfiguration dynamique de FPGA n'est pas modélisée, mais directement mise en œuvre sur circuit afin de permettre une réduction de la surface et/ou de la consommation. Sa mise en œuvre est souvent réalisée de façon « ad-hoc » et aucun formalisme n'est utilisé lors de l'étape de spécification. Toutefois un certain nombre d'équipes de chercheurs proposent des solutions pour prendre en compte une telle caractéristique dans un flot de développement basé sur les modèles.

5.2.1 Quelques propositions

L'une des toutes premières propositions pour la modélisation de systèmes embarqués reconfigurable émane du projet européen ANDRES [133][134]. La méthodologie permet de modéliser à la fois une application reconfigurable et une plate-forme hétérogène reconfigurable. La modélisation est basée sur une extension de la librairie SystemC intitulé OSSS+R (*Oldenburg System Synthesis Subset+Reconfigurable*) qui propose une adaptation du concept de polymorphisme et une abstraction de la reconfiguration matérielle. L'inconvénient majeur de la méthodologie ANDRES est le manque de portabilité des modèles car la librairie OSSS+R est propriétaire.

Dans [135], les auteurs utilisent également le SytemC pour modéliser des systèmes reconfigurables. Ils modifient le moteur d'exécution SystemC pour pouvoir simuler les temps de reconfiguration d'un système reconfigurable. Comme pour ANDRES, la portabilité des modèles est restreinte car les modifications apportées au noyau SystemC ne sont pas standardisées.

Dans [136], les auteurs utilisent les diagrammes de séquence UML pour indiquer la configuration courante (DBPSD – *Dynamic Bitstream Partitioning on Sequence Diagrams*). Le système embarqué est composé d'un processeur et d'un accélérateur matériel (FPGA) mais celui-ci ne met pas en œuvre la reconfiguration dynamique partielle. Chaque appel dans le diagramme de séquence est stéréotypé avec un numéro de configuration, qui déclenche la reconfiguration.

Basée sur la méthodologie MSCE [83], présentée dans le chapitre 2, Anthony BARRETEAU propose dans sa thèse [137] une approche pour intégrer la prise en compte de la reconfiguration, principalement dans un contexte radio logicielle. Sa proposition utilise une modélisation transactionnelle du système et une modélisation des propriétés non-fonctionnelles (estimation des heures de début et de fin des tâches, estimation du temps d'exécution des tâches, estimation du temps de reconfiguration). La modélisation se fait par le biais de l'outil Cofluent Studio [84], modélisation qui est propriétaire. Puis une génération SystemC est réalisée pour simuler le modèle et évaluer les performances de ce dernier.

Très impliquée dans le domaine de la modélisation, l'équipe DaRT du LIFL propose une extension du profil MARTE afin de représenter des systèmes reconfigurables [138]. Jusqu'à présent les travaux de DaRT se sont principalement focalisés sur la modélisation de la plate-forme avec une description explicite des ressources matérielles nécessaires à la mise en œuvre de la reconfiguration dynamique partielle d'un FPGA (par exemple, le contrôleur ICAP[14] est explicitement modélisé). Lors de la phase d'allocation de l'application sur la plate-forme matérielle, des stéréotypes spécifiques sont utilisés afin d'indiquer le caractère reconfigurable de l'exécution.

[14] ICAP (*Internal reConfiguration Access Port*) : module, embarqué dans le composant FPGA, qui permet de charger les images du FPGA depuis une mémoire externe ; module nécessaire à la mise en œuvre de la reconfiguration dynamique partielle d'un FPGA de Xilinx.

5.2.2 Une extension de la méthodologie MOPCOM

Nous ne remettons pas en cause les résultats et la qualité des travaux effectués sur la modélisation de systèmes flexibles (reconfigurables). Cependant, les différentes approches proposées jusqu'à présent ne prennent pas en compte tous les cas de reconfigurations possibles. De plus, certains de ces travaux de modélisation de systèmes reconfigurables ne se basent pas sur un langage de modélisation standardisé ou proposent une modification du langage standard utilisé. C'est pourquoi, afin de répondre à l'ensemble des caractéristiques d'un système flexible, nous proposons une solution qui se base sur une modélisation standardisée, qui se veut portable et évolutive. Nous tenons également à préciser que nous avons utilisé dans notre solution des idées proposées dans par ANDRES et DaRT.

Nous exposons dans ce chapitre une extension de la méthodologie MOPCOM (cf. chapitre 3) pour les systèmes flexibles.

5.2.2.1 Modélisation de l'application (PIM)

Pour modéliser le caractère reconfigurable d'une application, il nous est apparu évident d'exploiter les concepts d'héritage et de polymorphisme issus de la programmation objet, et de coupler ceux-ci avec la méthode dite des design pattern[15]. L'application est reconfigurable si son comportement évolue en fonction des paramètres extérieurs et/ou intérieurs au système. Illustré sur la Figure 86, le design pattern « Stratégie » [139] semble le mieux adapté pour modéliser une application reconfigurable. En effet, pour un contexte donné (contexte = scenario décrit en UML), le système applique une stratégie appropriée (stratégie = suite de tâches, constituées d'algorithmes, répondant au scénario décrit en UML) qui met en œuvre l'algorithme adapté. Ce qui revient à dire : un contexte/un scénario → une instance de l'application ; pour chaque contexte/scénario → une instance de l'application différente.

Nous montrons, au travers de nos travaux, que le couplage du design pattern « Stratégie » et du concept d'héritage permet de modéliser simplement les différents cas d'applications reconfigurables présentés précédemment.

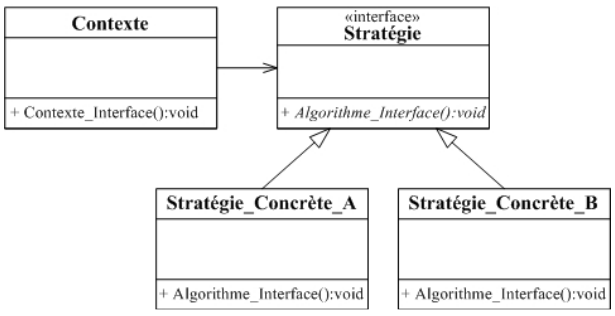


Figure 86. Méta-modèle du design pattern « Stratégie ».

[15] En informatique, le design pattern est un concept génie logiciel destiné à résoudre des problèmes récurrents suivant le paradigme objet. Les patrons de conceptions tirent leur origine des travaux de l'architecte Christopher ALEXANDER dans les années 70 : « *Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution* ».

De plus, comme nous l'avons déjà indiqué précédemment, une application reconfigurable doit intégrer un mécanisme qui va gérer ces caractéristiques de reconfiguration. Dans la section 5.1.2, nous avons succinctement présenté l'architecture HDCRAM proposée par l'équipe SCEE de Supélec. Cette architecture permet notamment, par sa branche ReM, de gérer la reconfiguration d'un système électronique embarqué temps réel. Nous avons donc couplé le modèle de l'application au modèle de gestion de la reconfiguration HDCRAM.

Cette modélisation est supportée par la méthodologie MOPCOM.

5.2.2.2 Modélisation de la plate-forme matérielle (PM)

Dans la section précédente, nous avons présenté et illustré notre vision et notre solution de modélisation d'une application reconfigurable. Dans un système, une application ne peut fonctionner sans une plate-forme d'exécution. Comme mentionné dans la section 5.1.1, la plate-forme matérielle peut être elle-même reconfigurable.

Dans le cas où celle-ci est constituée uniquement d'unités de calcul de type processeur (généraliste ou dédié) la modélisation proposée dans la méthodologie MOPCOM se suffit à elle-même. En effet, les éléments du profil MARTE permettent de modéliser un processeur, et ses caractéristiques. La programmabilité de celui-ci sera modélisée au niveau du PIM mais également par le modèle d'allocation (PSM) sur lequel le stéréotype « *Allocate* » du profil MARTE est utilisé. Associé à ce dernier, le tag *AllocationNature* permet d'indiquer s'il s'agit d'une allocation spatiale (i.e. statique) ou d'une allocation temporelle (i.e. partielle). En effet, même si les instructions du processeur sont figées lors de sa conception, le code exécuté sur ce processeur peut-être reprogrammé. Le code/programme exécuté sur le processeur est soit directement (re)programmable, ou, indirectement par le biais du système d'exploitation.

En revanche, dans le cas où la plate-forme est constituée d'un composant FPGA, et plus particulièrement d'un FPGA de la société Xilinx disposant de la technologie de reconfiguration dynamique partielle, il est impératif de faire apparaître les éléments matériels nécessaires à la mise en œuvre de cette reconfiguration dynamique partielle au sein du composant. La plate-forme matérielle doit en outre être constituée d'un FPGA associé à une mémoire externe de type RAM, celle-ci servant au stockage des fichiers binaires de configuration du FPGA. De plus, l'architecture implantée au cœur du FPGA, doit disposer d'une interface ICAP et d'un processeur logiciel (MicroBlaze [140] de Xilinx par exemple), qui gère l'aspect matériel de la reconfiguration.

Les caractéristiques reconfigurables de la plate-forme matérielle apparaissent uniquement au dernier niveau de modélisation MOPCOM : au niveau DML (*Detailed Modeling Level*). En effet, au niveau AML (*Abstract Modeling Level*), le modèle de plate-forme (PM) n'est qu'un modèle d'exécution qui ne reflète pas la topologie de la plate-forme réelle (cf. section 3.1.2.1 du chapitre 3). Au niveau de la modélisation intermédiaire, EML (*Execution Modeling Level*), le modèle PM fait apparaître la topologie de la plate-forme réelle mais aucune distinction n'est faite entre un composant de type processeur et un composant de type programmable (FPGA). Le raffinement du modèle PM au niveau DML fait apparaître le type des différents composants et leurs caractéristiques.

Par conséquent si la plate-forme matérielle contient un FPGA, qui plus est disposant de la technologie de reconfiguration dynamique partielle, alors le modèle devra contenir les éléments nécessaires à la mise en œuvre cette reconfiguration dynamique partielle.

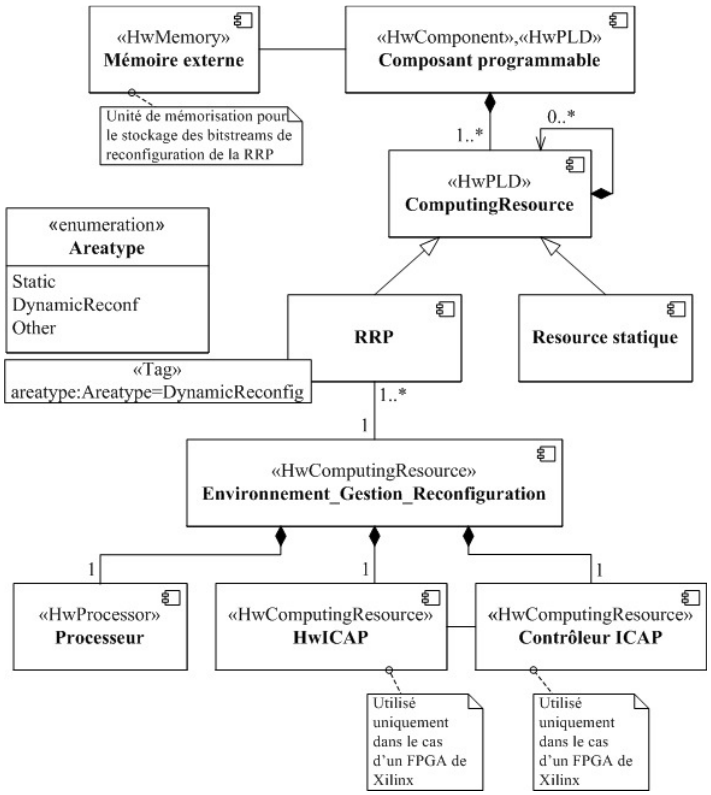


Figure 87. Méta-modèle d'une architecture matérielle reconfigurable d'un FPGA.

Nous nous sommes appuyés sur les travaux de l'équipe DaRT afin de modéliser une plate-forme matérielle reconfigurable composée principalement d'un composant de type FPGA. La Figure 87 présente une méta-modélisation d'une architecture reconfigurable matérielle. Si l'application est reconfigurable, la plate-forme cible doit disposer d'élément(s) reconfigurable(s). De plus, même si l'application n'est pas reconfigurable la plate-forme matérielle peut supporter la reconfiguration (cas du scénario illustré par la Figure 83). Dans ces deux cas, un environnement de gestion de la reconfiguration est impératif, notamment pour supporter l'architecture HDCRAM. Si la plate-forme n'est pas constituée de FPGA, l'environnement de gestion de la reconfiguration sera principalement constitué d'un processeur, sur lequel sera mise en œuvre l'architecture HDCRAM adaptée au système. L'environnement de gestion de reconfiguration doit pouvoir récupérer le(s) programme(s) (binaires) à charger. Les différents binaires (différentes configurations du FPGA) sont stockés soit dans une mémoire intégrée à la plate-forme matérielle, ou déportée sur un serveur de stockage. De plus, dans le cas d'un FPGA, cet environnement de gestion de reconfiguration est plus complexe car il faut gérer le mécanisme de reconfiguration de la zone à reconfigurer. Dans ce cas précis, le modèle de la plate-forme fait apparaître le composant ICAP et son contrôleur associé. Le méta-modèle proposé est hiérarchique. En effet, en fonction de l'utilisation du niveau de modélisation DML, il ne sera pas toujours nécessaire de faire clairement apparaître sur le modèle UML le détail de l'environnement de gestion de reconfiguration. Ce choix est laissé libre aux concepteurs.

Note : Nos travaux sur les systèmes reconfigurables se sont concentrés uniquement sur l'étape de modélisation dans le processus de développement. Nous n'avons pas traité l'étape de mise en œuvre (génération de code HDL) de plate-forme matérielle reconfigurable, notamment constituée d'un composant de type FPGA. Pour cela nous vous orientons vers les travaux de Jorgiano VIDAL [105] qui a proposé dans le cadre du projet MOPCOM une génération de scripts dédiés aux outils de la chaîne EDK de Xilinx à partir du niveau DML.

5.2.3 Modélisation d'un système RLR à l'aide de MOPCOM

Pour valider la mise en œuvre de l'architecture HDCRAM dans la méthodologie MOPCOM, et la modélisation d'une plate-forme matérielle reconfigurable, nous nous sommes appuyés sur la modélisation d'un système RLR.

5.2.3.1 Le système RLR modélisé

Le système RLR, présenté dans le chapitre 1, se caractérise par une application configurable et une plate-forme matérielle reconfigurable. La Figure 88 rappelle l'architecture fonctionnelle du système.

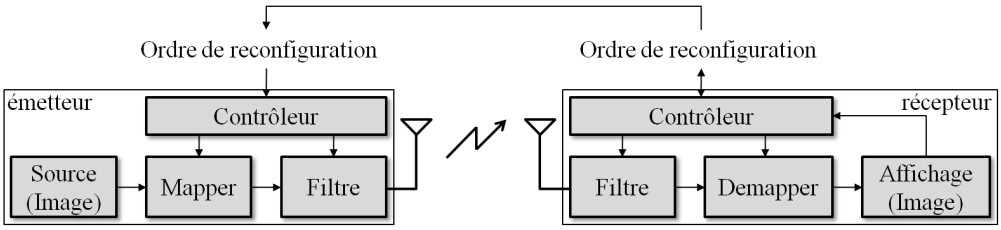


Figure 88. Synoptique du système de transmission sans-fil RLR simplifié

Les fonctions de filtrage et de Mapping/demapping sont configurables. Pour gérer cette reconfiguration les modules de contrôle intègrent une instance de l'architecture HDCRAM adaptée au système en question.

Contrôleur de l'émetteur

Ce module permet de gérer la reconfiguration des modules de la chaîne de TNS (*Mapper* et/ou du *Filtre*) de l'émetteur. L'architecture de ce module est une instance de l'architecture de gestion de reconfiguration HDCRAM. Cette instance est relativement simple au vu de la simplicité de la chaîne de TNS. Deux unités de reconfiguration de niveau 3, L3_ReMU, sont respectivement associées au *Mapper (L3_ReMU_Mapper)* et au *Filtre (L3_ReMU_Filter)*. Le système étant simple, une seule unité de niveau 2, L2_ReMU, est nécessaire. Enfin, quelle que soit la complexité du système, une seule et unique unité de niveau 1, L1_ReM, est instanciée.

Contrôleur du récepteur

L'architecture de ce module diffère de la version mise en œuvre pour l'émetteur. En effet, une instance de la branche cognitive de l'architecture HDCRAM est mise en œuvre : une unité de niveau 3, L3_CRMU, est associée au module *Affichage* (*L3_CRMU_TEB*). Celle-ci récupère une information de TEB (*Taux d'Erreur Binaire*), capturée par le module d'affichage (*Affichage*), et l'analyse. En fonction du niveau de TEB, l'unité *L3_CRMU_TEB* prend la décision de reconfigurer ou non le système. Cette information est remontée au niveau 2, L2_CRMU, puis au niveau 1, L1_CRM qui transmet l'ordre de reconfiguration à la branche descendante de l'architecture (ReM), qui gère la reconfiguration, du récepteur. Le L1_CRM transmet également l'ordre à son homologue de l'émetteur, qui la répercute à la branche descendante de l'architecture (ReM) qui gère la reconfiguration du l'émetteur.

