Modélisation en langage VHDL-AMS des systèmes pluridisciplinaires

David Guihal

THESE

Présentée au

LABORATOIRE D'ANALYSE ET D'ARCHITECTURE DES SYSTEMES DU CNRS

en vue de l'obtention du titre de

Docteur de l'Université Toulouse III

Ecole doctorale : Génie Electrique, Electronique, Télécommunications
Spécialité : Conception de Circuits Microélectroniques et Microsystèmes

Par

David GUIHAL

Ingénieur en Micro-Electronique

MODELISATION EN LANGAGE VHDL-AMS DES SYSTEMES PLURIDISCIPLINAIRES

Soutenue le 25 Mai 2007, devant la Commission d'Examen :

Rapporteurs

Yannick HERVE	*Maître de Conférences – HDR à l'Université Louis Pasteur Strasbourg*
Alain VACHOUX	*Professeur à l'Ecole Polytechnique Fédéral de Lausanne*

Examinateurs

Laurent ANDRIEUX	*Maître de Conférences à l'Université Toulouse II, Co-Directeur de thèse*
Daniel ESTEVE	*Directeur de recherche CNRS, Directeur de thèse*
Jean OUDINOT	*Docteur-Ingénieur, MENTOR GRAPHICS*
Christian PERCEBOIS	*Professeur à l'Université Paul Sabatier, Toulouse III*

Invité

Jean-Louis BOIZARD	*Maître de Conférences à l'IUFM Midi-Pyrénées*
Alain CAZARRE	*Profeseur à l'Université Paul Sabatier, Toulouse III*
Pascal PAMPAGNIN	*Ingénieur, AIRBUS*

A mes grands-parents partis trop vite…

Remerciements

J'exprime tous mes remerciements à Messieurs GHALLAB et CHATILA, directeurs successifs du Laboratoire d'Analyse d'Architecture des Systèmes du CNRS de Toulouse pour m'avoir permis de réaliser mes travaux de thèse en ces lieux. Je remercie très fortement Monsieur Darrell TEEGARDEN, responsable de l'équipe de développement de l'outil SystemVision au sein de la société MENTOR GRAPHICS, pour ses compétences, sa disponibilité, son écoute, et sans qui cette thèse n'aurait pu se faire. Je remercie également Madame Anne Marie GUE et Monsieur Jean-Yves FOURNIOLS, responsables successifs du groupe de recherche Microsystème et Intégration des Systèmes dans lequel j'ai pu évoluer.

Mes remerciements s'adressent tout particulièrement à mon directeur de thèse Monsieur Laurent ANDRIEUX pour son soutien, ses conseils scientifiques, et surtout son amitié qui m'ont permis de travailler dans les meilleures conditions et de garder de très bons souvenirs de ma thèse. Mes remerciements s'adressent aussi chaleureusement à Monsieur Daniel ESTEVE pour avoir co-encadré ma thèse. Sa disponibilité, ses conseils contructifs et son recul sur mes travaux m'ont guidés tout au long de ma thèse.

Mes remerciements s'adressent également à Monsieur Jean OUDINOT responsable Europe des solutions AMS pour la société MENTOR GRAPHICS, qui a permis de mettre en place cette thèse, qui s'est impliqué dans ce projet et qui a toujours veillé à ce que ma thèse se déroule dans les meilleures conditions. Je remercie également Monsieur Eric RONGERE, ingénieur d'application dans la société MENTOR GRAPHICS, pour sa disponiblité et son amitié. J'en profite aussi pour remercier l'ensemble de l'équipe de développement de SystemVision, avec qui j'ai pu travailler, bien souvent à distance, au cours de ces trois années de thèse (Subba Somanchi, Mike Donnelly, et tous les autres).

Je souhaiterais aussi remercier Monsieur Yannick HERVE, pour avoir été un de mes deux rapporteurs, et m'avoir donné le goût de la modélisation. Ayant été un de mes professeurs dans mon école d'ingénieur, j'ai aussi pu profiter de ses compétences et de ses conseils au cours de ma thèse. Je remercie également Monsieur Alain VACHOUX, pour avoir accepté d'être un de mes deux rapporteurs, et de s'être plongé dans ce sujet en émettant des remarques constructives qui m'ont permis d'améliorer mon travail.

Je remercie sincèrement Monsieur Christian PERCEBOIS qui m'a fait le très grand honneur de présider mon jury et de participer à l'amélioration de mes travaux par ses conseils judicieux et sa très grande pédagogie. Je tiens aussi à remercier Monsieur Alain Cazarre et Monsieur Jean Louis Boizard de mon groupe de recherche, d'avoir été des membres de mon jury de thèse. Je remercie également Monsieur Pascal PAMPAGNIN de la société AIRBUS d'avoir illustré l'intérêt d'industriel à mon travail en acceptant d'être membre de mon jury de thèse.

J'aimerais aussi remercier les personnes qui se sont investis avec moi sur les travaux tournant sur la conception système à savoir tout d'abord Juan Carlos Hamon avec qui j'ai eu beaucoup de plaisir de collaborer, mais aussi Rémy Maurice, Hernan Duarte, et Vincent Albert.

Je tiens à remercier tous ceux qui ont permis que ces trois années de thèse soient la meilleure expérience possible : mes collègues de bureau : Maxime, Eric, Alexandre, Joris, Hongwei, Sovann, le mur dédié à Zidane, le PSG,… ; mes collègues du groupe : Gustavo, Pierre, Christophe, Hélène, Carole, Nicole ; mes autres collaborateurs : Mario Paludetto, Alexandre Nketsa, Philippe Pons, Michel Bareille, Régis Pelouse, Luc Leroy. Enfin un remerciement spécial à mes amis Albert, Edu, Sylvain et Amine, ainsi qu'à tous mes collègues footballeurs (Nabil, Aimed, Mehdi, Nicolas, Samuel et tous les autres).

Mes derniers remerciements vont évidemment aux personnes à qui je tiens le plus et qui m'ont soutenu tout au long de mes travaux de thèse. Merci donc à mes parents, à ma sœur Caroline et à mon frère Maxime pour leur présence, leur confiance et leur soutien, et bien sûr un grand merci à celle qui me supporte quotidiennement et qui m'a notamment donné les forces pour finaliser mes travaux de thèse dans la dernière ligne droite.

Table des matières

Liste des Figures

Liste des Tableaux

INTRODUCTION GENERALE

Les Besoins en simulation augmentent et s'élargissent de la conception électronique vers la conception système. Ces besoins impliquent la mise en œuvre de disciplines autres que l'électronique, comme la mécanique, l'optique, la chimie, la biochimie, etc… Ces systèmes multi domaines ouvrent de nouveaux enjeux du fait de leur complexité et de leur hétérogénéité. Ces enjeux pluridisciplinaires, alliés au besoin d'optimiser le processus de conception pour réduire le « Time to Market », conduisent au développement d'approches nouvelles de modélisation et de vérification haut niveau, de modélisation fonctionnelle, de réutilisation et de génération de modules de propriété intellectuelle (IP)… Ces approches nouvelles doivent être considérées dès les premières étapes de la conception.

Les spécialistes de différents domaines doivent travailler en synergie pour assurer la compatibilité entre blocs de natures différentes. Les concepteurs ont besoin d'une nouvelle génération d'outils et de méthodologies qui permettent non seulement d'intégrer des systèmes électroniques incluant du matériel et du logiciel, mais aussi d'autres domaines physiques. Lors de la définition de ces méthodes et de ces processus de conception, une recommandation est de s'appuyer sur des langages et des procédures normalisées afin de mieux résoudre les questions de collaboration.

Une approche générale "Top-Down" est parfaitement illustrée par la conception de l'électronique numérique. Cette approche qui se découpe en quatre étapes principales : Spécifications, Formalisation des exigences, Prototypage virtuel et Réalisation, a d'ores et déjà prouvé qu'elle était extrêmement efficace si l'on détecte au plus tôt les erreurs au cours du cycle de conception. Elle permet par exemple de concevoir des circuits de plus en plus complexes pouvant aller jusqu'à 10^8 composants par module, de nos jours.

Le prototypage virtuel est une autre clé de la conception système actuelle. Il apparaît primordial de pouvoir se baser sur des modèles fonctionnels, exécutables, et échangeables pour vérifier, par simulation tout au long du processus de conception, la conformité au cahier des charges, mais aussi pour optimiser les performances du système et ce, avant d'entamer les démarches de matérialisation et de réalisation technologique. Une clé de ces évolutions est la normalisation du langage de description de matériel (HDL : Hardware Description Language) VHDL-AMS (IEEE 1076.1999) [Vhd99]. La modélisation multi abstractions et multi domaines avec le VHDL-AMS semble, en effet, être particulièrement prometteuse pour ce genre de systèmes hétérogènes complexes.

La tendance actuelle est d'aller au bout de cette logique de modélisation et d'avoir une approche uniquement basée sur les modèles. Ceux-ci sont une représentation d'un objet pouvant être proposée à différents niveaux d'abstraction suivant les objectifs à atteindre. Un modèle peut être développé très tôt dans le processus de conception pour formaliser les spécifications (modèle descriptif de haut niveau). Cette description permet alors d'explorer différentes options d'architectures

et de s'assurer de la bonne adéquation des fonctionnalités spécifiées. Le modèle de haut niveau est nécessaire pour préparer le prototypage virtuel. Il est particulièrement adapté pour avoir de bonnes performances de vitesse de simulation.

Au LAAS-CNRS, le problème de la conception système est étudié depuis déjà quelques années. Les premiers travaux, initiés en 2000, ont permis de concevoir et de proposer une méthode de conception descendante, à savoir la Plateforme HiLeS [Ham05]. Elle consiste notamment en partant du cahier des charges, de faire une représentation fonctionnelle du système en utilisant des diagrammes UML/SysML successifs. Puis, en enrichissant la représentation avec notamment des lois de commande basées sur les réseaux de Petri, nous obtenons une représentation logico-temporelle dans un formalisme spécifique HiLeS [Ham05]. Le développement de cette approche de haut niveau se base sur l'outil HiLeS Designer développé au laboratoire dont la Figure 1 illustre les principales composantes. L'outil s'intègre dans une démarche de conception « Top Down » et débouche sur une plateforme de prototypage virtuel qui permet notamment de générer un code VHDL-AMS du système étudié. On distingue sur la Figure 1 les deux niveaux de description que l'on peut avoir dans HiLeS :

- Une représentation fonctionnelle du système, hors des choix technologiques, qui va notamment permettre de vérifier, très tôt dans le cycle de conception, que les spécifications initiales du cahier des charges sont bien présentes dans la représentation et sont valides.
- Un prototype virtuel du système pour vérifier et valider la fonctionnalité réelle du système une fois les choix technologiques définis.

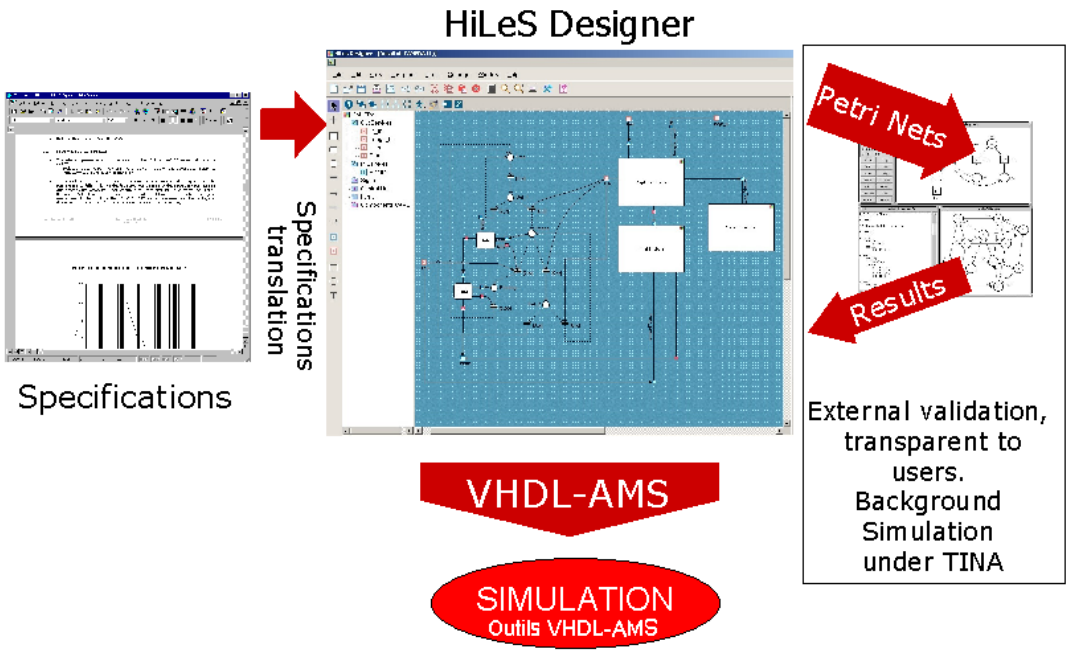


Figure 1 : Plateforme de Prototypage Virtuel HiLeS

Dans ce type de définition du processus de conception, il est nécessaire de prendre en compte les différents concepts et normes d'ores et déjà utilisés et approuvés dans le domaine de l'ingénierie système. Dans notre approche générale, la conception des systèmes est vue comme un assemblage de processus de transformations de modèles.