Nous rappelons également que la plate-forme matérielle supportant l'application du traitement numérique du signal est une carte de prototypage ML506 de Xilinx [30] composée principalement d'un FPGA Virtex-5 supportant la technologie dite de reconfiguration dynamique partielle.

5.2.3.2 Une modélisation UML/MARTE du système RLR

La Figure 89 illustre un extrait du diagramme de classe du PIM du système RLR au niveau de modélisation AML. Il s'agit donc du modèle fonctionnel du système. Cette vue illustre principalement le diagramme de classe de l'émetteur. Le récepteur est représenté par une « boîte noire ». Comme indiqué précédemment, on retrouve l'instance de l'architecture HDCRAM dédiée à l'émetteur.

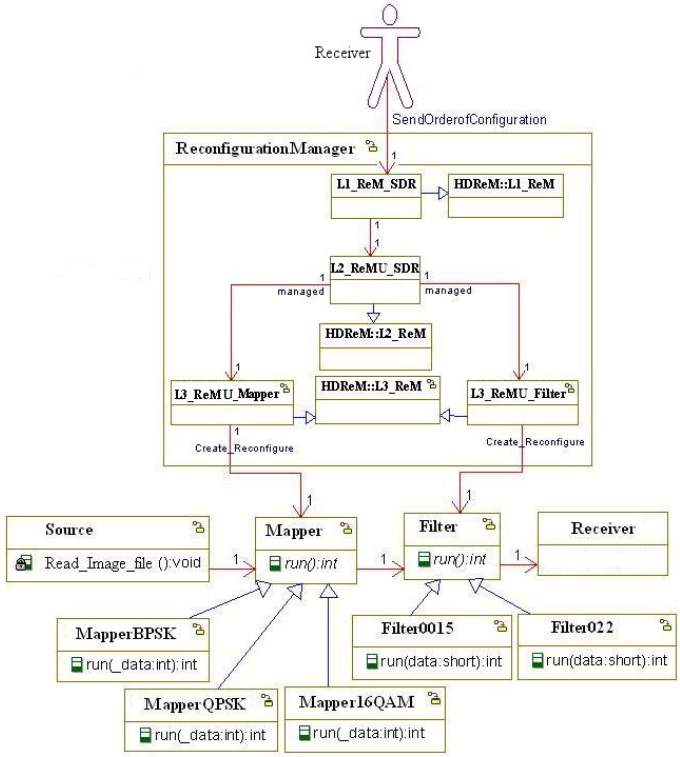


Figure 89. Extrait du diagramme de classe du système RLR intégrant un gestionnaire de reconfiguration.

Toutes les informations, tels que les attributs et les méthodes des différentes classes de cette vue, ne sont volontairement pas apparentes, ceci dans un souci de lisibilité du diagramme. En effet, un des points délicats de l'utilisation des diagrammes en UML est leur utilisation à bon escient : choisir le bon diagramme pour répondre aux besoins et trouver un compromis sur le nombre d'informations à faire apparaître sur celui-ci. Un diagramme contenant trop d'informations deviendra très vite illisible, de même qu'un diagramme ne comportant pas suffisamment d'informations le rendrait inutile.

La Figure 90 illustre l'allocation des fonctions du PIM sur le PM.

Note : Nous rappelons ici qu'au niveau AML de la méthodologie MOPCOM, le modèle de la plate-forme est abstrait et se caractérise par des éléments d'exécution communiquant point-à-point. Cette communication respecte la sémantique de MoCs. Dans le cas de notre système RLR, l'application étant peu complexe, le modèle PM du niveau AML l'est également.

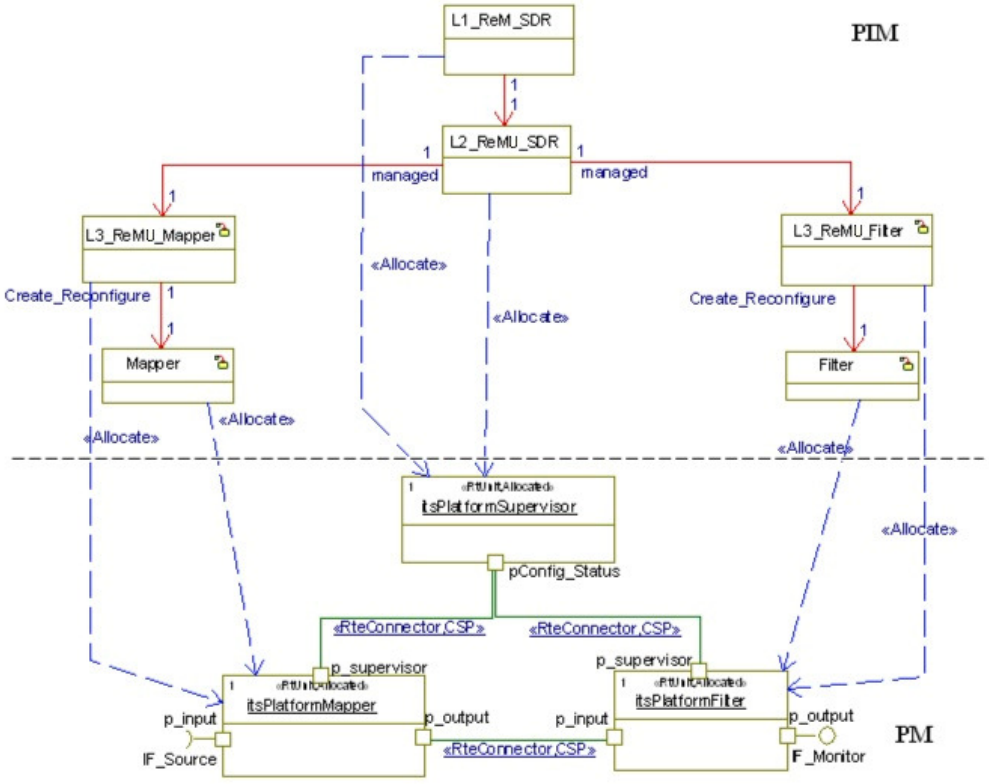


Figure 90. Extrait du diagramme d'allocation du PIM sur le PM du niveau AML du système RLR.

La Figure 90 met en évidence un extrait du PM, à savoir le PM correspondant à l'exécution des fonctionnalités de l'émission. La communication entre les composants d'exécution, stéréotypés «RtUnit» (issu du profile MARTE), s'effectue en respectant le MoC CSP. Ce choix n'est pas anodin sur l'exécution du système. En effet, le comportement des MoCs peut influencer le fonctionnement global du système. Il est donc impératif de bien choisir le type de MoC entre chaque composant. Ceci étant dit, encore une fois, notre système étant simple, le choix n'a pas posé de problème.

Pour bien comprendre la vue présentée, nous pouvons préciser que la classe *Mapper* n'est pas directement allouée sur le composant d'exécution *itsPlatformMapper*. En effet, selon le contexte, il s'agit soit d'une instance des classes enfants *MapperBPSK*, *MapperQPSK* ou *Mapper16QAM* qui sera concrètement allouée sur le composant d'exécution *itsPlatformMapper*. L'allocation est spécifiée à l'aide du stéréotype « Allocate » de MARTE, et à l'aide du tag associé « nature » nous précisions qu'il s'agit d'une allocation temporelle (*TimeScheduling*), donc limitée dans le temps, ou spatiale.

Au niveau EML, le caractère reconfigurable d'un système n'engendre pas de nouvelles contraintes de modélisation par rapport à un système non-reconfigurable. Un certain nombre de contraintes temporelles vont être ajoutées au modèle, notamment des estimations de temps de reconfiguration afin d'observer l'impact de celles-ci sur le système, et ainsi valider ou non la topologie de la plate-forme matérielle au regard des contraintes temporelles. Toutefois, ceci peut être tout à fait modélisé à l'aide du langage UML et surtout du profile MARTE. Nous n'avons donc pas vu la nécessité de surcharger la méthodologie MOPCOM à ce niveau.

En revanche, et tout particulièrement dans le cas où la plate-forme matérielle est basée sur un composant de type FPGA (comme dans notre système de transmission RLR), la modélisation du système au niveau DML nécessite de prendre en compte ces caractéristiques. Pour modéliser la carte de prototypage ML506 de Xilinx au niveau DML nous avons utilisé le méta-modèle exposé dans la section 5.2.2.2. La Figure 91 représente le diagramme de composition illustrant l'architecture implantée dans le FPGA. Dans cette vue, les éléments nécessaires à la mise en œuvre de la reconfiguration dynamique partielle apparaissent clairement dans le modèle. Ainsi, nous trouvons un processeur logiciel (MicroBlaze) associé à ses éléments de communication, une RRP et un composant HwICAP. La zone fixe est constituée de l'ensemble des éléments que compose ce modèle sauf la RRP intitulée *ReconfigurableArea* sur la Figure 91.

La Figure 92 illustre un extrait de l'allocation du PIM sur le PM au niveau DML, allocation de la chaîne de TNS de l'émetteur sur l'architecture du FPGA. Pour information, le module *Mapper* (*itsMapper*) est alloué sur le processeur (*itsMicroBlaze*), et le module *Filtre* (*itsFilter*) sur la RRP (*ReconfigurableArea*). De plus, l'instance de l'architecture HDCRAM sera déployée sur le processeur.

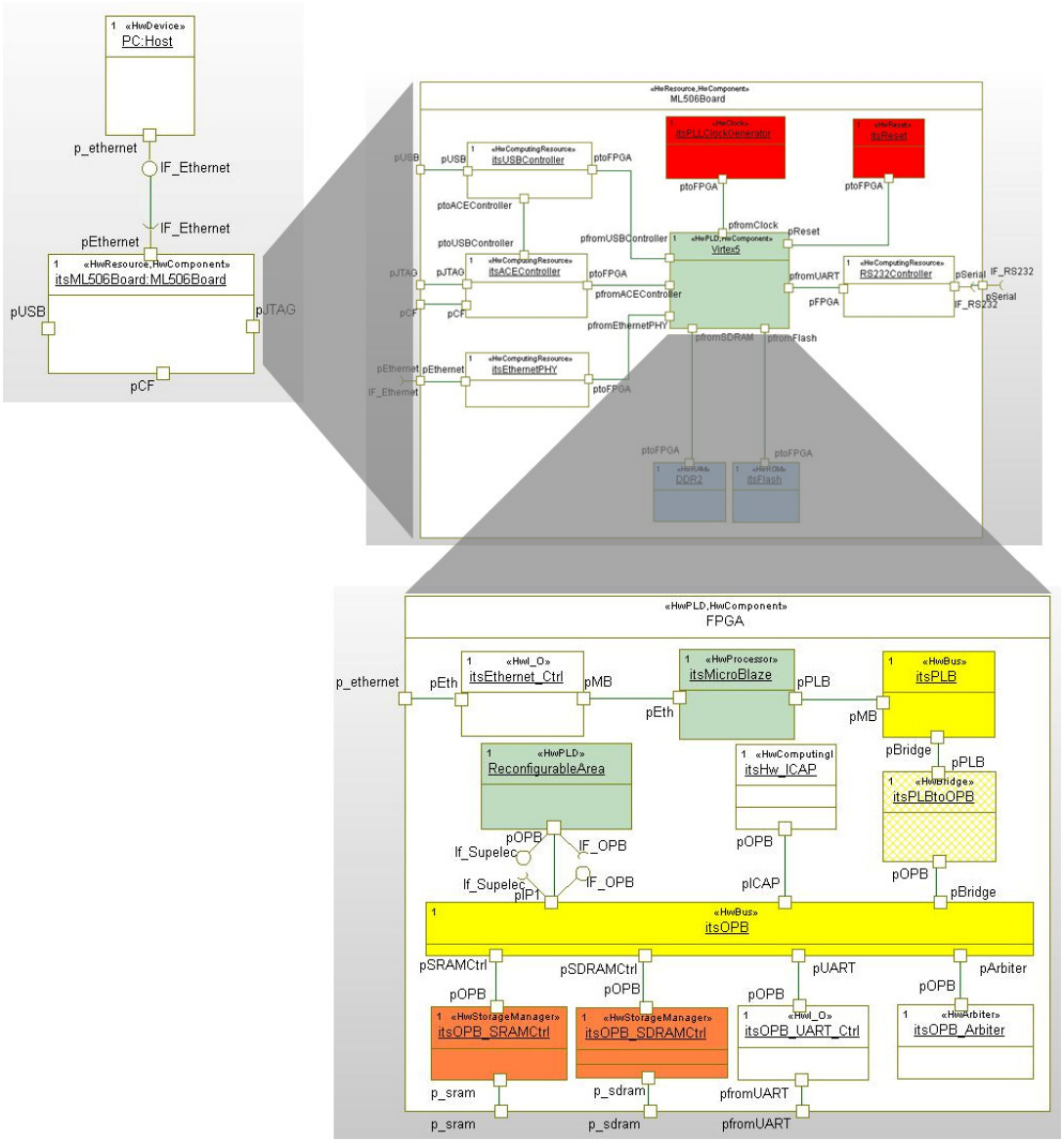


Figure 91. Extrait d'un modèle PM d'une plate-forme matérielle reconfigurable basée sur un FPGA.

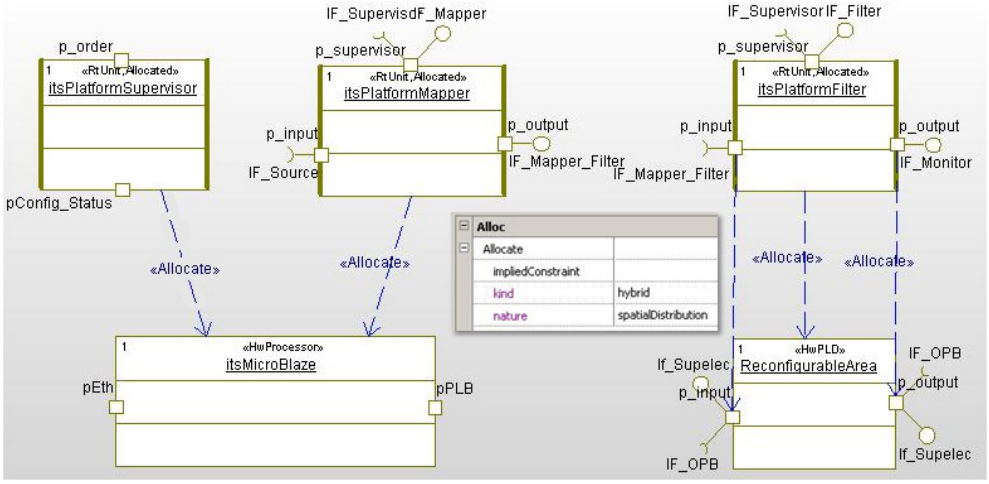


Figure 92. Extrait de l'allocation du PIM sur le PM du niveau DML du système RLR.

5.2.4 Des limitations

Nous venons d'exposer et d'illustrer, au travers d'un système RLR simple, la méthodologie MOPCOM étendue pour des systèmes reconfigurables (reconfiguration logicielle et/ou matérielle).

Toutefois, dans l'état actuel des choses, la modélisation UML ne permet pas de valider complètement l'extension de la méthodologie MOPCOM pour des systèmes reconfigurables. Il faut pouvoir valider le caractère dynamique du système embarqué, l'ordonnancement des tâches et valider les performances temporelles du système afin d'observer et de valider l'impact de la reconfiguration sur le fonctionnement global du système embarqué. Comme nous l'avons déjà indiqué, le langage UML n'est pas nativement exécutable. Alors, comment vérifier que le choix architectural soit conforme au cahier des charges du système embarqué, et que la mise en œuvre de la reconfiguration respecte les contraintes temps réel ?

Certains outils de modélisation UML, tel que Rational Rhapsody d'IBM [67], offrent la possibilité de simuler les modèles UML. Cependant, dans le cas de Rhapsody, le moteur d'exécution supportant cette simulation surcharge le langage UML et interprète la norme. Une des solutions qui semble la plus aboutie pour répondre à ce problème est le passage du monde UML vers le monde SystemC. C'est ce que nous proposons de réaliser dans la prochaine section.

5.3 Une simulation SystemC

Nous avons opté pour le « langage » de modélisation SystemC pour rendre exécutable le modèle d'un système embarqué reconfigurable. Notre solution s'inspire des travaux sur les méthodologies de l'IDM, qui proposent de coupler la modélisation UML à la modélisation SystemC. De notre point de vue, ces deux langages sont complémentaires. Nous expliquons dans cette section la raison de notre choix et la manière dont nous avons exploité la modélisation SystemC pour la conception de systèmes embarqués reconfigurables.

5.3.1 La librairie SystemC

Comme nous l'avons déjà indiqué dans ce mémoire, l'utilisation d'une multitude de langages tout au long de la phase de développement d'un système embarqué est un frein majeur à l'amélioration de la productivité. Pour remédier à ce problème et réduire les coûts de développement, SystemC [141][142] est apparu dans les années 90, sous l'impulsion de la société Synopsis. SystemC est aujourd'hui supporté par plusieurs industriels regroupés au sein de l'Open SystemC Initiative (OSCI), chargée du développement et de la diffusion de cette librairie. Aujourd'hui la librairie SystemC est standardisée par l'IEEE sous la norme IEEE 1666™-2005 [106] depuis 2005.

5.3.1.1 Une architecture

SystemC est souvent considéré comme un langage à part entière bien qu'il s'agisse d'une extension du langage orienté objet C++. Basé sur les concepts de ce dernier, SystemC ajoute les éléments nécessaires à la description des éléments matériels. Il permet également une exécution à la fois séquentielle et concurrente du modèle, ce qui le différencie des langages de programmation tels que le C++ et le Java. A ce titre, il est souvent considéré comme un langage de description matérielle comme le VHDL ou le Verilog. Toutefois, SystemC n'est ni un langage de programmation objet ni un langage de description matérielle, mais une librairie du C++ qui permet de concilier dans un même formalisme modélisation logicielle orientée objet et modélisation matérielle. De plus, contrairement aux langages de modélisation, tel que l'UML, le SystemC dispose d'un noyau de simulation événementiel. Ainsi, SystemC s'intègre parfaitement dans une méthodologie de conception ESL mais n'est en aucun cas une méthodologie. L'architecture de la librairie SystemC, illustrée par la Figure 93, est représentée sous forme de couches :

- La couche basse représente le socle du SystemC : le langage C++ ;
- La 2nde couche spécifie le moteur de simulation du SystemC, les éléments de base de la librairie, ainsi que les types de données disponibles (l'ingénieur logiciel sera familiarisé avec les types issus du langage C++, et l'ingénieur matériel sera familiarisé avec les types issus des langages de description matérielle) ;
- La 3ème couche met en avant le caractère transactionnel de la modélisation SystemC, où des canaux de bases sont disponibles aux concepteurs, pouvant être utilisés pour modéliser des canaux de communications plus complexes ;
- La couche supérieure fait apparaître des extensions du SystemC dédiés à des domaines spécifiques.

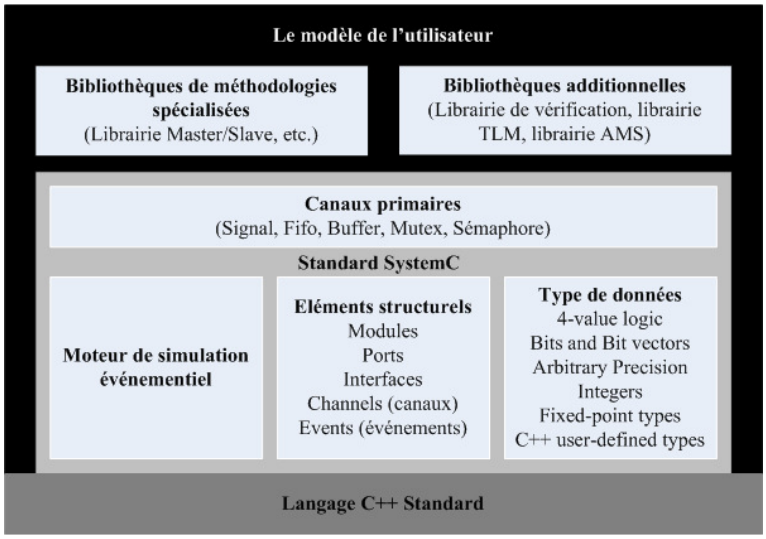


Figure 93. Architecture en couche de la librairie SystemC [142].

5.3.1.2 Les niveaux d'abstraction

Afin de se démarquer d'une description RTL (VDHL ou Vérilog), la librairie SystemC propose, depuis sa seconde version, trois niveaux de modélisations basés sur la notion de modélisation TLM (*Transaction Level Modelin*g) (cf. Tableau 14). Le niveau le plus abstrait, PV (*Programmers View*), fait totalement abstraction de toutes contraintes temporelles, alors que le niveau CC (*Cycle Callabe*) permet une modélisation au cycle près, modèle proche d'une description RTL.

	Niveau d'abstraction	Caractéristiques	Temps de simulation
	System Algorithm Model (SAM)		-
TLM	Programmers View (PV)	Modélisation fonctionnelle	x100
TLM	PV-Timed (PV-T)	Exécution et communication timés	x10
TLM	Cycle Callabe (CC)	Exécution au cycle près	x1~x2
	RTL		x1 (référence)

Tableau 14. Les différents niveaux d'abstraction du SystemC (selon l'OSCI).

Note : Il faut être prudent sur les termes employés pour désigner un niveau de modélisation TLM car tous acteurs du SystemC n'utilisent pas les mêmes.

5.3.1.3 Pourquoi choisir le SystemC ?

Pour simuler le modèle d'un système reconfigurable notre choix s'est porté sur le SystemC car il permet :

- De modéliser la partie logicielle et matérielle d'un système embarqué avec un même formalisme ;
- D'exécuter de manière séquentielle mais aussi d'émuler la concurrence des tâches, ce qui n'est pas le cas des langages de programmation orientés objet tel que le C++ ou le Java ;
- De modéliser à différents niveaux d'abstraction selon les besoins ;
- De réaliser des modèles dont la portabilité, la réutilisation, et la maintenance est accrue car la librairie SystemC est standardisée.

De plus, le positionnement du SystemC (cf. Figure 94) par rapport aux différents langages utilisés dans les méthodologies de co-conception actuelles de SoC (ou SoPC) en fait un bon candidat pour son utilisation dans la méthodologie de co-conception ESL.

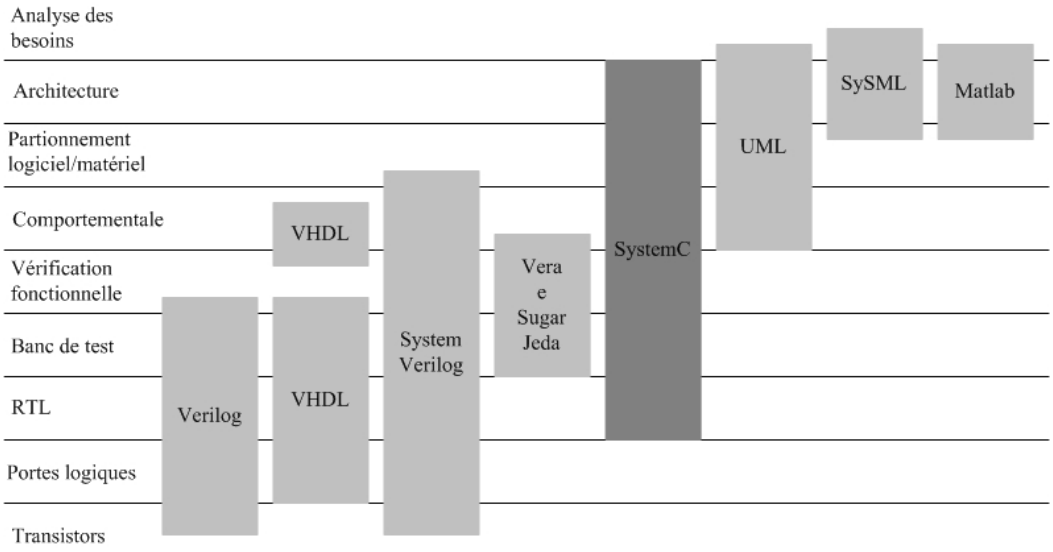


Figure 94. Positionnement du SystemC par rapport à d'autres langages utilisés dans un flot de conception de SoC [142].

Au vu de ses points forts, SystemC semble le « langage » de modélisation conjoint logiciel/matériel par excellence. La question suivante peut alors être posée :

Pourquoi ne pas avoir d'emblée développé une méthodologie de co-conception pour les systèmes embarqués basée sur le SystemC plutôt que sur le langage UML ?

Tout d'abord parce que l'UML permet de modéliser un système de manière plus abstrait que le SystemC. Ensuite, le SystemC ne dispose pas de représentation graphique normalisé comme c'est le cas de l'UML. Ce dernier a l'avantage, grâce à sa représentation graphique, de permettre une meilleure compréhension réciproque et donc une meilleure interaction entre les différents corps de métiers (ingénieur système, ingénieur logiciel, ingénieur matériel, etc.). Une représentation graphique des modèles SystemC existe [143], toutefois celle-ci n'est pas standardisée. De plus, le SystemC est aujourd'hui principalement utilisé par les équipes de conception matérielle pour la modélisation de la plate-forme matérielle d'un système embarqué et de l'ensemble du système embarqué, mais peu utilisé par les équipes de conception logicielle. Enfin, nous ajoutons que l'utilisation du SystemC est à ce jour encore marginale par rapport à l'utilisation de l'UML.

5.3.2 Modélisation d'un système reconfigurable en SystemC

La modélisation d'un système embarqué reconfigurable en SystemC doit bien évidemment reprendre les mêmes éléments qu'une modélisation UML. Idéalement le modèle SystemC doit être équivalent au modèle UML. Pour obtenir un tel résultat, la meilleure solution serait une génération de code automatique depuis le modèle UML vers un modèle SystemC. Dans l'état actuel de l'environnement de développement MOPCOM, nous ne disposons pas d'un tel outil en standard. De plus, le développement d'un tel générateur de code est en dehors des objectifs de cette thèse. Toutefois, de tels générateurs existent, mais le plus souvent ils sont propriétaires.

Nous pouvons notamment citer :

- Le profil UML for SystemC [61] permet de faire apparaître les éléments du SystemC dans un modèle UML et ainsi de générer un modèle SystemC depuis un modèle UML ;
- Le générateur de code UML vers SystemC développé dans le cadre du projet MARTES [87] ;

Aujourd'hui, il existe encore peu de générateur UML vers SystemC intégré dans les outils de conception ESL du marché. Toutefois, l'outil ESL Cofluent Studio [84], de la société Cofluent Design, accepte en entrée un modèle UML qui peut être indirectement transformé en un modèle SystemC. Cette transformation s'appuie sur l'outil MDWorkbench [107] de Sodius, utilisé dans l'environnement de conception MOPCOM. Comme perspective à ces travaux, il serait intéressant d'étudier la faisabilité d'intégrer le jeu de règles de la transformation UML vers SystemC de CoFluent Studio dans l'environnement d'outillage MOPCOM.

Dans nos travaux, nous nous sommes concentrés sur un développement manuel d'une description SystemC d'un système reconfigurable. De plus, nous nous sommes focalisés sur le niveau de modélisation AML de la méthodologie MOPCOM. En effet, l'architecture HDCRAM est actuellement dans l'état actuel uniquement spécifiée à ce niveau de modélisation.

Note : Le développement de l'architecture HDCRAM à des niveaux de modélisation plus fin n'était pas prévu dans mes travaux de thèse. Toutefois, l'équipe SCEE a pour objectifs de poursuivre des travaux en ce sens et ainsi d'exploiter cette architecture HDCRAM à différents niveaux de modélisation afin de parcourir l'ensemble des trois niveaux de modélisation proposés par MOPCOM.

Nous avons donc procédé à une transcription manuelle en SystemC du méta-modèle HDCRAM. Ce passage en SystemC de l'architecture nous a amené à apporter des modifications afin de respecter la sémantique de la librairie SystemC, sémantique ayant des limitations pour la modélisation de systèmes reconfigurables.

5.3.2.1 Des limitations

Comme indiqué dans la section 5.2.2.1, le concept de polymorphisme (illustré à gauche de la Figure 95) et d'héritage sont utilisés pour modéliser une tâche ayant différentes mises en œuvre possibles suivant le contexte dans lequel se trouve le système. Ces concepts sont présents à la fois dans le langage de modélisation UML et dans le langage de programmation C++, tous deux basés sur les concepts orientés objet. Par conséquent, lors de l'instanciation du système, une seule des classes enfants sera instanciée (objet). Lorsque l'état du système évolue de manière à ce que le mode opératoire de la tâche reconfigurable doit être changé, alors une destruction de l'objet est effectuée et la création d'un nouvel objet le remplaçant est réalisée (instance du nouveau mode opératoire). Ces opérations se réalisent durant la phase d'exécution du programme. Ce mécanisme reflète alors le fonctionnement d'un système réel sur sa plate-forme cible.

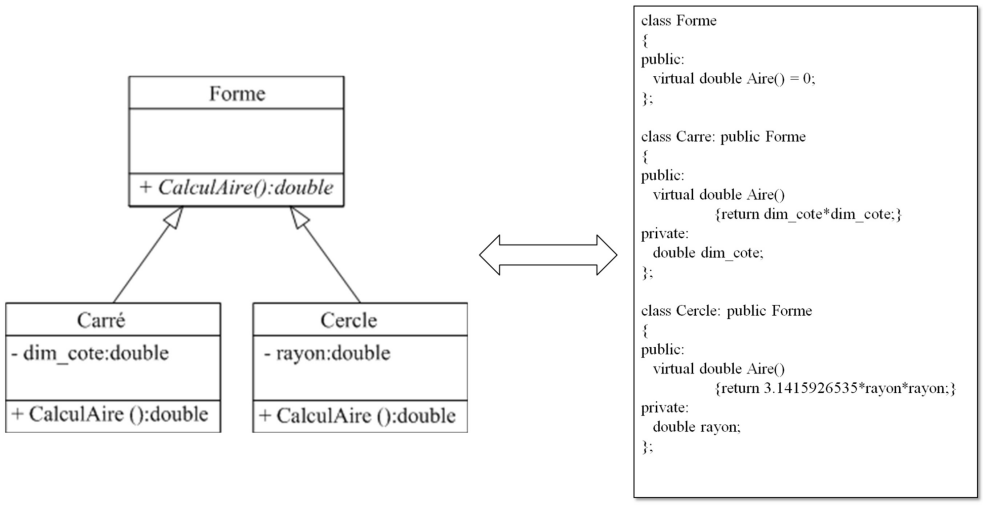


Figure 95. Représentation du concept de polymorphisme en UML et en C++.

Le code, à droite de la Figure 95, illustre l'équivalence en C++ du diagramme UML à gauche de cette même figure.

La librairie SystemC étant basée sur le langage C++, le développeur lambda peut imaginer que ce même mécanisme puisse être mis en œuvre dans un modèle SystemC. Cependant, la sémantique de la librairie SystemC ne permet pas de détruire et de créer des éléments du modèle durant la phase de simulation. Une fois le modèle SystemC déployé (phase qui précède la phase de simulation) l'architecture du modèle est figée. Cette contrainte est imposée par la modélisation de l'architecture matérielle. En effet, lors de des spécifications initiales, du SystemC, la technologie ne permettait pas de reconfigurer une plate-forme matérielle dynamiquement. Depuis, les technologies ont évolué mais la librairie SystemC n'a malheureusement pas évoluée en conséquence.

5.3.2.2 Notre solution : un opérateur reconfigurable

Pour contourner ce problème, nous aurions pu nous appuyer sur les processus[16] dynamiques de type *sc_spawn* de la librairie SystemC [106]. Ces processus peuvent non seulement être créés et détruits durant la phase d'élaboration de l'architecture du système mais également durant la phase de simulation. Dans ce cas-ci, la notion d'héritage utilisée dans une modélisation UML disparaît. En UML, les différents modes opératoires d'une tâche sont représentés par une classe, chacune héritant d'une même classe abstraite parente (exemple de la tâche Mapper du système RLR, cf. Figure 90). Par contre en SystemC, ces modes opératoires seront représentés par des processus dynamiques créés et détruits dynamiquement durant la phase de simulation. Ces processus appartiennent au module[17] qui représente la tâche. En d'autres termes, en reprenant l'exemple de la tâche *Mapper* du système RLR (cf. Figure 91), seule la classe parente *Mapper* sera présente dans la modélisation SystemC, les classes enfants *MapperBPSK*, *MapperQPSK* et *Mapper16QAM* disparaissent et sont intégrées dans la classe parente *Mapper* sous la forme de processus dynamiques. Il va de soi que la classe *Mapper* n'est plus abstraite et

[16] Processus du SystemC ≈ méthode en UML ≈ fonction en C ; Process du SystemC ≈ Process du VHDL dans le sens où ils sont concurrents entre eux.
[17] Module du SystemC ≈ classe en UML/C++.

peut donc être instanciée dans ce cas-là, au contraire de la représentation UML dans laquelle celle-ci est totalement virtuelle et donc non-instanciable. Toutefois, bien que cette solution convienne pour modéliser et simuler un système embarqué reconfigurable à un haut niveau d'abstraction nous n'avons pas opté pour cette solution : celle-ci nous semble trop éloignée de la modélisation UML présentée précédemment (cf. 5.2.3).