Ce processus de conception s'inspire des recommandations de l'EIA-632 [Eia98] qui découpe le champ de la conception en treize grands processus interconnectés entre eux qui sont regroupés en cinq catégories (cf. Figure 2) :

- Conception système,
- Techniques de management,
- Acquisition et fourniture,
- Réalisation de produits,
- Evaluation technique.

Ils sont ensuite déclinés en trente trois exigences (cf. Figure 7) qui sont autant de points à vérifier et valider tout au long du processus général de conception. Notre travail s'intègre donc dans ce processus de conception système. Il est aussi fortement lié au processus "évaluation technique" avec notamment les étapes de vérification et la validation des exigences.

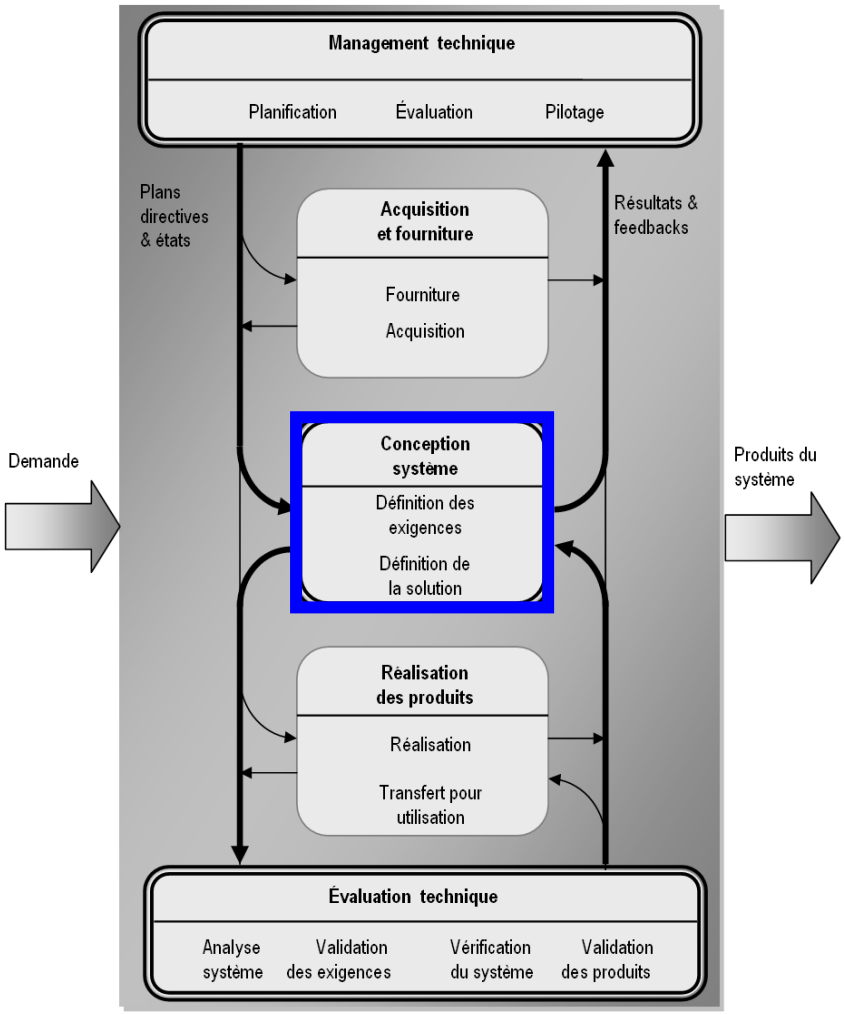


Figure 2 : Démarche globale de développement selon l'EIA-632

La démarche de conception définit dans l'EIA-632 met l'accent sur le traitement des exigences (cf. Cadre bleu sur la Figure 2) qui passe par la création d'une solution logique qui répond aux exigences fonctionnelles du système pour donner lieu à une solution physique intégrant les exigences techniques Le système complexe est décomposé en blocs de construction qui à chaque niveau vont définir les deux types de solution (cf. Figure 3).

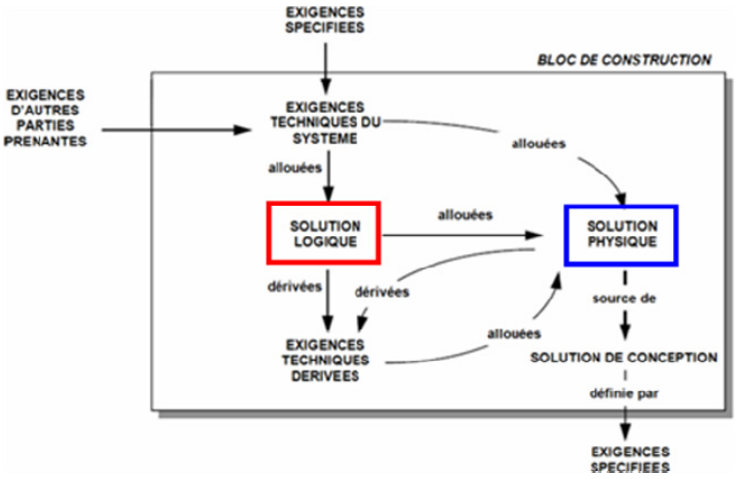


Figure 3 : Bloc de Construction Niveau N

HiLeS, que nous avons présenté précédemment se veut être un candidat pour être le support de cette solution logique [Ham05]. **Dans notre processus de conception nous avons donc besoin d'un support pour la solution physique et d'étudier les moyens pour y converger, c'est ce qui va nous motiver dans cette thèse.**

Comme nous l'avons précédemment dit, le processus détaillé de conception peut être vu comme une succession de transformations de modèles. De nombreux concepts ont été définis afin d'en formaliser la démarche : l'OMG[1] [OMG] a défini le MDA (Model Driven Architecture) [MDA] qui est une application des concepts liés à l'ingénierie des modèles (MDE Model Driven Engineering) pour le domaine du logiciel. L'idée générale est de distinguer les aspects conceptuels du support matériel (plateforme).