Nous avons opté pour une solution dans laquelle le modèle SystemC fait apparaître plus clairement le caractère reconfigurable d'un opérateur. La solution choisie, contrairement à la solution précédemment exposée, fait toujours apparaître les modules représentant les différents modes opératoires d'une tâche. Nous rappelons que l'instance d'une classe/module SystemC peut uniquement être créée lors de la phase d'élaboration de l'architecture. Par conséquent, tous les modes opératoires, rattachés à une même tâche primitive, sont encapsulés et instanciés dans un module intitulé *OpérateurReconfig* pour opérateur reconfigurable (cf. Figure 96).

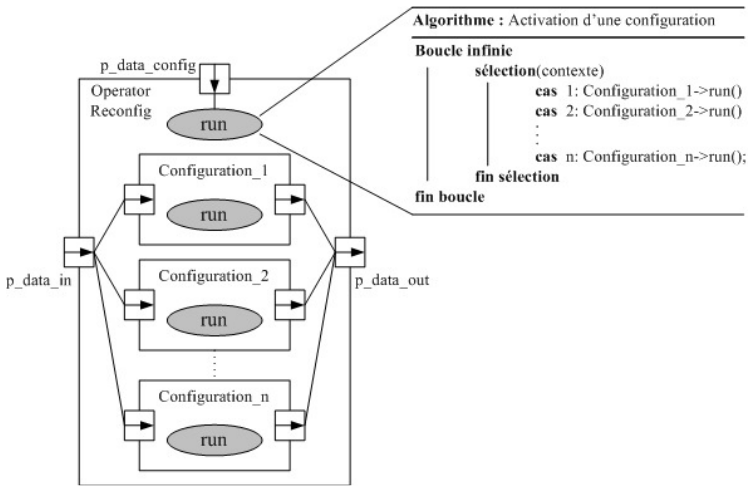


Figure 96. Représentation graphique de l'opérateur reconfigurable modélisé en SystemC.

Cet opérateur active le mode opératoire correspondant au contexte dans lequel se trouve le système. Celui-ci ne contient pas d'intelligence, par conséquent, cet opérateur ne fait pas lui-même le choix du mode opératoire à activer, ce choix lui étant indiqué par son niveau L3_ReMU (niveau le plus bas de la branche ReM d'HDCRAM) associé lors de l'envoi d'un ordre de reconfiguration. Le module *OperatorReconfig* est constitué d'un processus principal, nommé *run*, actif en permanence : celui-ci est de type sc_*thread*[18]. Ce processus a pour rôle d'activer le mode opératoire en adéquation avec l'ordre de reconfiguration et ainsi activer la méthode principale de celui-ci (les autres modes opératoires étant quant à eux inactifs). Par rapport à un opérateur non-reconfigurable (en se référant au méta-modèle HDCRAM), l'opérateur reconfigurable comportera obligatoirement un port de communication supplémentaire. Ce port nommée *p_data_config* permet de recevoir l'ordre de reconfiguration depuis le niveau L3_ReMU associé. L'ordre de reconfiguration étant associé à l'identifiant du mode opératoire à activer. Ce port est relié directement au processus *run* de l'opérateur reconfigurable.

[18] deux types de processus en SystemC existent : le *sc_method*, activé uniquement si un événement survient sur l'un des éléments de sa liste de sensibilité associée (similitude avec le process en VHDL) ou le *sc_thread*, activé en permanence.

Celui-ci est en permanence scruté afin de capturer à tout instant un nouvel ordre de reconfiguration.

Ci-dessous nous donnons le code SystemC du module *OperatorReconfig* et un exemple d'instanciation du module *Demapper* du système RLR présenté dans le chapitre 1 (cf. Figure 88).

```
#ifndef _OPERATOR_RECONFIG_H
#define _OPERATOR_RECONFIG_H

#include <systemc.h>
#include « csp_chan.h » // canal de communication de type point à point bloquant
#include «Operator.h »    //  l'opérateur reconfigurable hérite de la super classe Opérateur définit dans le méta-modèle HDCRAM

// déclaration du module OperatorReconfig
Template <typename Tconfig>
class OperatorReconfig : public Operator
{
Public :
        // déclaration du port supplémentaire
        sc_port<csp_in_if<Tconfig> > p_data_config ;

        // déclaration du constructeur
        SC_HAS_PROCESS(OperatorReconfig) ;
        OperatorReconfig(sc_module_name _name, const int _config) :
        Operator(_name),
        current_activate_config(_config)
        {
                // processus de type thread sans liste de sensibilité
                SC_THREAD(run) ;
        }

        // déclaration du destructeur
        ~OperatorReconfig() ;

Private :
        // attributes
        int current_activate_config;

        Operator *Configuration_1;
        Operator *Configuration_2;
        .
        .
        .
        Operator *Configuration_n ;

}
#endif
```

Note : Tout comme la classe *Operator* du méta-modèle HDCRAM la classe *OperatorReconfig* est virtuelle et ne peut donc pas être instanciée.

```
#ifndef _DEMAPPER_H
#define _DEMAPPER_H

#include <systemc.h>
#include «OperatorReconfig.h »
#include « DemapperBPSK.h »
#include « DemapperQPSK.h »
#include « Demapper16QAM.h »

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// déclaration du module OperatorReconfig
// le module Demapper est un opérateur reconfigurable, il hérite donc des caractéristique de ce dernier
class Demapper : public OperatorReconfig <int>
{
Public :
        // déclaration du constructeur
        SC_HAS_PROCESS(Demapper) ;
        Demapper(sc_module_name _name, const int _config) :
        Operator(_name),
        current_activate_config(_config)
        {
                // processus de type thread sans liste de sensibilité
        SC_THREAD(run) ;
        }

        // déclaration du destructeur
        ~Demapper() ;

Private :
        //déclaration de pointeurs
        DemapperBPSK        * demabpsk ;    // mode opératoire 1
        DemapperQPSK        * demaqpsk ;    // mode opératoire 2
        Demapper16QAM       * dema16qam ;   // mode opératoire 3

        // method
        void run() ;
        void initialize() ;
}
#endif
```

La déclaration du module *Demapper* fait apparaître trois pointeurs dans la zone de déclaration privée de celui-ci. Ces trois pointeurs pointent vers les objets, une fois créés, correspondant aux trois modes opératoires que peut prendre le *Demappper*.

5.3.3 Simulation d'un système de radio logicielle restreinte

La modélisation SystemC du système RLR (cf. Figure 88), présentée dans la section 5.2.3.1 de ce chapitre, nous permet de valider la proposition précédemment présentée. Afin de donner plus de réalisme à cette preuve de concept, nous avons mis en œuvre, sous la forme d'un démonstrateur, une transmission sans-fil réelle. La chaîne de traitement du signal et la gestion de reconfiguration sont simulées par le modèle SystemC et la transmission RF (et le traitement associé) est assurée par une carte dédiée. Cela permet en outre de valider une première ébauche de la conception haut niveau dans un contexte pseudo-réaliste. Cette étape peut être effectuée en avance de phase par rapport à la l'intégration finale du système, ce qui permet de valider notamment l'architecture fonctionnelle du système mais aussi de valider la gestion de la reconfiguration à haut niveau.

Les spécifications d'un système RLR …

Les spécifications détaillées du système RLR utilisé comme preuve de concept sont présentés dans la section du chapitre 1. La Figure 88 rappelle le synoptique de la chaîne de traitement numérique du signal de ce système. Nous tenons à rappeler que ce système se caractérise par des opérateurs reconfigurables. Une modélisation UML, réalisée avec la méthodologie MOPCOM étendue, est présentée dans la section 5.2.3 de ce chapitre.

… en passant par une modélisation SystemC…

Nous rappelons que le modèle SystèmC du système RLR est réalisée manuellement. La modélisation SystemC de la chaîne de traitement du signal et de la gestion de la reconfiguration du système RLR correspond au modèle AML de ce même système. Les contraintes temporelles du système ne sont ici pas modélisées.

Dans un premier temps, seul l'émetteur du système RLR a été modélisé pour valider l'opérateur reconfigurable (*OperatorReconfig*). De plus, l'ordre de reconfiguration n'émane pas d'une décision de la branche cognitive de l'architecture HDCRAM (CRM) mais d'un utilisateur tiers. La Figure 97 illustre, sous forme d'une représentation graphique, la modélisation SystemC de l'émetteur. Sur cette figure ne sont pas représentés les modules *Source* et *Affichage*, le premier servant à la génération d'un flux binaire, et le second à l'affichage des données modulées puis mise en forme sous la forme d'une constellation IQ.

Le modèle SystemC reprend la même instance de l'architecture HDCRAM que la modélisation UML du système (cf. Figure 89). La différence apparait dans la manière de modéliser les opérateurs *Mapper* et *Filtre* et leurs modes opératoires associés. Dans notre modélisation SystemC ces deux opérateurs sont représentés par des modules héritant de la classe abstraite *OperatorReconfig* proposée précédemment. Ainsi, par exemple, les trois modes opératoires de l'opérateur *Mapper* font partie intégrant de ce dernier mais un seul de ces modes est actif à un instant donné en fonction de l'état du système. Le basculement d'un mode opératoire à un autre s'effectue lorsque l'opérateur reconfigurable *Mapper* reçoit un ordre de reconfiguration depuis son niveau L3_ReMU associé (*L3_ReMU_Mapper*).

Notons également que la communication entres les modules du modèle SystemC s'effectue par des canaux basés sur une sémantique de type CSP[19]. Nous retrouvons ainsi dans le modèle SystemC les éléments du PM du niveau AML (cf. Figure 90).

La Figure 98 illustre une vue UML équivalente au modèle SystemC réalisé de l'opérateur *Mapper*. Cette vue fait apparaitre clairement la différence entre le modèle UML non-exécutable (cf. Figure 89) et le modèle SystemC exécutable du système RLR (cf. Figure 97).

[19] CSP (*Communicating Sequential Processes*) : communication point-à-point par rendez-vous, communication bloquante à l'écriture et ainsi qu'à la lecture. Tant que le consommateur n'a pas lu l'information envoyée par l'émetteur ce dernier est bloqué, et inversement, tant que l'émetteur n'a pas envoyé de nouvelle information alors que le consommateur en attend une ce dernier est bloqué.

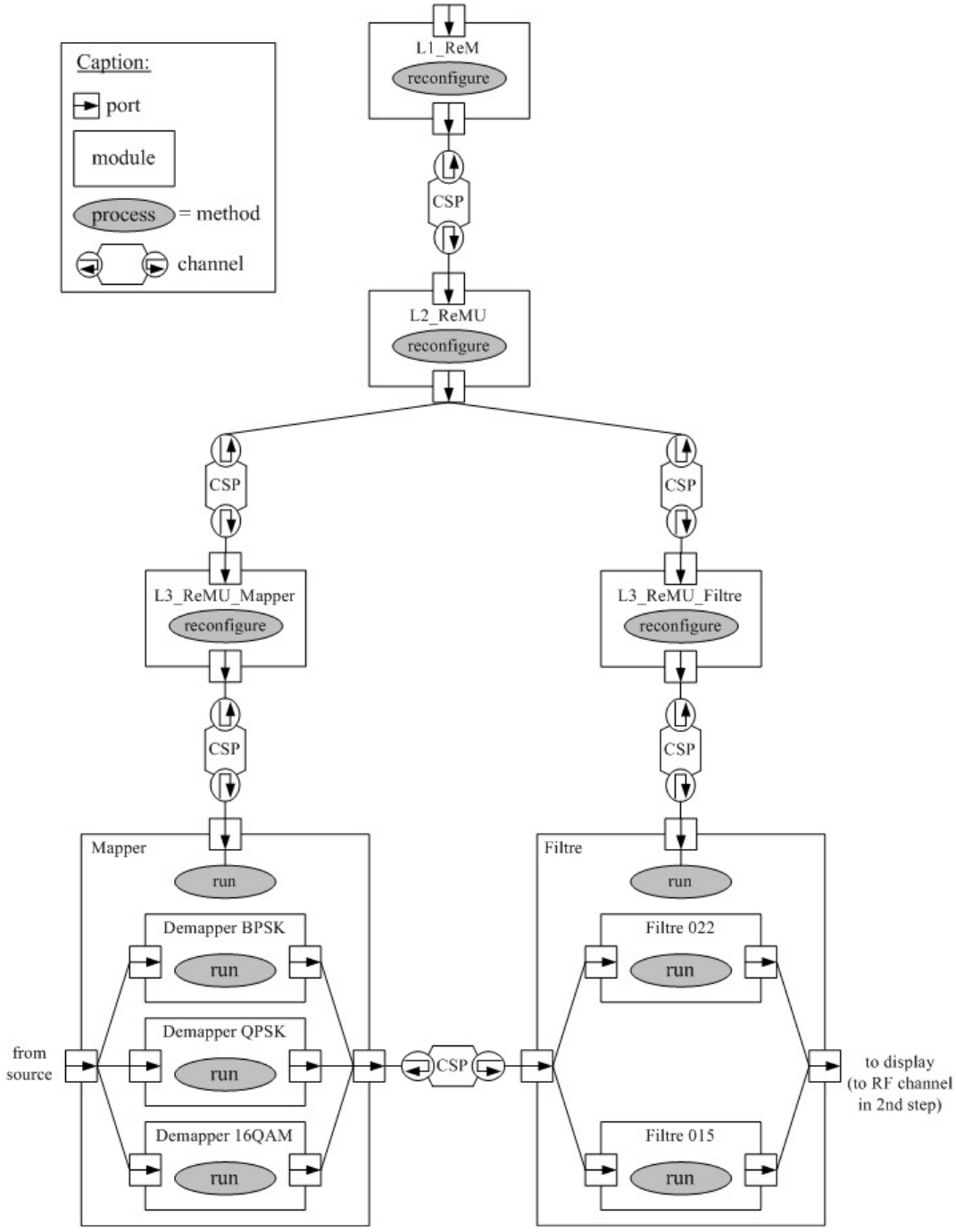


Figure 97. Extrait de la représentation graphique, non-standardisée, du modèle SystemC de l'émetteur du système RLR.

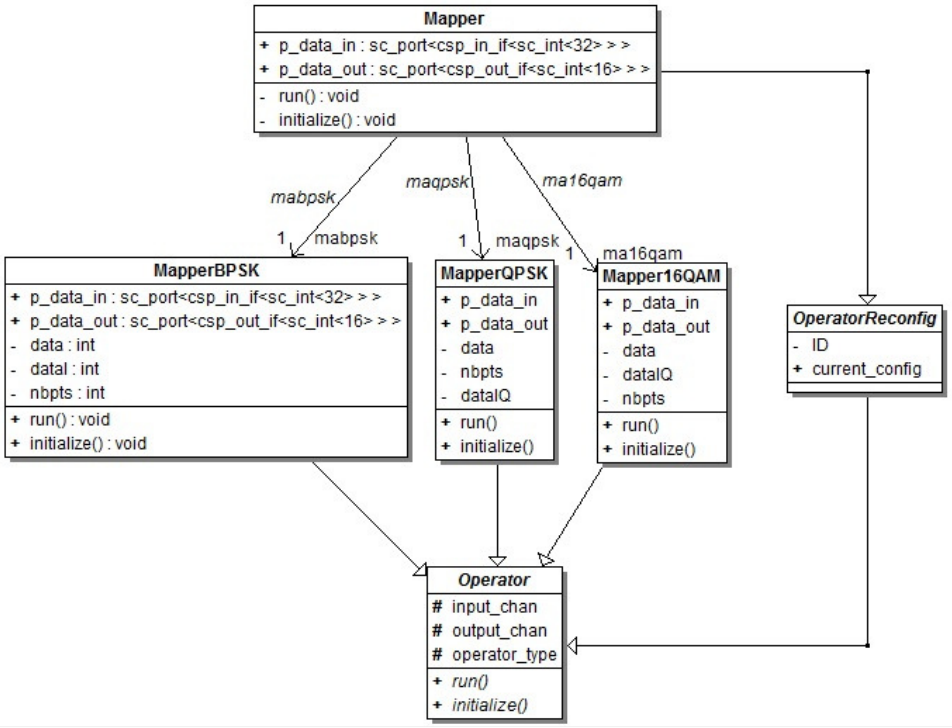


Figure 98. Extrait de la représentation graphique, basée sur le formalisme UML, du modèle SystemC du système RLR.

Les traces d'exécution du modèle SystemC de l'émetteur du système RLR, illustrées par la Figure 99, permettent de vérifier non seulement la fonctionnalité du modèle mais aussi de suivre l'acheminement de l'ordre de reconfiguration d'étage en étage de l'architecture HDCRAM instanciée (depuis le niveau L1 jusqu'à l'opérateur reconfigurable concerné).