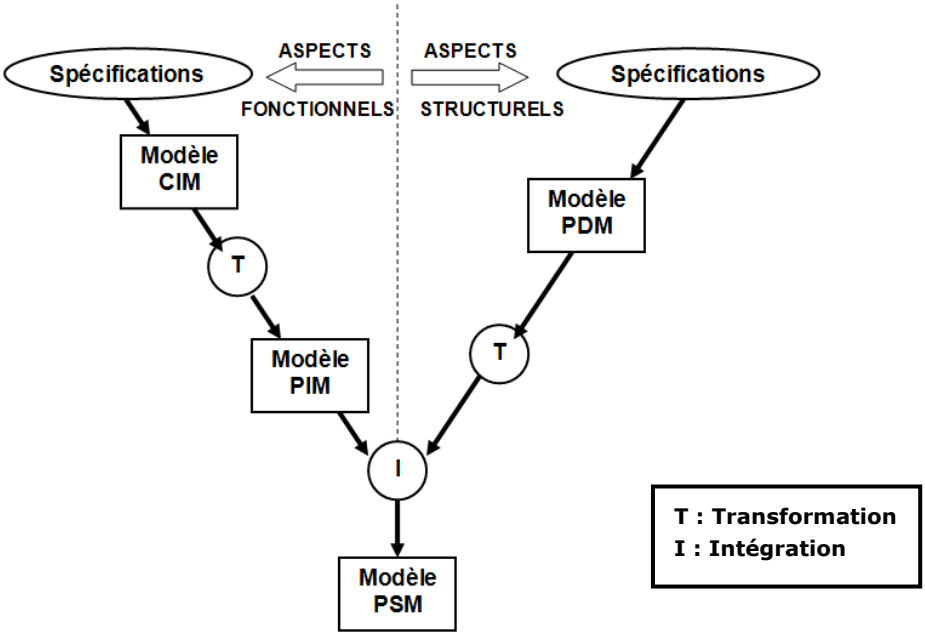


Figure 4 : Processus élémentaire de conception [MDA]

[1] L'OMG™ (Object Management Group) est un organisme international de normalisation. Recommandations : CORBA (Common Object Request Broker Architecture) , UML (Unified Modeling Language), méthode d'analyse et conception d'objet.

Le processus élémentaire de conception est illustré sur la Figure 4 avec :

- CIM (Computation Independent Model), représente le modèle non compilé.
- PIM (Platform Independent Model), représente le modèle indépendant à une quelconque plateforme.
- PDM (Platform Dependent Model), représente un modèle de plateforme.
- PSM (Platform Specific Model), représente le modèle lié à une plateforme particulière.

Le processus de conception complet progresse par des transformations multiples qui peuvent être de différents types (endogène/exogène, vertical/horizontal). Cela sera explicité au cours de ce mémoire (cf. paragraphe 2.1.1).

Cette problématique de transformation de modèle s'intègre complètement dans le sujet traité par cette thèse. Il explore, dans ce cadre, une autre proposition de l'OMG, celui des **meta modèles**, signifiant « modèle de modèle ». Nous nous proposons, afin de répondre à une problématique d'interopérabilité des outils, de baser notre démarche sur ces concepts. Nous proposons d'associer à chaque formalisme (HDL) un meta modèle et d'élaborer les règles de transformation de modèles entre ces meta modèles des langages d'entrée et de sortie. La Figure 4 illustre cette proposition générale avec un schéma de transformation.

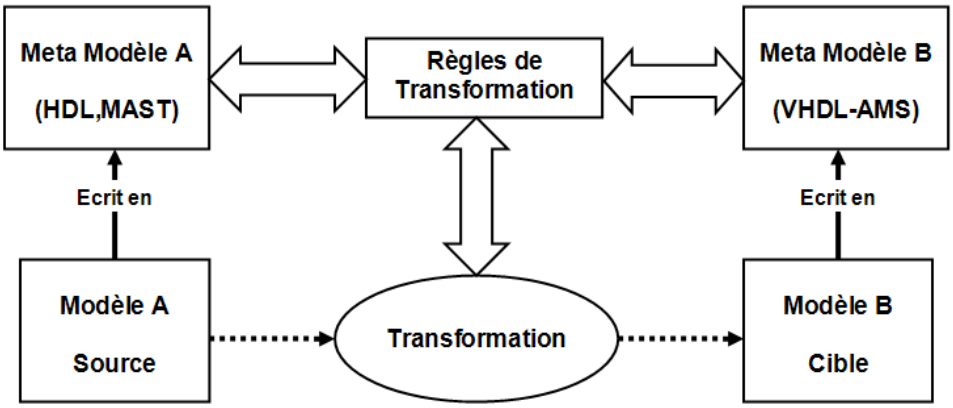


Figure 5 : Proposition de Transformation basée sur les meta modèles

L'attrait évident du prototypage virtuel dans la conception système amène un grand nombre de projets à se développer sur ce thème. Les travaux de recherches sur les langages de description de matériel (HDL) traitent principalement des exemples d'application [Her02] [Coo01], et des méthodes numériques de calcul pour les simulations [Coo01]. **Nous nous positionnons, dans cette thèse, dans une volonté de répondre à l'objectif qui est de faciliter la migration des modèles existants (souvent écrits dans un langage propriétaire) vers la norme (IEEE 1076.1999) VHDL-AMS, et ainsi préserver les investissements recherche qui leur sont arrêtés.**

Quelles sont les étapes et les moyens nécessaires à cette migration ? Quel est le rendement de ce processus ? Ce travail de thèse contribue à apporter une réponse à ces questions.