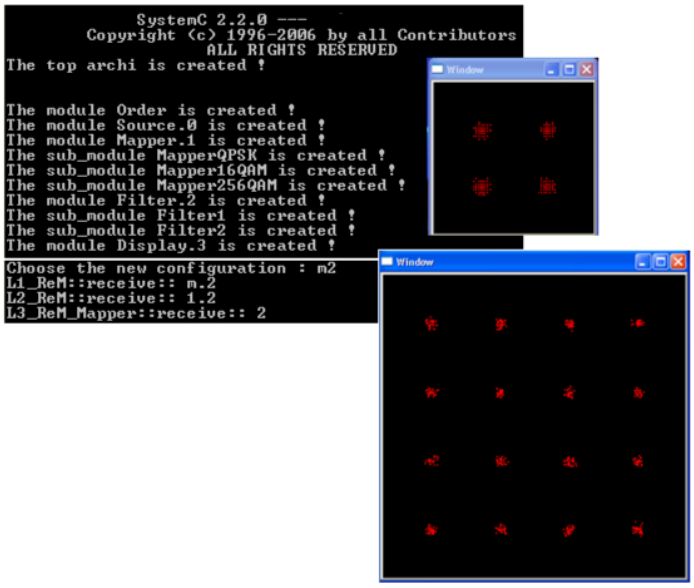


Figure 99. Traces visualisées lors de l'exécution du modèle SystemC de l'émetteur du système RLR.

Nous prouvons ainsi que notre manière de modéliser en SystemC un système reconfigurable est opérationnelle et viable. Afin de donner plus de consistance au résultat obtenu, une transmission réelle a été est mise en œuvre.

… pour aboutir à une transmission sans-fil réelle

Afin de mettre en évidence l'intérêt d'un modèle SystemC et de l'architecture HDCRAM, nous avons mis en œuvre une transmission sans-fil réelle sous la forme d'un démonstrateur RLR [144]. Cette démonstration met en œuvre le système RLR présenté dans le chapitre 1. La Figure 100 illustre le synoptique de cette démonstration.

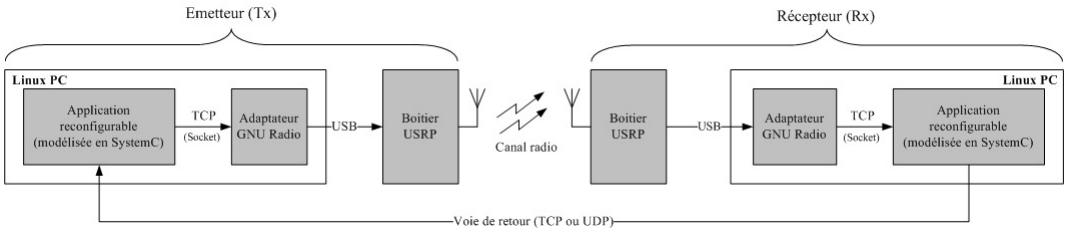


Figure 100. Synoptique du démonstrateur mettant en œuvre le système RLR présenté dans le chapitre 1.

La transmission sans-fil composé d'un émetteur (Tx) et d'un récepteur (Rx) est composée de trois sous-parties. La chaîne de traitement du signal bande de base de Tx et Rx (intitulé application reconfigurable sur la Figure 100) est reconfigurables et intègre l'architecture de gestion de la reconfiguration HDCRAM. Le scénario cognitif est le suivant :

- Un capteur, du côté du récepteur, évalue la qualité d'une métrique (TEB ou du SNR) ;
- Un processus de prise de décisions simple, basé sur des seuils, décide de reconfigurer ou non la modulation selon le niveau de la métrique mesurée ;
- Un processus de reconfiguration est activé, à la fois au niveau du récepteur et de l'émetteur, pour changer la constellation en conséquence.

Les chaînes de traitement du signal numérique de l'émetteur et du récepteur sont modélisées en SystemC non-timé (aucunes contraintes temporelles). La gestion du cycle cognitif est assurée par une instance de l'architecture HDCRAM, également modélisée en SystemC.

La transmission RF est assurée par des cartes USRP[20]. Un module GNU[21] radio permet d'adapter les informations échangées entre modèle SystemC et la carte URSP. A l'émission l'adaptateur GNU radio permet d'interpoler et d'amplifier le signal en sortie

[20] USRP (Universal Software Radio Peripheral) : carte, basée sur un FPGA, développée dans le cadre de la RL qui assure la conversion analogique/numérique ou numérique/analogique du signal. La transposition fréquentielle bande de base vers RF est assurée par une carte fille. La fréquence de transmission est comprise entre 2,3GHz et 2,9GHz.

[21] GNU radio: librairie de composant de traitement du signal développé en C++. L'outil GRC associé, fournit une interface graphique qui permet de déployer une chaîne de traitement du signal basé sur la libraire GNU.

du modèle SystemC. En réception, l'adaptateur GNU radio permet de filtrer le signal autour de la fréquence porteuse (filtre bande passante), et d'accrocher en phase et en symbole le signal reçu.L'adaptateur GNU radio est connecté au modèle SystemC au travers d'une liaison TCP (*Transmission Control Protocol*) émulée par un socket. De l'autre côté, l'environnement GNU radio communique avec la carte USRP via une liaison USB.

Pour un souci de simplicité, l'ordre de reconfiguration, décidé par le récepteur, est transmis à l'émetteur par une liaison TCP. Cette transmission aurait très bien pu être effectuée par voie radio.

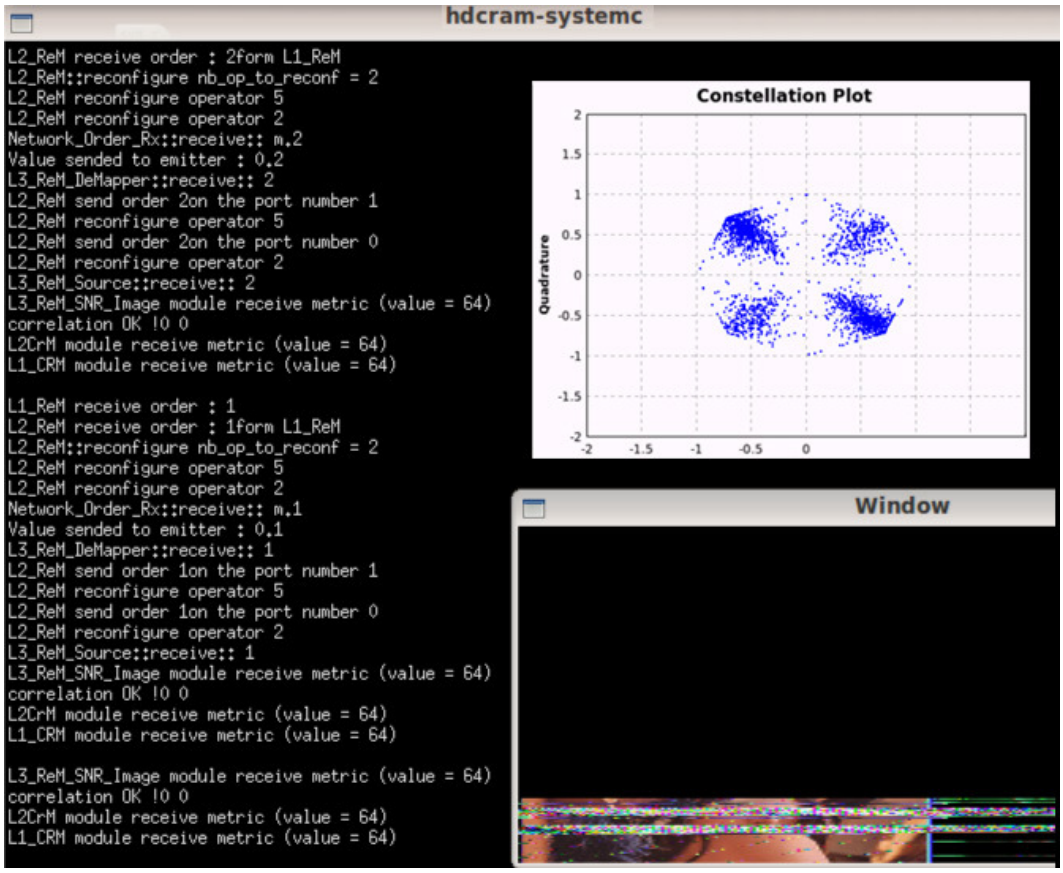


Figure 101. Illustration d'une image réceptionnée bruitée et d'une prise de décision de reconfiguration du système RLR.

Une image est transmise entre le récepteur et l'émetteur. Au cours de la démonstration, la transmission débute avec un fort niveau de SNR, et une modulation de type QPSK. Le niveau d'amplification du signal à l'émission (Tx) est diminué, et, le récepteur (Rx) détecte automatiquement le niveau de réception sous lequel la qualité de service n'est plus acceptable. Ce dernier envoie alors un ordre de changement de modulation à l'émetteur et se reconfigure lui-même afin que la transmission retrouve, de nouveau, une bonne qualité de service. Toute la chaîne de traitement du signal en bande de base et la gestion de la reconfiguration du système sont assurées par un modèle SystemC. La Figure 101 illustre la réception d'une image bruitée, ce qui engendre une reconfiguration du système. Les traces à gauche de la Figure 101 illustrent les messages de reconfiguration de l'architecture HDCRAM déployée.

5.4 Conclusion

Dans ce chapitre nous avons présenté une extension de la méthodologie MOPCOM pour modéliser les systèmes reconfigurables et tout particulièrement dans le contexte de la radio logicielle. Pour limiter l'overhead dû à la reconfiguration, une architecture de gestion de la reconfiguration intelligente est nécessaire. Afin de répondre à ce besoin, l'architecture HDCRAM, développée par l'équipe SCEE de Supélec, est intégrée dans la méthodologie MOPCOM. De plus, afin de palier au manque d'exécution des modèles UML, nous proposons une solution en SystemC pour simuler à haut niveau un système reconfigurable. Nous proposons un opérateur reconfigurable pour modéliser une tâche reconfigurable. La mise en œuvre d'une transmission sans-fil réelle RLR, basée sur une chaîne de traitement du signal numérique modélisée en SystemC, permet de donner plus de poids à notre proposition. Enfin, le développement du modèle SystemC nous a permis de suggérer des améliorations à apporter à l'architecture HDCRAM, notamment au niveau de la manière dont celle-ci est déployée.

Conclusion générale

Les évolutions technologiques, tant dans le domaine de la physique, que de l'électronique, permettent de proposer dès aujourd'hui, et d'autant plus dans les années futures, des systèmes électroniques temps réels embarqués offrant de plus en plus de fonctionnalités. Ce phénomène implique que les systèmes électroniques sont de plus en plus complexes à concevoir de par leur hétérogénéité. Pour concevoir de tels systèmes la communauté de l'EDA (*Electronic Design Automation*) essaye de proposer, depuis de nombreuses années, des outils de plus en plus performants pour aider les équipes de conception et améliorer la productivité. Cependant, en ce début de XXI^{ème} siècle, le constat fait par l'ITRS et la communauté industrielle du domaine des systèmes électroniques embarqués est sans appel : les méthodes de conception actuellement utilisées, bien que faisant la part belle aux outils d'aide à la conception et à la réutilisation de composants, ne suivent pas le rythme effréné de l'évolution technologique. De ce fait, l'écart entre cette évolution technologique et la productivité s'accroît d'année en année. C'est pourquoi, il convient de trouver de nouvelles méthodes permettant de combler ce retard.

L'une des solutions préconisées par la communauté de l'ESL (*Electronic System Level*) est l'utilisation d'une méthode unifiée pour la conception conjointe logicielle/matérielle basée sur des modèles. Une telle méthodologie se doit d'offrir aux équipes de conception interdisciplinaire (système, logiciel, matériel, intégration et test) un environnement leur proposant une meilleure synergie afin d'accroitre significativement la productivité par rapport aux approches actuellement déployées. Cette synergie n'est possible que si tous les acteurs de la chaîne de développement d'un système électronique embarqué se comprenent.

Notre contribution

Nous venons de présenter dans ce mémoire une méthodologie de conception conjointe basée sur l'architecture dirigée par les modèles. Notre méthodologie s'appuie sur l'approche MDA (*Model Driven Architecture*), standardisée par l'OMG, et sur le profil MARTE (*Modeling Architecture Real Time Embedded*), extension de l'UML (*Unified Modeling Language*), développé pour modéliser les systèmes électroniques embarqués. L'une des particularités de cette méthodologie est sa formalisation qui permet de guider les équipes de conception à chaque étape de la méthodologie tout en laissant une certaine souplesse dans sa mise-en-œuvre.

Nos travaux se sont focalisés sur trois axes :

- La formalisation d'une méthodologie MDA couvrant le cycle de développement d'un produit depuis les spécifications système jusqu'à la réalisation. Cette étape s'est déroulée en collaboration avec des partenaires de compétences différentes ;
- Le couplage de notre méthodologie MDA avec un outil HLS (*High Level Synthesis*), dédié à la génération de code RTL (*Register Transfert Level*) ;
- L'extension de la méthodologie aux systèmes dits reconfigurables, tant au niveau applicatif que matériel dans un contexte de radio logicielle.

La formalisation de cette méthodologie en trois niveaux de modélisation, basée sur le paradigme en Y proposé par MDA, offre aux équipes de conception un cadre de développement à différents niveaux de représentation permettant de se concentrer sur différentes préoccupations, étape par étape, ce qui est de notre point de vue un point fort de cette méthodologie. De plus, nous avons tenté de proposer un cadre d'utilisation du profil MARTE, profil encore peu utilisé. Cependant, la méthodologie demande encore à être éprouvée et notamment, par des personnes n'ayant pas participé à son développement. De par nos retours d'expérience, nous pouvons noter que la méthodologie proposée n'est aujourd'hui pas totalement indépendante des outils nécessaires à sa mise en œuvre et tout particulièrement de l'outil de modélisation UML utilisé. Nous pouvons également citer que l'automatisation de toutes les transformations (modèle vers modèle et modèle vers texte) ne sont pas encore mises en œuvre, notamment pour le passage d'un niveau de modélisation à un autre. Toutefois, est-il nécessaire d'automatiser toutes les étapes de transformations ? N'est-il pas plus réaliste de proposer une aide à ces transformations et laisser une certaine souplesse et initiative aux équipes de conception ?

Nous sommes arrivés à la conclusion qu'une méthodologie de conception conjointe basée sur les modèles ne peut être complète sans proposer des liens avec les outils de l'EDA et l'ESL. Dans nos travaux, nous avons pris le parti de coupler cette méthodologie avec les outils de synthèse de haut niveau pour proposer une génération de code RTL des ressources matérielles, tout en utilisant la génération du squelette et de l'architecture matérielle depuis la modélisation UML. Il nous semble évident que la génération de tout le code HDL (*Hardware Design Language*) d'un système à partir d'une modélisation UML n'est pas pertinente. En dehors des objectifs de ce mémoire, une autre piste, complémentaire à notre approche, est également proposée avec cette méthodologie : l'utilisation de librairies de composants sous forme d'IP (*Intellectual Property*). Notre approche permet aux ingénieurs matériel de se focaliser sur l'architecture des IPs générées par le biais d'outils HLS et de ne plus se soucier du développement de code HDL en tant que tel. Au vu des résultats obtenus sur le mécanisme de couplage, nous ne remettons pas en cause notre choix, cependant ces résultats mettent en évidence certaines limites à cette approche. De par nos expérimentations, nous avons pu mettre en évidence l'importance d'anticiper l'utilisation de tels outils dans la découpe fonctionnelle et la modélisation UML. Ces limites ne sont pas uniquement dues, en tant que telles, au mécanisme de couplage modèle UML/outil HLS, mais aux outils HLS. En effet, nous avons pu mettre en évidence que les performances du code HDL généré dépendent de la complexité de la fonction d'entrée et de la maîtrise par le concepteur de l'outil de synthèse de haut niveau.

Le besoin en ressources de calculs est aujourd'hui de plus en plus important mais limité par les contraintes du contexte de l'embarqué. Pour contourner cet obstacle potentiel, les systèmes sont de plus en plus configurables. Non seulement l'application est configurable mais la plate-forme matérielle est également configurable voire, aujourd'hui, reconfigurable à la volée avec l'utilisation de certaines technologies FPGA (*Field Programmable Gate Array*). De ce constat, nous proposons d'étendre la méthodologie aux systèmes reconfigurables. Pour cela nous avons intégré au niveau de la modélisation UML, mais aussi au niveau SystemC, une architecture de gestion hiérarchique distribuée pour la reconfiguration et la prise de décisions, issue des travaux de l'équipe SCEE de Supélec. De plus, la génération de code UML vers SystemC permet de simuler le comportement dynamique du système et tout particulièrement d'observer l'impact de la

reconfiguration sur le comportement global du système et cela à différents niveaux de la conception. Cette facilité est d'après nous un besoin indispensable pour les futurs systèmes flexibles, de radio logicielle notamment, permettant ainsi de valider au plus tôt, par simulation, les modèles développés sur une plate-forme radio réelle.

Les résultats obtenus lors de ces travaux de thèse sont prometteurs. Nous avons pu montrer tout l'intérêt d'une méthodologie formalisée, unifiée et basée sur des modèles. Toutefois à ce stade, la méthodologie proposée dans le cadre du projet MOPCOM manque de maturité, et les expérimentations menées dans ces travaux de thèse ne permettent pas d'affirmer que la mise en œuvre de cette méthodologie dans le cadre de la conception d'un système embarqué permettra un gain significatif de la productivité. Cependant, il va de soi que pour prouver définitivement l'intérêt d'une telle méthodologie, aux industriels du domaine de l'électronique embarquée, il est nécessaire de compléter ces travaux de thèse par une évaluation quantitative plus complète.

Perspectives

Dans ces travaux de thèse nous proposons une première définition d'une méthodologie de co-conception basée sur l'approche MDA afin de répondre au manque de productivité des méthodologies actuelles pour les systèmes électroniques embarqués. De par nos évaluations, nous pouvons préciser qu'il reste de nombreux points à améliorer, et d'autres non encore traités dans la méthodologie proposée. Cependant, ces travaux n'ont pas remis en cause notre idée première qu'une méthodologie MDA est une solution prometteuse.

L'une des toutes premières perspectives à ces travaux de thèse serait de développer un système temps réel embarqué en mettant en œuvre, d'un côté, notre méthodologie, et de l'autre, une méthodologie de conception conjointe éprouvée, afin de comparer leurs résultats. De plus, il serait nécessaire de multiplier l'utilisation de la méthodologie MOPCOM pour développer des systèmes embarqués ; développement assuré par des ingénieurs n'ayant pas contribués au développement de cette méthodologie afin d'obtenir un retour sur expérience réellement objectif. Ceci permettrait de consolider et d'entériner les choix de modélisation UML/MARTE faits pour les 3 niveaux d'abstractions proposés, et de passer, ensuite, à une étape de diffusion et d'utilisation de celle-ci à plus grande échelle.

Une des extensions directes de notre travail est la poursuite des travaux sur la modélisation de systèmes dits reconfigurables. Pour cela, l'équipe SCEE prévoit d'affiner l'architecture HDCRAM (*Hierarchical and Distributed Cognitive Radio Management*) afin de pouvoir l'utiliser au niveau de la modélisation EML (*Execution Modeling Level*) et DML (*Detailed Modeling Level*). Il faut pour cela définir de manière exhaustive la messagerie qui transite entre les différents modules de l'architecture de gestion de reconfiguration. Ceci pourrait permettre ensuite, par exemple, en utilisant les outils mis en place dans le cadre des travaux de thèse de Jorgiano VIDAL [105], de générer une plate-forme reconfigurable pour cible FPGA.

Dans ce même contexte, il est également nécessaire de poursuivre les travaux sur la modélisation SystemC afin de proposer une simulation des modèles aux 3 niveaux de modélisation UML afin de valider chacun de ces 3 niveaux. Aujourd'hui, seul le niveau AML (*Abstract Modeling Level*) a été transcrit en un équivalent SystemC TLM (*Transfert*

Level Modeling). Il faudrait pour cela proposer un générateur de code UML vers SystemC respectant les 3 niveaux de modélisation. De plus, il serait également intéressant de coupler notre méthodologie avec un environnement d'outils tel qu'OpenTLM [145], environnement sous licence libre qui offre un outillage pour aider à la modélisation SystemC/TLM de systèmes complexes. Cet outil permettrait de simuler et de valider les composants logiciels dudit système en avance de phase sur une représentation virtuelle (abstraite) de la plate-forme d'exécution réelle. Il nous semble également intéressant de se pencher sur les travaux du projet SoClib [146] qui propose une plate-forme virtuelle ouverte pour réaliser du prototypage de système de multiprocesseurs utilisant des modèles SystemC/TLM, modèles disponibles sous forme d'une librairie.

Au niveau de la modélisation fonctionnelle (PIM – *Platform Independant Model*), il nous semble également intéressant de coupler notre méthodologie avec des modèles Matlab comme point d'entrée du niveau AML. Il n'y a rien de nouveau dans cette approche car des outils de modélisation UML, tel Rational Rhaposdy, proposent déjà un couplage Matlab/UML. Cela permettrait dans le cas de notre méthodologie de récupérer des modèles fonctionnels Matlab développés par des algorithmiciens pour les transformer en modèles UML ou proposer une co-simulation Matab/modèle MOPCOM. Ceci permettrait de se focaliser sur l'architecture fonctionnelle et non sur la modélisation des différentes fonctions en elle-même du PIM.

Il pourrait également être intéressant de coupler notre méthodologie MDA avec des environnements d'outils sous licence libre de développement MDD (*Model Driven Development)* de systèmes embarqués complexes tel que OpenEmbeDD [147] ou encore TopCased [148], environnement de développement MDD basé sur Eclipse. TopCased propose sous licence libre les outils pour faire de la modélisation, de la génération de code, de la transformation de modèle, et de la génération documentaire mais ne proposant pas de méthodologie de conception.

Enfin, dans une perspective plus large et dans un horizon plus lointain, nous trouvons également intéressant d'étudier un rapprochement entre les méthodologies de conception conjointe basées sur des modèles de haut niveau comme nous le proposons dans ce mémoire de thèse et la méthode XP (*eXtreme Programming*) [149] utilisée dans le domaine l'informatique et plus particulièrement dans le génie logiciel. Cette méthode, basée sur la méthode Agile, pousse à l'extrême la notion de simplicité afin de faciliter l'adaptation à des changements en cours de développement d'un produit.

Pour conclure, notre sentiment sur les méthodologies de conception conjointe pour les systèmes électroniques temps réel embarqués basées sur l'approche IDM (*Ingénierie Dirigée par les Modèles*), et tous ses dérivés tels que le MDA, est qu'il reste encore un long chemin avant de voir s'implanter au sein des entreprises de telles méthodologies. Aujourd'hui, le manque de maturité de ces approches, le manque de maturité des outils associés, le bien fondé de leur apport encore peu exhaustif, et la culture industrielle fait que nous sommes encore loin de voir de telles méthodologies s'implanter en masse et significativement. Il faut donc aujourd'hui à la fois poursuivre les travaux engagés depuis près d'une décennie, mettre en évidence des résultats tangibles et, en parallèle, donner une place plus importante à tous les concepts abordés dans ce mémoire au niveau des formations des futurs ingénieurs et chercheurs du domaine de l'électronique.

Annexe 1 : Décomposition QR selon Gram-Schmidt

La décomposition QR (aussi appelée factorisation QR) d'une matrice est une décomposition d'une matrice en une matrice orthogonal et une matrice triangulaire. La décomposition QR d'une matrice carrée réelle *A* est donnée par :

$$A = QR,$$

où Q est une matrice orthogonale (tel que $Q^TQ = I$) et R est une matrice triangulaire supérieure.

Note : si la matrice *A* est non-singulière, la factorisation est unique.

Processus de Gram-Schmidt

En considérant la procédure Gram-Schmidt, avec des vecteurs considérés dans le processus comme des colonnes de la matrice *A*. Ce qui donne :

$$A = [a_1|a_2|\dots|a_n]$$

Alors,

$u_1 = a_1$ et $e_1 = \frac{u_1}{\|u_1\|}$;

$u_2 = a_2 - (a_2 \cdot e_1)e_1$ et $e_2 = \frac{u_2}{\|u_2\|}$;

$u_{k+1} = a_{k+1} - (a_{k+1} \cdot e_1)e_1 - \cdots - (a_{k+1} \cdot e_k)e_k$ et $e_{k+1} = \frac{u_{k+1}}{\|u_{k+1}\|}$

Décomposition QR

Le résultat de la décomposition QR de la matrice *A* est :

$$A = [a_1|a_2|\dots|a_n] = [e_1|e_2|\dots|e_n]\begin{bmatrix} a_1 \cdot e_1 & a_2 \cdot e_1 & \dots & a_n \cdot e_1 \\ 0 & a_2 \cdot e_2 & \cdots & a_n \cdot e_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \cdot e_n \end{bmatrix} = QR.$$

Exemple

Considérons la matrice suivante :

$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$, avec les vecteurs $a_1 = (1,1,0)^T$, $a_2 = (1,0,1)^T$, $a_3 = (0,1,1)^T$.

En utilisant la procédure de Gram-Schmidt, nous obtenons :

$u_1 = a_1 = (1,1,0)$,

$e_1 = \frac{u_1}{\|u_1\|} = \frac{1}{\sqrt{2}}(1,1,0) = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right)$,

$u_2 = a_2 - (a_2 \cdot e_1)e_1 = (1,1,0) - \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right) = \left(\frac{1}{2}, -\frac{1}{2}, 1\right)$,

$e_2 = \frac{u_2}{\|u_2\|} = \frac{1}{\sqrt{3/2}}\left(\frac{1}{2}, -\frac{1}{2}, 1\right) = \left(\frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{6}}, \frac{2}{\sqrt{6}}\right)$,

$u_3 = a_3 - (a_3 \cdot e_1)e_1 - (a_3 \cdot e_2)e_2$,

$u_3 = (0,1,1) - \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right) - \frac{1}{\sqrt{6}}\left(\frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{6}}, \frac{2}{\sqrt{6}}\right) = \left(-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$,

$e_3 = \frac{u_3}{\|u_3\|} = \left(-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$,

Ainsi donc nous obtenons les matrices *Q* et *R* suivantes :

$$Q = [e_1|e_2|e_3] = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}} \\ 0 & \frac{3}{\sqrt{6}} & \frac{1}{\sqrt{3}} \end{bmatrix},$$

$$R = \begin{bmatrix} a_1 \cdot e_1 & a_2 \cdot e_1 & \dots & a_n \cdot e_1 \\ 0 & a_2 \cdot e_2 & \cdots & a_n \cdot e_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \cdot e_n \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{3}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ 0 & 0 & \frac{2}{\sqrt{3}} \end{bmatrix}.$$

Annexe 2 : Contraintes de modélisation du niveau EML de MOPCOM

Cette annexe est un extrait du document intitulé « *Metamodels specification - Annex 1 - Constraints added by Mopcom* » du projet ANR MOPCOM SoC/SoPC.

CHAPTER 2

EML constraints

This chapter presents the Mopcom constraints applicable to models of the EML level.

As stated in the document "Processus de modélisation" - Spécification des métamodèles, the EML level metamodel is a refinement of the UML plus the MARTE profile metamodel. This refinement is done by adding constraints on those metaclasses.

The constraints are sorted by Metaclass on which they are verified.

The Mopcom process declares 12 invariants to check this level conformance.

> **Caution**
>
> 1 of them are declared unfinished.

> **Warning**
>
> 3 of them are declared not implemented.

2.1. uml::Model

List of the Mopccom constraints defined on the metaclass `uml::Model`

2.1.1. existHwClockInModel

EML model must contain at least one Hardware clock (needs to be validated)

Mopcom Constraint UID = `EML???`

Kermeta constraint :

```
do
   var ph : profiledResourceHelper::ProfilesHelper init profiledResourceHelper::ProfilesHelper.new
   ph.initialize(self.containingResource, self.containingResource)
   if mopcomMLModelHelper::EMLModelHelper.new.isMLPackage(self) then
      self.containsAnHwClock(ph)
   else
      true
   end
end
```

2.1.2. existsEMLPackage

Model must have an EML package

Mopcom Constraint UID = `EML001`

Kermeta constraint :

```
mopcomMLModelHelper::EMLModelHelper.new.containsMLPackage(self)
```

2.2. uml::Operation

List of the Mopccom constraints defined on the metaclass `uml::Operation`

2.2.1. notEmptyBody

A concrete operation's body cannot be empty

> **Caution**
>
> Implemented = yes, but it is probably too strong

Mopcom Constraint UID = `COM00`

Kermeta constraint :

```
if self.isAbstract then
   true
else
   self.method.size.isGreater(0)
end
```

2.3. uml::Package

List of the Mopccom constraints defined on the metaclass `uml::Package`

2.3.1. existsApplicationPackageInEMLPackage

The model has at least one "Application" subpackage in the "EML" package

Mopcom Constraint UID = `EML003`

Kermeta constraint :

```
do
   var modelHelper : mopcomMLModelHelper::EMLModelHelper init mopcomMLModelHelper::EMLModelHelper.new
   if modelHelper.isMLPackage(self) then
      self.packagedElement.select{elem |
         elem.isInstanceOf(uml::Package)}.exists{packElem |
         packElem.asType(uml::Package).name.equals("Application")}
```

```
    else
      true
    end
end
```

2.3.2. existsPlatformPackageInEMLPackage

The model has at least one "Platform" subpackage in the "EML" package

Mopcom Constraint UID = `EML004`

Kermeta constraint :

```
do
    var modelHelper : mopcomMLModelHelper::EMLModelHelper init mopcomMLModelHelper::EMLModelHelper.new
    if modelHelper.isMLPackage(self) then
        self.packagedElement.select{elem |
            elem.isInstanceOf(uml::Package)}.exists{packElem |
            packElem.asType(uml::Package).name.equals("Platform")}
    else
        true
    end
end
```

2.3.3. existsComputingResourceInEMLPlatformPackage

The "Platform" package has one "ComputingResource" class as top level(or one of its descendant stereotypes)

Mopcom Constraint UID = `EML005`

Kermeta constraint :

```
do
    var ph : profiledResourceHelper::ProfilesHelper init profiledResourceHelper::ProfilesHelper.new
    ph.initialize(self.containingResource, self.containingResource)
    var modelHelper : mopcomMLModelHelper::EMLModelHelper init mopcomMLModelHelper::EMLModelHelper.new
    if self.name.equals("Platform").and(modelHelper.isInML(self)) then
        self.packagedElement.select{elem |
            elem.isInstanceOf(uml::Classifier)}.exists{classifierElem |
            ph.getStereotypesFor(classifierElem).exists{st |
                st.isInstanceOf(MARTE::MARTE_Foundations::GRM::ComputingResource)}}
    else
        true
    end
end
```

2.3.4. existsLogicalClockInEMLPackage

il y a bien au moins une horloge logiqueï¿½(on ne tient pas compte dï¿½ï¿½ventuel changement dï¿½horloge ï¿½ ce niveau)

> **Warning**
>
> Implemented = no, need precision, what is a logical clock ? (an HwLogical::Hwtiming::HwClock)?

Mopcom Constraint UID = `EML006`

2.4. uml::Classifier

List of the Mopccom constraints defined on the metaclass `uml::Classifier`

2.4.1. useOnlyGRMStereotypes

Utilisation Uniquement des stï¿½rï¿½otypes prï¿½sents dans le paquetage "GRM"

Warning

Implemented = no, need precision about what is allowed or forbidden and in which subpackages of EML

Mopcom Constraint UID = `EML002`

Liste des figures

Liste des tableaux

Contributions de l'auteur

Revue internationale

Stéphane Lecomte, Samuel Guillouard, Christophe Moy, Pierre Leray, Philippe Soulard, « *A co-design methodology based on Model Driven Architecture for Real Time Embedded systems* », publié dans la revue Mathematical and Computing Modelling, pour la Special Issue intitulée Telecommunications Software Engineering: Emerging Methods, Models and Tools, Vol. 53, Issue 3-4, pp. 471 – 484, Février 2011.

Publications internationales avec actes

Stéphane Lecomte, Christophe Moy, Pierre Leray, « *Multi-level Modeling and Simulation of Cognitive Radio Equipments* », SDR Technical Conference, Washington, D.C. ,USA, 30 novembre – 3 décembre 2010.

Stéphane Lecomte, Samuel Guillouard, Christophe Moy, Pierre Leray, Philippe Soulard, « *A co-design methodology based on model driven engineering for SDR equipments* », SDR Technical Conference, Washington, D.C., USA, 30 novembre – 3 décembre 2009.

Chapitre de livre

Denis Aulagnier, Ali Koudri, Stéphane Lecomte, Philippe Soulard, Joël Champeau, Jorgiano Vidal, Gilles Perrouin, Pierre Leray, « *SoC/SoPC development using MDD and MARTE profile* », publié dans le livre intitulé Model Driven Engineering for distributed Real-Time Systems, édition ISTE & WILEY, 2010.

Communications sans actes

Stéphane Lecomte, « *Conception de système de transmission sans-fil basée sur la modélisation et la synthèse de haut niveau* », Journée thématique GdR ISIS, Paris, France, 21 octobre 2010.

Stéphane Lecomte, « *Conception de systèmes embarqués basée sur des approches de haut niveau* », Journée des doctorants de l'IETR, Rennes, France, 16 juin 2010.