Nos travaux de recherche ont été effectués au sein du groupe MIS (Microsystèmes et Intégration des Systèmes) du laboratoire de recherche LAAS-CNRS. La mission de ce groupe est notamment de mettre en place des méthodes et des outils pour la conception des micro systèmes. Des collaborations se sont créées entre le LAAS-CNRS et la société MENTOR GRAPHICS pour développer une réflexion

système et proposer une plateforme de prototypage virtuel. Cet effort commun s'inscrit dans une activité de création de modèles VHDL-AMS avec, en perspective, le développement de bibliothèques de modèles **écrits en langage VHDL-AMS comme support du prototypage virtuel**.

Ce mémoire de thèse s'articule autour de quatre chapitres :

Dans le premier chapitre nous présenterons l'importance et le rôle des modèles dans la conception système. Le processus de conception système qui nous anime suivant les recommandations de l'EIA-632 sera explicité. Dans ce contexte de création de modèle qui anime notre thèse, il parait nécessaire de définir au mieux le concept de modèle pour apporter des bases solides aux discussions relatives à leur création et à leur transformation. Ce chapitre nous permettra donc d'aborder le concept de modèle de manière générale, et de se recentrer vers la solution physique qui nous intéresse. Il introduira les transformations que l'on peut faire et les processus qui permettent d'y converger. Le choix du support pour la solution physique sera abordé par une cartographie des langages de modélisation. Un choix de candidat sera effectué avec l'évocation de l'utilisation de la **norme IEEE 1076.1999 VHDL-AMS** en tant que langage de modélisation pour le prototype virtuel. Cela nous permettra d'introduire la problématique de nos travaux qui s'articule plus particulièrement sur la création de modèles en VHDL-AMS.

Le chapitre deux s'attachera plus particulièrement à étudier la transformation de modèle. Nous aborderons tout d'abord les différents types de transformation de modèle qui pourraient répondre à nos besoins pour converger vers la solution physique. Puis nous nous focaliserons dans ce chapitre sur la transformation mettant en jeu le concept de meta modèle qui consiste notamment à créer un modèle pour chaque formalisme mis en jeu. Nous proposerons au cours de ce chapitre **un meta modèle du formalisme VHDL-AMS** pour illustrer ces propos. Nous évoquerons ensuite l'utilisation que l'on peut en faire, avec notamment l'étude de la conformité des modèles créés et la rédaction de règles de transformation au niveau meta modèle.

Le chapitre trois mettra en avant les qualités du VHDL-AMS à être le support de la solution physique et nous étudierons les moyens d'y parvenir. Ainsi, les différentes méthodologies envisagées seront évoquées et illustrées d'exemples. Ces méthodologies concernent les différents cas de figure qui nous concernent comme le passage entre une solution logique et une solution physique en VHDL-AMS au sens de l'EIA-632, mais aussi comme la migration de modèles entre deux langages de description de matériel. Leur évaluation permettra notamment de définir une méthodologie optimisée et adaptée à des cas concrets comme la migration complète d'une bibliothèque de modèle, en alliant rapidité et qualité de transformation. La **modélisation VHDL-AMS d'un système de mise à feu sécurisé à base de composants pyrotechniques** nous permettra de confirmer l'utilisation de ce langage comme support de modélisation des systèmes pluridisciplinaires.

Nous terminerons ce mémoire par un dernier chapitre consacré à l'application de nos méthodologies de transformation de modèle vers le langage VHDL-AMS, à plusieurs **exemples concrets** pris dans les domaines de la recherche et de l'industrie. Cela nous permettra plus particulièrement d'aborder et d'affiner l'étape de validation du modèle qui s'avère critique et qui génère un certain nombre de problèmes à résoudre et de points à préciser, notamment concernant le domaine de validité du modèle et l'attente concernant le modèle cible (cahier des charges à respecter). Enfin l'évaluation de notre approche nous permettra d'en dégager les limites, mais aussi ses perspectives d'évolution.

CHAPITRE 1

1. LES MODELES DANS LA CONCEPTION SYSTEME

L'activité de modélisation est à la base de tout processus de conception. Cette tâche consiste essentiellement à développer une description abstraite d'une réalité physique de telle façon qu'elle soit utile pour le processus de conception. Le modèle dépend du point de vue selon lequel on observe le système, mais aussi suivant l'utilisation que l'on souhaite faire de ce modèle au sein du processus de conception. Ainsi un modèle peut être utilisé pour valider les caractéristiques d'une certaine partie du système ou de tout l'ensemble, au niveau des fonctionnalités ou des performances [PLV05].

La modélisation d'un système simple, monofonctionnel, consiste à définir une représentation de son fonctionnement dans son environnement d'utilisation. Cette représentation peut techniquement être textuelle ou graphique, analytique ou fonctionnelle. Le niveau de détail peut être varié ainsi que le niveau d'abstraction.

La définition d'un modèle est nécessaire afin d'éviter les confusions et les incompréhensions sur l'attente que les différents collaborateurs du projet ont sur ce modèle. Ainsi, on dit :

- Que **le modèle est physique** si son écriture s'appuie sur la physique des mécanismes mis en jeu,
- Que **le modèle est comportemental** lorsque la loi ou le tableau numérique décrivant le matériel est déduit de résultats expérimentaux mesurés sur le système réel : en automatique on parle d'identification de modèles. Ce type de modèle est descriptif, avec un champ d'application limité au delà duquel il n'est plus valide et ne permet pas d'optimiser le système.

Le choix d'un modèle, même lorsqu'il s'agit d'un système simple, n'est pas sans difficulté puisque doivent être définis la précision, le domaine de validité et le point de vue selon lequel on souhaite approcher le système. On doit évaluer, lorsqu'il est non linéaire, les difficultés qui peuvent apparaître dans son exploitation pouvant conduire à des aberrations.