Stéphane Lecomte, Pierre Leray, Christophe Moy, « *Using co-design methodology based on high-level model for reconfigurable embedded systems* », Journée thématique du GdR SoC/SiP intitulée, Paris, France, 19 novembre 2009.

Stéphane Lecomte, « *Models driven co-design methodology for SDR systems* », Séminaire SCEE, Rennes, France, 24 septembre 2009.

Bibliographie

[1] Gordon E. Moore, « *Cramming more components onto integrated circuits* », Electronics Magazine, Vol. 38, No. 8, Avril 1965.

[2] Wikipédia, « *Loi de Moore* », http://fr.wikipedia.org

[3] C. Jego, « *Conception de systèmes numériques le CoDesign*, édition 2006 », cours de l'ENST Bretagne, 2006.

[4] ITRS, « *Design, edition 2009* », http://www.itrs.net, 2009 [4.4]

[5] J. Burr, A. Chandrakasan, F. Assaderaghi, F. Catthoor, F. Fox, D. Greehill, D. Singh, J. Sproch, « *Low power design without compromise (panel)* », in proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), New-York, NY, USA, 1997, doi : 10.1145/263272.263355

[6] ITU ou UIT, http://www.itu.int [1.2.1]

[7] E. H. Armstrong, « *The super-heterodyne-its origin, development, and some recent improvements* », in proceedings of the Institute Radio Engineers (IRE), Vol. 12, Issue 5, pp. 539 – 552, Octobre 1924, doi : 10.1109/JRPROC.1924.219990 [1.2.1]

[8] IEEE standard association, « *IEEE 802.16 : Broadband Wireless Metropolitan Area Networks* », http://standards.ieee.org/ [1.2.2.1]

[9] F. Le Bolzer, S. Guillouard, C. Guguen, P. Fontaine, R. Monnier, « *Prodim@ges - A new Video Production Environment based on IP wireless and optical links* », NEM'SUMMIT, Saint-Malo, France, Octobre 2008. [1.2.2.1]

[10] IEEE standard association, « *IEEE 802.3 : IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks* », http://standards.ieee.org/ [1.2.2.1]

[11] ISO/IEC, « *Standard 7498-1994* », http://www.iso.org [1.2.2.1]

[12] J. Proakis, « *Digital Communications, 5th edition* », McGraw-Hill Science, ISBN-13 : 978-0072957167, p. 1150, 2007. [1.2.2.1]

[13] S. M. Alamouti, « *A simple transmit diversity technique for wireless communications* », in IEEE Journal on select areas in Communications, Vol. 16, No. 8, pp. 1451 – 1458, Octobre 1998. [1.2.2.1]

[14] R. van Nee, « *OFDM for Wireless Multimedia Communications* », Artech House Publishers, ISBN-13 : 978-0890065303, p. 284, 1999. [1.2.2.1]

[15] A. S. Householder, « *Unitary Triangularization of a Nonsymmetric Matrix* », in Journal of ACM, Vol. 5, Issue 4, pp. 339 – 342, 1958, doi : 10.1145/320941.320947 [1.2.2.3]

[16] W. Givens, « *Computation of plane unitary rotations transforming a general matrix to triangular form* », in Journal of the Society for Industrial and Applied Mathematics, Vol. 6, No.1, pp. 26 – 50, 1958. [1.2.2.3]

[17] A. Salah, A. Othman, G. Ouertani, S. Guillouard, « *New soft decoder for MIMO channel* », in proceedings of 42th Asilomar Conference on Signal, Systems and

Computers (ACSSC), Pacific Grove, CA, USA, pp. 1754 – 1758, Octobre 2008, doi : 10.1109/ACSSC.2008.5074727 [1.2.2.3]

[18] R.I. Lackey, D.W. Upmal, « *SPEAKeasy, the Military Software Radio* », publié dans IEEE Communications Magazine, Vol. 33, Issue 5, pp. 56 – 61, Mai 1995, doi : 10.1109/35.392998 [1.3.1]

[19] J. Mitola, « *Software radios-survey, critical evaluation and future directions* », in proceedings on the National Telesystems Conference, Washington, DC, USA, Mai 1992, doi : 10.1109/NTC.1992.267870 [1.3.1]

[20] J. Mitola, « *The software Radio Architecture* », IEEE Communications Magazine, Vol. 33, Issue 5, pp. 26 – 38, Mai 1995, doi : 10.1109/35.393001 [1.3.1]

[21] The Wireless Innovation Forum, http://www.wirelessinnovation.org/ [1.3.1]

[22] GNU radio, http://gnuradio.org [1.3.1]

[23] JTRS, « *Software Communication Architecture* », http://sca.jpeojtrs.mil/ [1.3.1], [5.1.2]

[24] OCCAR, « *ESSOR: European Secure SOftware defined Radio* », http://www.occar-ea.org [1.3.1]

[25] Euler consortium, « *EULER - EUropean Software Defined radio for wireless in joint security operations* », projet européen FP7, http://www.euler-project.eu/ [1.3.1]

[26] J.P. Delahaye, « *Plate-forme hétérogène reconfigurable : application à la radio logicielle* », PhD Thesis, Ecole SUPérieur d'Electricité, 2007. [1.3.1], [5.1.1.2], [5.1.2]

[27] L. Godard, « *Modèle de gestion hiérarchique distribuée pour la reconfiguration et la prise de décision dans les équipements de la radio cognitive* », PhD thesis, Ecole SUPérieur d'Electricité, Décembre 2008. [1.3.1], [5.1.2]

[28] L. Godard, C. Moy, J. Palicot, « *An Executable Meta-Model of a Hierarchical and Distributed Architecture Management for the Design of Cognitive Radio Equipments* », Annals of Telecommunications, Special Issue on Cognitive Radio, Springer Editions, Vol. 64, No. 7 – 8, pp. 463 – 482, Août 2009. [1.3.1], [5.1.2]

[29] J. Palicot, « *De la radio logicielle à la radio intelligente* », publié aux éditions Hermes Science Publications, Collection Télécom, Novembre 2010. [1.3.1]

[30] Xilinx, « *ML506 reference designs* », http://www.xilinx.com [1.3.2.2], [5.2.3.1]

[31] J. Delorme, J. Martin, A. Nafkha, C. Moy, F. Clermidy, P. Leray, J. Palicot, « *A FPGA partial reconfiguration design approach for cognitive radio based on NoC architecture* », in NEWCAS, Montréal, Canada, Juin 2008. [1.3.2.2]

[32] J. Delorme, A. Nafkha, P. Leray, C. Moy, « *New OPBHWICAP interface for real-time Partial reconfiguration of FPGA* », in International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, Décembre 2009. [1.3.2.2], [5.1.1.2]

[33] T.A. Henzinger, J. Sifakis, « *The Discipline of Embedded Systems Desig*n », publié dans le journal IEEE Society Computer, Vol. 40, Issue 10, pp. 32 – 40, Octobre 2007, doi : 10.1109/MC.2007.364 [2.1]

[34] A. Koudri, S. Metafli, J.L. Dekeyser, « *IP Integration in embedded system modeling* », in 14th IP based Soc Design Conference (IP-SoC), Grenoble, France, 2005. [2.1.3]

[35] The SPIRIT Consortium (Accellera), « *IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tools Flows*», IEEE 1685-2009, http://www.accellera.org/activities/ip-xact/ [2.1.3], [2.2.3.3]

[36] G. Stitt, F. Vahid, W. Najjar, « *A code refinement methodology for performance-improved synthesis from c* », in proceedings of the International Conference on Computer-aided Design (ICCAD), IEEE/ACM, pp. 716 – 723, New-York, NY, USA, 2006, doi : 10.1145/1233501.1233649 [2.1.3]

[37] A. Sangiovanni-Vincentelli, L. Carloni, F. de Bernadinis, M. Sgroi, « *Benefits and challenges for platform-based design* » in proceedings of the 41st annual Design Automation Conference (DAC), ACM, pp. 409 – 414, New-York, NY, USA, 2004, doi : 10.1145/996566.996684 [2.1.3], [2.3.1.4]

[38] D. Densmore, R. Passerone, A. Sangiovanni-Vincentelli, « *A Platform-Based Taxonomy for ESL Design* », IEEE Design and Test of Computers, Vol. 23, No. 5, pp. 359 – 374, Sep./Oct. 2006, doi : 10.1109/MDT.2006.112 [2.1.3]

[39] D. C. Schmidt, « *Model-Driven Engineering* », in Computer Society journal, IEEE, Vol. 39, Issue 2, pp. 25 – 31, Février 2006, doi : 10.1109/MC.2006.58 [2.2.1]

[40] Object Management Group, « *MDA Guide Version 1.0.1* », omg/2003-06-01, Juin 2003. [2.2.1], [2.2.2]

[41] Jean-Marie FAVRE, « *Concepts fondamentaux de l'IDM. De l'ancienne Egypte à l'Ingénierie des Langages* », Journées sur l'Ingénierie Dirigée par les Modèles (IDM), Lille, France, Juin 2006. [2.2.1.1]

[42] Object Management Group, « *Meta Object Facility (MOF) Core Specification, version 2.4 Beta 2.0* », ptc/2010-12-08, 2010. [2.2.1.2], [2.2.2]

[43] P.A. Muller, F. Fleurey, D. Vojtisek, Z. Drey, D. Pollet, F. Fondement, P. Studer, et J.M. Jézéquel, « *On Executable Meta-Languages applied to Model Transformations* », in Model Transformations In Practice Workshop, Montego Bay, Jamaïque, 2005. [2.2.1.3], [3.1.2], [3.1.4], [3.3.4.2], [5.1.2]

[44] Eclipse, « *Atlas Transformation Language* », http://www.eclipse.org/atl/ [2.2.1.3]

[45] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, « *EMF: Eclipse Modeling Framework, 2nd Edition* », édition Addison-Wesley Professional, Collection Eclipse Series, ISBN-10 : 0-321-33188-5, 2008. [2.2.1.3]

[46] Object Management Group, « *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification* », formal/2011-01-01, 2011. [2.2.1.3]

[47] Object Management Group, « *MDA* », http://www.omg.org/mda/index.htm [2.2.2]

[48] Object Management Group, « *Common Warehouse Metamodel (CWM) Specification, version 1.1* », formal/2003-03-02, 2003. [2.2.2]

[49] Object Management Group, « *Unified modeling language (UML) superstructure specification, version 2.3* », formal/2010-05-05, 2010. [2.2.2], [2.2.3]

[50] SAE, « *Architecture Analysis & Design Language* », standard AS5506A, 2009-01-20, http://standards.sae.org [2.2.3]

[51] G. Booch, « *The Booch method: process and pragmatics* », Object development methods, SIGS Publications Inc., ISBN : 0-9627477-9-3, New-York, NY, USA, pp. 149 – 166, 1994. [2.2.3]

[52] J. Rumbaugh, M. Blaha, W. Lorensen, F. Eddy, W. Prermerlani, « *Object-Oriented Modeling and Design* », 1st edition, Prentice-Hall, ISBN : 0136298419, Octobre 1990. [2.2.3]

[53] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard, « *Object-Oriented Software Engineering: A Use Case Driven Approach* », HarlowEssex England Addison, Addison-Wesley, ISBN : 0201544350, p. 552, 1992. [2.2.3]

[54] G. Martin, « *UML for Embedded Systems Specification and Design: Motivation and Overview* », in proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE), Paris, France, Mars 2002, doi : 10.1109/DATE.2002.998386 [2.2.3]

[55] Y. Wanderperren, W. Mueller, W. Dehaene, *« UML for Electronic Systems Design: A Comprehensive overview* », in book Design Automation for Embedded Systems, edition Springer, ISSN : 0929-5585, Vol. 12, No. 4, pp. 261 – 292, 2008, doi : 10.1007/s10617-008-9028-9 [2.2.3]

[56] Object Management Group, « *OMG Systems Modeling Language, version 1.2* »**,** formal/2010-06-01, 2010. [2.2.3.2], [3.1.1]

[57] Object Management Group, « *UML™ Profil for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms specification, version 1.1* », formal/2008-04-05, 2008. [2.2.3.2], [3.1.1]

[58] Object Management Group, « *UML Profil for System on a Chip (SoC), version 1.0.1* », formal/06-09-01, 2006. [2.2.3.2], [3.1.1]

[59] Object Management Group, *« UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, version 1.0* », formal/2009-11-02, 2010, http://www.omg.org/spec/MARTE/ [2.2.3.2], [3.1.1]

[60] Object Management Group, *« UML™ Profil for Schedulability, Performance and Time specification, version 1.1* », formal/05-01-02, 2005. [2.2.3.2], [3.1.1]

[61] E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio, « *A UML 2.0 Profile for SystemC: Toward High Level SoC* Design », in proceedings of the 5th ACM international conference on EMbedded SOFTware (EMSOFT), Jersey City, NJ, USA, Septembre 2005, doi : 10.1145/1086228.1086254 [2.2.3.2], [2.3.2.2], [3.1.3.1], [5.3.2]

[62] B. Selic, « *UML-RT: A profile for modeling complex real-time architectures* », Draft ObjectTime Limited, Décembre 1999. [2.2.3.2]

[63] B. Selic, « *Real-time object-oriented modeling (ROOM)* », in proceedings of the Real Time Technology and Applications Symposium (RTAS), IEEE Computer Society, pp. 214, 1996, doi : 10.1109/RTTAS.1996.509538 [2.2.3.2]

[64] Object Management Group, « *MOF 2 XMI Mapping, Version 2.4* », ptc/2010-12-06, 2010, http://www.omg.org/spec/XMI/ [2.2.3.3]

[65] Object Management Group, « *XML/Value type Language Mapping Specification* », formal/2003-04-01, 2003, http://www.omg.org/spec/XML/ [2.2.3.3]

[66] IBM Rational, « *Rational Rose* », http://www-01.ibm.com/software/awdtools/developer/rose/, [2.2.3.4]

[67] IBM Rational, « *Rational Rhapsody UML Modeler* », http://www.-01.ibm.com/software/awdtools/rhapsody/ [2.2.3.4], [3.1.4], [3.2.6], [3.3.4.2], [4.2.2], [5.2.4]

[68] AUTOSAR consortium, « *AUTOmotive System Architecture* », http://www.autosar.org/ [2.2.3.4]

[69] DoDAF Working Group, « *DoD Architecture Framework, Volume I: Definitions and Guidelines, version 1.5* », Avril 2007. [2.2.3.4]

[70] The UK Ministry of Defence, « *MOD Architecture Framework, V1.2.004 Released* », http://www.mod.uk/, Mai 2010. [2.2.3.4]

[71] CEA, « *Papyrus, Open Source Tool for Graphical UML2 Modelling* », http://www.papyrusuml.org/ [2.2.3.4], [3.3.4.2]

[72] The Mathworks, « *Model-Based Design for Embedded Signal Processing with Simulink* », 2007. [2.3.1.1]

[73] The Mathworks, « *Simulink HDL Coder* », http://www.mathworks.com/. [2.3.1.1]

[74] The Mathworks, « *Des spécifications algorithmiques à la génération de code RTL* », webinar : http://www.mathworks.fr/company/events/webinars/, 29 Avril 2010. [2.3.1.1]

[75] W. Mueller, A. Rosti, S. Bocchio, E. Riccobene, P. Scandurra, W. Dehaene, Y. Vanderperren, « *Uml for esl design : basic principles, tools, and applications* », in proceedings of the International Conference on Computer-aided Design (ICCAD), IEEE/ACM, pp. 73 – 80, New-York, NY, USA, 2006. [2.3.1.1]

[76] Y. Vanderperren, W. Dehaene, « *From UML/SysML to matlab/simulink: Current state and future perspectives* », in proceedings of the conference on Design, Automation and Test in Europe (DATE), Munich, Allemagne, 2006. [2.3.1.1]

[77] T. Grandpierre, C. Lavarenne, Y. Sorel, « *Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors* », in proceedings of the 7th international workshop on Hardware/software codesign (CODES), Rome, Italie, 1999, doi : 10.1145/301177.301489. [2.3.1.2]

[78] Y. Sorel, « *SynDEx: System-level cad software for optimizing distributed real-time embedded systems* », ERCIM News, No. 59, pp. 68 – 69, October 2004, http://www.syndex.org. [2.3.1.2]

[79] E.A. Lee, S. Neuendorffer, « *Tutorial: Building Ptolemy II Models Graphically* », Technical Report No. UCB/EECS-2007-129, Electrical Engineering and Computer Sciences, University of California, Berkeley, USA, Octobre 2007. [2.3.1.3]

[80] Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, « *Ptolemy: a framework for simulating and prototyping heterogeneous systems* », IEEE, Kluwer Academic Publishers, Vol. 10, pp. 527 – 543, 2002. [2.3.1.3]

[81] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, Q. Zhu, « *A Next-Generation Design Framework for Platform-Based Design* », in proceedings Design and Verification Conference (DV-Con), Février 2007. [2.3.1.4]

[82] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, « *Metropolis : An integrated electronic system design environment* », publié dans la revue Computer, IEEE Computer Society, Vol. 36, Issue 4, pp. 45 – 52, Avril 2003. doi : 10.1109/MC.2003.1193228. [2.3.1.4]

[83] Cofluent Design et J.P. Calvez, « *The MCSE Methodology Overview* », White paper, 2009, http://www.cofluentdesign.com/ [2.3.1.5], [5.2.1]

[84] Cofluent Design, « *Cofluent Studio* », http://www.cofluentdesign.com [2.3.1.5], [5.2.1], [5.3.2]

[85] T. Robert, V. Perrier, « *Cofluent Methodology for UML* », Cofluent Design White Paper, 2010, http://www.cofluentdesign.com [2.3.1.5]

[86] No Magic Inc., « *MagicDraw* », http://www.magicdraw.com [2.3.1.5]

[87] MARTES Partners, « *MARTES project* », EUREKA ITEA project 04006, http://www.martes-itea.org [2.3.1.5], [5.3.2]

[88] V. Perrier, K. Kronlöf, « *Mobiles : l'exploration d'architecture renforce une méthode orientée service* », revue Electronique, n°195, Octobre 2008. [2.3.1.5]

[89] S. Rouxel, J.P. Diguet, N. Bulteau, J. Carre-Gourdin, J.E. Goubard, C. Moy, « *UML framework for PIM and PSM verification of SDR systems* », SDR Forum Technical Conference, Anaheim, USA, Novembre 2005. [2.3.2.1]

[90] Objecteering Software, « *Objecteering, the model-driven development tool* », http://www.objecteering.com/ [2.3.2.1]

[91] E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio, « *Designing a Unified Process for Embedded Systems* » , in proceedings of the 4th international workshop on model-based methodologies for pervasive and embedded software (MOMPES), IEEE Computer Society, pp. 77 – 90, Avril 2007, doi : 10.1109/MOMPES.2007.5 [2.3.2.2], [3.1.3.1]

[92] E. Piel, R.B. Atitallah, P. Marquet, S. Meftali, S. Niar, A. Etien, J.L. Dekeyser, P. Boulet, « *Gaspard 2: from MARTE to SystemC Simulation* », in proceeedings of the DATE'08 workshop on Modeling and Analyzis of Real-Time and Embedded Systems with the MARTE UML profile, Munich, Allemagne, Mars 2008. [2.3.2.3], [3.1.3]

[93] P. Boulet, « A*rray-OL revisited, multidimensional intensive signal processing specification* », Research Report RR-6113, INRIA, Février 2007. [2.3.2.3]

[94] F. Mischkalla, D. He, W. Mueller, « *Closing the Gap between UML-based Modeling Simulation and Synthesis of Combined HW/SW Systems* », in proceedings of the 13th Design, Automation and Test in Europe conference (DATE), Dresden, Allemagne, Mars 2010. [2.3.2.4]

[95] F. Bellard, « *QEMU: an open source processor emulator* », http://wiki.qemu.org [2.3.2.4]

[96] S. Gerard, F. Terrier, Y. Tanguy, « *Using the Model Paradigm for Real-Time Systems development: ACCORD/UML* », Advances in Object-Oriented

Information Systems, SPRINGLINK, Ed., Vol. 2426/2002 of Lecture Notes in Computer Science, pp. 260 – 269, 2002. [2.3.2.4]

[97] Y. Wang, X. G. Zhou, B. Zhou, L. Liang, C. L. Peng, « *A MDA based SoC modeling approach using UML and SystemC* », in proceedings of the 6th IEEE International Conference on Computer and Information Technology (CIT), IEEE Computer Society, pp. 245, Washington, DC, USA, 2006. [2.3.2.4]

[98] S. Demathieu, F. Thomas, C. Andre, S. Gerard, F. Terrier, « *First experiments using the UML profile for MARTE* », in proceedings on the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC), Orlando, FL, USA, Mai 2008, doi : 10.1109/ISORC.2008.36 [3.1]

[99] CEA List, Thales, INRIA, « *MARTE Tutorial, an OMG Standard: UML profile to develop Real-Time Embedded systems* », 2007. [3.1.1]

[100] Object Management Group, « *Real-time CORBA Specification, version 1.2* », formal/05-04-01, 2005. [3.1.1]

[101] Object Management Group, « *Software & Systems Process Engineering Meta-Model Specification, version 2.0* », formal/2008-04-01, 2008. [3.1.2]

[102] A. Koudri, J. Champeau, « *MODAL: a SPEM Extension to Improve Co-design Process Models* », in proceedings of the international conference on software process (ICSP), Paderborn, Allemagne, Juin 2010. [3.1.2], [3.1.4]

[103] A. Koudri, J. Champeau, J.C. Le Lann, V. Leilde, « *MoPCoM Methodology: Focus on Models of Computation* », in proceedings of the 6th European Conference on Modelling Foundations and Applications (ECMFA), Paris, Juin 2010. [3.1.2.1], [3.2.2.3], [3.3.1.2]

[104] A. Koudri, « *Méthodologie UML/MARTE pour la conception conjointe logicielle / matérielle* », Laboratoire d'Informatique et Fondamentale de Lille (LIFL), Université Lille, Juillet 2010. [3.1.2.1], [3.2.2.3]

[105] J. Vidal, « *Dynamic and partial reconfigurable embedded systems design with UML* », Thèse de doctorant, Lab-STICC, Université de Bretagne Sud, Lorient, Novembre 2010. [3.1.2.3], [5.3.1], [5.2.2.2]

[106] IEEE Computer Society, « *IEEE Standard SystemC language Reference manual* », IEEE 1666™-2005, Mars 2005. [3.1.3.1], [5.3.1], [5.3.2.2]

[107] Sodius, « *MDWorkbench® platform* », http://www.mdworkbench.com [3.1.4], [3.2.6], [3.3.4.2], [4.1.2], [5.3.2]

[108] F. Singhoff, J. Legrand, L. Nana, L. Marcé, « *Cheddar : a Flexible Real Time Scheduling Framework* », in proceedings of international conference on Special Interest Group on Ada (SIGAda), ACM, Atlanta, Georgia, USA, Novembre 2004, doi : 10.1145/1032297.1032298 [3.2.3]

[109] S.R. Chidamber, C.F. Kemerer, « *A Metrics Suite for object Oriented Design* », IEEE Transactions on Software Engineering, Vol. 20, No. 6, Juin 2006. [3.2.6]

[110] Qualixo, « *UML Quality Tool »,* http://www.qualixo.com/ [3.2.6]

[111] Bruno Pagès, « *BOUML* – a free UML tool box », http://bouml.free.fr/ [3.3.4.2]

[112] P. Coussy, A. Morawiec, « *High Level Synthesis from Algorithm to Digital Circuit* », Springer, 2008, doi : 10.1007/978-1-4020-8588-8 [4.2.1]

[113] D.L. Johannsen, « *Bristle Blocks: A Silicon Compiler* », in proceedings of the 16th Design Automation Conference (DAC), San Diego, CA, USA, pp. 310 – 313, Juin 1979. [4.2.1]

[114] D. Knapp, « *Behavioral Synthesis: Digital System Design Using the Synopsys Behavioral Compiler* », Prentice Hall, p. 231, ISBN-13: 978-0135692523, 1996. [4.2.1]

[115] R. Goering, « *Behavioral revived for ESL design* », EE Times Online article, Octobre 2004. [4.2.1]

[116] G. Moretti, « *Synfora Introduces PICO Extreme* », EE Times Online article, Février 2008. [4.2.1

[117] Mentor Graphics, « *Catapult C Synthesis User's and Reference Manual, Release 2009* », http://www.mentor.com/esl/catapult/ [4.2.1], [4.2.2.3]

[118] Mentor Graphics, « *Catapult C Synthesis Style Guide, Release 2009* ». [4.2.1], [4.2.2.3]

[119] Mentor Graphics, « *The High Level Synthesis Blue book* ». [4.2.1], [4.2.2.3]

[120] Electronic Design Automation Consortium, http://www.edac.org/ [4.2.1]

[121] Lab-STICC, « *GAUT - High-Level Synthesis tool From C to RTL* », http://www.labsticc.univ-ubs.fr/www-gaut/ [4.2.2.2]

[122] Mentor Graphics, « *ModelSim - Advanced Simulation and Debugging* », http://www.model.com/ [4.3.2.1]

[123] Altera, « *Quartus II : tool for analysis and synthesis of HDL designs* », http://www.altera.com/products/software/quartus-ii/ [4.3.2.2]

[124] A. Al Ghouwayel, « *Intérêts des techniques de Paramétrisation pour des Systèmes Radio Logicielle Reconfigurables* », Thèse de doctorant, Ecole SUPérieur d'Electricité, Novembre 2009. [5.1.1.1]

[125] S. T. Gul, « *Optimization of Multistandards Software Defined Radio Equipments: A Common Operators Approach* », Thèse de doctorant, Ecole SUPérieur d'Electricité, Novembre 2009. [5.1.1.1]

[126] L. Alaus, « *Architecture Reconfigurable pour un Equipement Radio Multistandard* », Thèse de doctorant, Ecole SUPérieur d'Electricité, Mai 2010. [5.1.1.1]

[127] Xilinx, http://www.xilinx.com/ [5.1.1.2]

[128] A. Pope, « *The corba reference guide: Understanding the common object request broker architecture* », edition Addison Wesley Publishing Company, p. 407, Décembre 1997. [5.1.2]

[129] L. Pucker, J. Holt, « *Extending the SCA core framework inside the modem architecture of a software defined radio* », IEEE communications magazine, Vol. 43, Issue 3, pp. 21 – 25, Mars 2004, doi : 10.1109/MCOM.2004.1273770 [5.1.2]

[130] J. M. Jacob, M. Uhm, « *Corba for FPGAs: Tying together GPPs, DSPs, and FPGAs* », in The Journal of Embedded Signal Processing DSP-FPGA, 2007, http://www.dsp-fpga.com [5.1.2]

[131] X. Revès, A. Gelonch, V. Marojevic, R. Ferrús, « *Software Radios: Unifying the Reconfiguration Process over Heterogeneous Platforms* », in EURASIP Journal

on Applied Signal Processing, Vol. 5, Issue 16, pp. 2626 – 2640, Septembre 2005. [5.1.2]

[132] Triskell team, « *Triskel Metamodeling Kernel* », http://www.kermeta.org/ [5.1.2]

[133] ANDRES project, http://andres.offis.de [5.2.1]

[134] A. Herrholz, F. Oppenheimer, P.A. Hartman, A. Schallenberg, W. Nebel, C. Grimm, M. Damm, J. Haase, F. Brame, F. Herrera, E. Villar, I. Sander, A. Jantsch, A.M. Fouilliart, M. Martinez, « *The ANDRES project: Analysis and design of run-time reconfigurable, heterogeneous systems* », in proceedings in international conference on Field Programmable Logic and Applications (FPL) ,Amsterdam, Pays-Bas, pp. 396 – 401, Août 2007, doi : 10.1109/FPL.2007.4380679 [5.2.1]

[135] A.V. Brito, M. Kuhnle, M. Hubner, J. Becker, E.U.K. Melcher, « *Modelling and Simulation of Dynamic and Partially Reconfigurable Systems using SystemC* », in IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Porto Alegre, Brésil, Mai 2007. [5.2.1]

[136] C.H. Tseng and P.A. Hsiung, « *A UML-Based Design Flow and Partitioning Methodology for Dynamically Reconfigurable Computing Systems* », in Workshop on UML for SoC design (UML-SoC) at DAC, San-Francisco, USA, Juillet 2006. [5.2.1]

[137] A. Barreteau, « *Techniques de modélisation transactionnelle pour le dimensionnement des futurs systèmes de radiocommunication mobiles* », Thèse de doctorant, IREENA, Polytech'Nantes, Décembre 2010. [5.2.1]

[138] I.R. Quadri, S. Meftali, et J.L. Dekeyser, « *High level modeling of Partially Dynamically Reconfigurable FPGAs based on MDE and MARTE* », in Reconfigurable Communication-centric SoCs (ReCoSoC), Barcelone, Espagne, Juillet 2008. [5.2.1]

[139] G. Erich, R. Helm, R. Johnson, and J. Vlissides, « *Design Patterns: Elements of reusable Object-Oriented Software* », Addison-Wesley, ISBN : 0-201-63361-2, 1995. [5.2.2.1]

[140] Xilinx, « *MicroBlaze Processor Reference Guide – EDK 13.1 release* », http://www.xilinx.com/tools/microblaze.htm [5.2.2.2]

[141] Open SystemC Initiative,« *SystemC* », http://www.systemc.org [5.3.1]

[142] D.C. Black, J. Donovan, B. Bunton, and A. Keist, « *SystemC: From the Ground Up* », Second Edition, Springer, ISBN : 978-0-387-69957-8, 2010. [5.3.1], [0], [5.3.1.3]

[143] T. Grötker, S. Liao, G. Martin, S. Swan, « *System Design with SystemC* », edition Springer, ISBN : 978-1-4020-7072-3, 2002. [5.3.1.3]

[144] S. Lecomte, W. Jouini**,** C. Moy, P. Leray, J. Palicot, « *A SystemC Radio-in-the-Loop Modeling for Cognitive Radio equipments* », demonstration présentée lors de SDR'10 Wireless Innovation Conference and Product Exposition, Washington, DC, USA, 30 November - 3 December 2010. [5.3.3]

[145] OpenTLM consortium, « *OpenTLM: environment of collaborated HW/SW development with TLM/SystemC simulation of embedded systems* », http://opentlm.minalogic.net/

[146] SoClib consortium, « *SoClib* », http://www.soclib.fr

[147] OpenEmbeDD consortium, « *OpenEmbeDD: an open source platform for Model Driven Engineering* », http://openembedd.org

[148] TopCased consortium, « *TopCased: the Open-Source Toolkit for Critical Systems* », http://www.topcased.org

[149] K. Beck, « *Extreme Programming Explained: Embrace Change* », édition Addison-Wesley Professional, Octobre 1999.

VU :

Le Directeur de Thèse
(Christophe MOY)

VU :

Le Responsable de l'École Doctorale

VU pour autorisation de soutenance

Rennes, le

Le Président de l'Université de Rennes 1

Guy CATHELINEAU

VU après soutenance pour autorisation de publication :

Le Président de Jury,
(Nom et Prénom)

Résumé :

Dans cette thèse nous proposons une méthodologie de co-conception logicielle/matérielle pour les systèmes de radio logicielle, et plus généralement pour les systèmes électroniques embarqués, permettant de répondre aux défis imposés par la conception de tels systèmes et d'améliorer la productivité. Basée sur une démarche de conception dirigée par les modèles (dérivée de l'ingénierie dirigée par les modèles), notre méthodologie de co-conception permet une modélisation de haut niveau, depuis des modèles UML/MARTE (extension de l'UML dédié à la modélisation matérielle), jusqu'à la mise en œuvre matérielle (génération de code VHDL) par un jeu de transformations successives et de raffinements itératifs des modèles. Pour cela, nous avons défini le niveau de modélisation intermédiaire, intitulé Execution Modeling Level, qui permet de se focaliser sur le partitionnement logiciel/matériel et sur l'exploration d'architecture afin de dimensionner la plate-forme matérielle. Afin de compléter la génération de code matérielle associée à cette méthodologie, nous mettons en avant l'intérêt de coupler une méthodologie basée sur des modèles de haut niveau avec la synthèse comportementale, ce que nous illustrons au travers de la mise en œuvre d'un décodeur MIMO. Enfin, dans un contexte de radio-logicielle, nous proposons une extension de la méthodologie afin de prendre en compte le caractère flexible des systèmes embarqués. Pour cela, nous avons intégré à notre méthodologie une architecture de gestion de la reconfiguration proposée par Supélec. Une exécution de modèles de haut niveau sur une plate-forme radio réelle nous a permis de valider notre approche.

Abstract:

In this thesis, we suggest a hardware/software co-design methodology for software radio systems, and more generally for flexible embedded electronics systems, allowing to answer the new design challenges it imposes and to improve the productivity. Our co-design methodology is based on a high-level UML/MARTE (extension of UML dedicated to the hardware modeling) modeling approach. Based on model driven architecture (derived from model driven engineering), our methodology allows to start at a high-level modeling level and go down to the hardware implementation (generation of VHDL code) by successive rules of transformation and iterative refinements of the models. For that, we defined the middle level of modeling, e.g. Execution Modeling Level, which allows focusing on hardware/software partitioning and focusing on the exploration of architecture to design the hardware platform. To complete the generation of the hardware design language, associated with this methodology, we recommend to couple a co-design methodology based on high-level models with the behavioral synthesis concept. This approach is illustrated with a MIMO decoder example. Finally, in the software radio context, we suggest an extension of the methodology in order to take into account the flexibility of the embedded systems. For that, we include into our methodology an architecture defined at Supélec to manage the reconfiguration. An execution of high-level models on a real radio platform allowed to validate our approach.