Il faut retenir que le modèle fixe la relation entrée-sortie, c'est-à-dire la (les) fonction(s) entrée-sortie. Dans la réalité, dans les systèmes complexes actuels, la relation est multifonctionnelle ce qui suppose le développement de méthodes et d'outils eux-mêmes extrêmement complexes :

i) L'approche par la physique est le plus souvent inopérante sinon pour donner des éléments de principe qui peuvent être des guides utiles. La

raison est que la mise en œuvre des lois physiques dans un environnement réel est très délicate et conduit à des systèmes d'équations très vite inextricables. Il faut toutefois constater que l'effort de la physique s'organise dans ce sens. Des exemples existent en météorologie, en pharmacologie, en modélisation atomistique des matériaux… Cette stratégie s'intègre alors dans une approche de **modélisation multi-échelles** entre des modèles physiques très sophistiqués et très friands en temps de calcul, et des modèles comportementaux, simples mais limités en performances notamment prédictives…

ii) L'approche par l'expérimentation est plus immédiatement efficace car elle peut apporter, dans un domaine d'utilisation restreint, des modèles utilisables. Il faut toutefois rester très attentif à la qualité de modélisation qui peut s'altérer lorsque l'on s'éloigne des domaines référents où le modèle a été établi et lorsque l'on multiplie des modèles de ce type en interaction sans pouvoir connaître les aberrations qui peuvent en découler (degré de confiance, domaine de validité…).

Ces questions de modélisation sont au cœur d'un problème très difficile, celui de la maîtrise de la complexité des systèmes artificiels, au sens des systèmes que l'on est déjà amené à concevoir, à produire et à diffuser. L'Aéronautique, les Transports, l'Espace, les Systèmes Télécoms, les Machines et l'Instrumentation…, sont des exemples de cette complexité permise par les développements fondamentaux des matériaux et de l'Electronique. **La stratégie qui se développe rapidement aujourd'hui est d'amorcer et d'accompagner la création du système réel par la création parallèle d'un simulateur informatique de ce même système**. On peut ainsi :

- Anticiper et accélérer le processus de conception et de production du Système.
- Maîtriser et optimiser le Système tout au long de son cycle de vie.

Souvent, le compromis type est d'atteindre un maximum de fiabilité dans le processus de conception, avec un minimum de coût et de temps de conception. On doit s'assurer que les exigences sont clairement spécifiées et comprises de tous. Une des plus grandes causes d'un coût excessif est l'obligation de révision d'un système après sa fabrication pour en corriger les erreurs. En évitant ces extrêmes et en utilisant des outils améliorés pour le processus de conception, il est évident que les coûts et les retards peuvent être maîtrisés [TAP03].

Développer une Simulation Système n'est possible et crédible que par la disponibilité des modèles représentatifs mais ce n'est pas la seule difficulté : En pratique, la modélisation est portée par un algorithme informatique spécifique soit lié à l'origine « métier » de ce modèle soit simplement au choix du « langage » dans lequel il a été écrit. On voit donc qu'une bonne modélisation implique non seulement une connaissance de la physique des mécanismes, une connaissance des méthodes et des outils de l'identification comportementale, mais aussi une connaissance informatique très approfondie des langages et des formalismes de description.

Cela peut donc définir une **ingénierie de la modélisation liée à l'ingénierie système** qui permettrait de définir la méthodologie d'élaboration d'un modèle, sa validation et son intégration au sein d'une modélisation système.

1.1 Les Modèles

Avec les récentes tendances vers les développements guidés par les modèles (MDA : Model Driven Architecture cf. 1.2), il est communément acquis que la modélisation et les modèles sont une problématique centrale de la conception. Afin d'éviter tout problème de compréhension et de communication entre les personnes, il est nécessaire de définir d'abord la notion de modèle afin que les résultats obtenus soient conformes aux attentes de chacun [Küh05].

Dans l'ingénierie logicielle, un modèle est traditionnellement référencé à un artefact formulé dans un langage de modélisation, qui décrit un système. D'après Stachowiak [Sta73], un modèle doit posséder les trois propriétés suivantes :

- Un modèle doit être basé sur son original (« mapping feature »),
- Un modèle peut refléter seulement une partie pertinente des propriétés de l'original (« reduction feature »),
- Un modèle doit pouvoir être utilisé à la place de l'original tout en restant sur les mêmes objectifs (« pragmatic feature »).

On retrouve dans les deux premières propriétés la notion de « projection ». Certaines informations sont perdues au cours de la projection, mais ce qui est retenu est lié à ce pourquoi le modèle va être utilisé. La troisième propriété indique que l'on fait, avec un modèle, une représentation exacte de seulement certaines propriétés que l'on souhaite étudier du système original.

Il est communément accepté que les modèles soient caractérisés en deux catégories générales : les modèles descriptifs et les modèles prédictifs. On trouve dans la littérature des appellations différentes (les modèles de représentation opposés aux modèles de simulation, les modèles jeton (« token ») opposé aux modèles type (« type ») [Küh05]) mais les concepts restent les mêmes. Ainsi :

- Un modèle descriptif doit convertir les entrées aux sorties correspondantes suivant la même dépendance que le système réel. Le modèle peut être le résultat d'une identification ou d'une retranscription comportementale. Par exemple, un modèle qui retranscrit un système en utilisant des règles logiques et/ou mathématiques (exemple approximation par des polynômes) sans se référer à la structure interne du système réel, se classe dans les modèles descriptifs.

- Un modèle prédictif, quant à lui, va s'appuyer sur la structure détaillée et les fonctionnalités réelles du système à modéliser pour avoir ce caractère prédictif, dans un domaine précis de simulation.

Chacun de ces deux types de modèle peut correspondre à un des types suivants :

- Les modèles mathématiques : Ils vont utiliser des expressions mathématiques (ou des méthodes numériques) pour représenter les fonctions qui lient les entrées et les sorties du système. Ces modèles mathématiques peuvent être :
    - Déterministe opposé à Stochastique : dans les modèles déterministes, l'ensemble des variables est déterminé par les paramètres du modèle et l'état précédent de ces variables. Ainsi pour un cas particulier de conditions initiales, le modèle déterministe va se comporter de la même façon à chaque fois. Inversement un modèle stochastique ou statistique, est basé sur le caractère aléatoire de

certaines variables qui peuvent suivre une distribution statistique [Mcl99].
    - Statique opposé à Dynamique : un modèle statique ne prend pas en compte le paramètre « temps ».
    - Linéaire opposé à non linéaire : un modèle mathématique est généralement composé de variables qui sont des abstractions de quantités présentes dans le système décrit, et d'opérandes qui agissent sur ces variables. Elles peuvent être des opérandes algébriques, de fonctions ou d'opérandes différentiels. Si tous les opérandes sont linéaires, alors le modèle est considéré comme linéaire, non linéaire sinon.

- Les **modèles physiques** : ces modèles sont bâtis sur les lois de la Physique.

- Les modèles graphiques : ces modèles représentent certains attributs du système par des éléments graphiques. Les propriétés géométriques du graphique permettent de représenter les relations logiques ou les processus du système étudié.

Au cours du processus de conception d'un système hétérogène, outre les contraintes liées aux différents domaines physiques mis en jeu, on doit aussi prendre en compte les contraintes liées au problème d'échelle. La Figure 5 décrit les variations d'échelles que l'on peut rencontrer dans la modélisation d'un système.

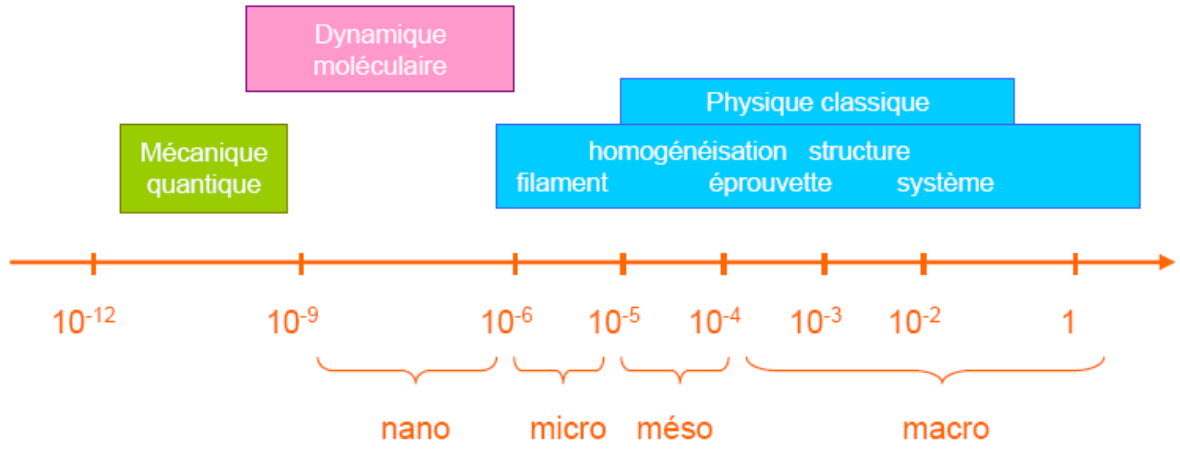


Figure 6 : Modélisation Multi-échelles

L'approche **multi échelle** est intéressante car elle permet de donner de la cohérence et de la prédictibilité aux modèles que l'on utilise en pratique. Cette aptitude peut être mise à profit selon des objectifs complémentaires :

- Cas où le système existe déjà, il fonctionne mais présente certaines anomalies dont on veut connaître les causes. Le recours à la modélisation multi-échelle, via la traçabilité des paramètres au sein des différents niveaux de modélisation, va permettre de faire le lien entre l'anomalie et son origine profonde dans le système.

- Cas où le système est en cours de conception et l'on souhaite connaître l'impact d'un choix technologique sur ses caractéristiques à tous les niveaux. Le recours à la modélisation multi-échelle, via la traçabilité, va permettre d'en établir les liaisons.

Ces exigences de traçabilité des modèles sont aussi présentes en pratique, dans ce que l'on appelle la ré-utilisation (re-use) :

- Ré-utilisation de modèles déjà éprouvés par des travaux antérieurs (capitalisation des acquis).
- Ré-utilisation des composants déjà validés par des réalisations antérieures.

Dans la démarche de conception, ces procédés de réutilisation sont systématiques et pas toujours parfaitement identifiés. Au-delà, des démarches méthodologiques formalisées, il y a tout l'informel de l'expérience d'ingénieur et de l'expertise qui intervient à tout instant. Il convient donc de caractériser un maximum le modèle créé (domaine d'utilisation, fonctionnalités) afin qu'il puisse être retrouvé rapidement (cf. paragraphe sur les bases de données 1.3.3) et réutilisé dans les conditions pour lesquelles il a été créé.

1.2 Les Transformations de Modèles

Dans notre cadre de conception guidé par les modèles, une fois les modèles en tant que tel définis, il convient d'aborder l'autre constituant essentiel de ce type de démarche : les transformations de modèles.

Le modèle est un « constituant vivant » de la conception système : il est toujours le même et toujours différent en fonction de l'utilisation que l'on veut en faire dans la démarche générale de conception :

- Il peut être enrichi par des nouvelles données concernant sa structure et son environnement d'utilisation,
- Il peut être simplifié pour des raisons techniques de vitesse de calcul, d'exigence temps réel, ou de restriction du domaine de validité,
- Il peut être réécrit dans un environnement informatique différent (changement de langage).

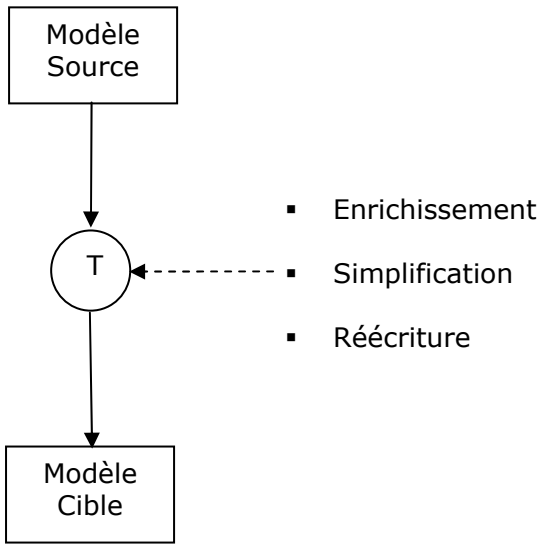


Figure 7 : Processus élémentaire de Transformation de Modèle

Dans cette optique de transformation, le modèle original que l'on appelle source va alors subir une transformation qui donnera lieu à un nouveau modèle appelé cible et qui sera une version "enrichie", ou "simplifiée", ou "réécrite" du modèle source (cf. Figure 6).

Les informaticiens sont les premiers à s'être intéressé à ce concept de transformation des modèles. C'est en effet dans le domaine de la conception de système logiciel que l'on a les premières applications et les premiers outils mettant en jeu la transformation de modèle. Ainsi, l'OMG [OMG] a appliqué l'ingénierie guidée par les modèles aux systèmes logiciels, en proposant une approche de conception de logiciel appelée "architecture guidée par les modèles" (MDA [MDA]), officiellement lancée en 2001.

Le processus MDA, présenté dans l'introduction générale (cf. Figure 3) inclut les quatre principaux modèles que je reprécise dans ce paragraphe :

- CIM : Computation Independent Model
- PIM : Platform Independent Model : Les fonctionnalités du système y sont définies via un langage spécifique à un domaine d'utilisation (DSL).
- PDM : Platform Dependent Model : Modèle décrivant une plateforme (CORBA [Cor], HiLeS [Ham05],…).
- PSM : Platform Specific Model : Modèle correspondant à l'implémentation des fonctionnalités sur une plateforme spécifique.

La transformation de modèle entre le modèle PIM et un ou plusieurs PSMs est faite par des outils en conformité avec la norme de l'OMG QVT [OMG] qui est un langage normalisé pour exprimer les transformations de modèles (QVT : Query View Transformation). L'implémentation de QVT se retrouve notamment dans le langage de transformation ATL [ATL] (ATLAS Transformation) qui permet, par exemple, au sein de la plateforme d'Outils ECLIPSE [Ecl], de réaliser des transformations.

Transformer un modèle, c'est apporter un changement à l'un ou l'autre de ses constituants d'origine pour formuler une nouvelle écriture. Du point de vue informatique cela suppose d'appliquer des règles de transformation rigoureusement.

Les différents types de transformations seront évoqués dans le deuxième chapitre, notamment dans l'objectif d'appliquer les différents concepts à notre problématique. Comme évoqué au cours de l'introduction générale, nous comptons mettre en œuvre cette stratégie de transformation de modèle au sein de notre processus de conception système en définissant des règles de transformation au niveau méta modèle, comme décrit sur la Figure 4.

1.3 La Conception Système vue comme un Assemblage de Processus et de Transformations de Modèles

Un processus est par définition une suite d'opérations élémentaires réalisées dans un but précis. Dans notre approche de modélisation système, comme évoqué dans le paragraphe précédent, la transformation de modèle va jouer un rôle pivot primordial : la transformation de modèle sera considérée comme un processus élémentaire, liant un modèle source et un modèle cible. En revenant à la définition d'un processus, nous allons considérer qu'un processus peut être représenté par une série de transformations de modèle d'objets très diverses :

- Processus d'enrichissement d'un modèle élémentaire.
- Processus de simplification d'un modèle complexe.
- Processus de traduction d'un modèle vers un nouveau langage.

Un processus de modélisation vise à l'obtention d'un modèle terminal répondant aux besoins de simulation et de vérification. Le développement d'un système résulte donc de la mise en œuvre de nombreuses étapes de modélisation interconnectées qui, fondées sur les exigences du cahier des charges, convergent vers le prototypage virtuel puis réel du système lui-même. Cette organisation en étapes multiples a fait naturellement l'objet d'importants travaux industriels et académiques, fondations de l'ingénierie système… **Concernant notre problématique, les recommandations de l'EIA-632 (cf. Figure 2) fixent le cadre de notre approche globale de conception système** :

Treize grands processus interconnectés sont définis par l'EIA-632 pour le développement système et sont répartis en cinq catégories :

- Gestion technique (Technical management)
- Acquisition and supply
- System design
- Product realisation
- Technical evaluation

Ces processus sont détaillés en trente trois exigences (cf. Figure 7).

SUPPLY PROCESS REQUIREMENTS
1—Product Supply

ACQUISITION PROCESS REQUIREMENTS
2—Product Acquisition
3—Supplier Performance

PLANNING PROCESS REQUIREMENTS
4—Process Implementation Strategy
5—Technical Effort Definition
6—Schedule and Organization
7—Technical Plans
8—Work Directives

ASSESSMENT PROCESS REQUIREMENTS
9—Progress Against Plans and Schedules
10—Progress Against Requirements
11—Technical Reviews

CONTROL PROCESS REQUIREMENTS
12—Outcomes Management
13—Information Dissemination

REQUIREMENTS DEFINITION PROCESS REQUIREMENTS
14—Acquirer Requirements
15—Other Stakeholder Requirements
16—System Technical Requirements

SOLUTION DEFINITION PROCESS REQUIREMENTS
17—Logical Solution Representations
18—Physical Solution Representations
19—Specified Requirements

IMPLEMENTATION PROCESS REQUIREMENTS
20—Implementation

TRANSITION TO USE PROCESS REQUIREMENTS
21—Transition to Use

SYSTEMS ANALYSIS PROCESS REQUIREMENTS
22—Effectiveness Analysis
23—Tradeoff Analysis
24—Risk Analysis

REQUIREMENTS VALIDATION PROCESS REQUIREMENTS
25—Statements Validation
26—Acquirer Requirements Validation
27—Other Stakeholder Requirements Validation
28—System Technical Requirements Validation
29—Logical Solution Representations Validation

SYSTEM VERIFICATION PROCESS REQUIREMENTS
30—Design Solution Verification
31—End Product Verification
32—Enabling Product Readiness

END PRODUCTS VALIDATION PROCESS REQUIREMENTS
33—End Products Validation

Figure 8 : Les Exigences de la conception système : Norme EIA-632

Partant du cahier des charges, le développement système va devoir identifier et mettre en œuvre de nombreux processus interconnectés organisés de telle sorte qu'ils correspondent à une organisation des processus couvrant l'ensemble du projet :

- Chaque entité opérationnelle à l'intérieur, des grands processus et des 33 exigences, gère son modèle d'activité qui intègre toutes les données et modèles intermédiaires utiles.

- Chaque entité assure le travail de coordination des décisions partielles prises par rapport à des critères fixés, en tenant compte du projet général de développement système.

Les données multiples, informations, recommandations du cahier des charges sont proprement formalisées et rendues en modèles intermédiaires, en modèles intégrés et finalement en modèles complets couvrant l'activité système.

Notre problématique de conception système se place évidemment dans ce processus général de développement système et requiert aussi de reboucler vers certains sous processus du processus général d'évaluation technique contenant notamment les étapes de validation et de vérification.

1.3.1 Les couplages en Y de processus

Dans cette vision de multi processus interconnectés définissant le processus général de conception, il convient de représenter le couplage entre sous processus. Nous proposons de faire une représentation en Y de ce couplage en restant conforme à l'esprit de la transformation de modèles.

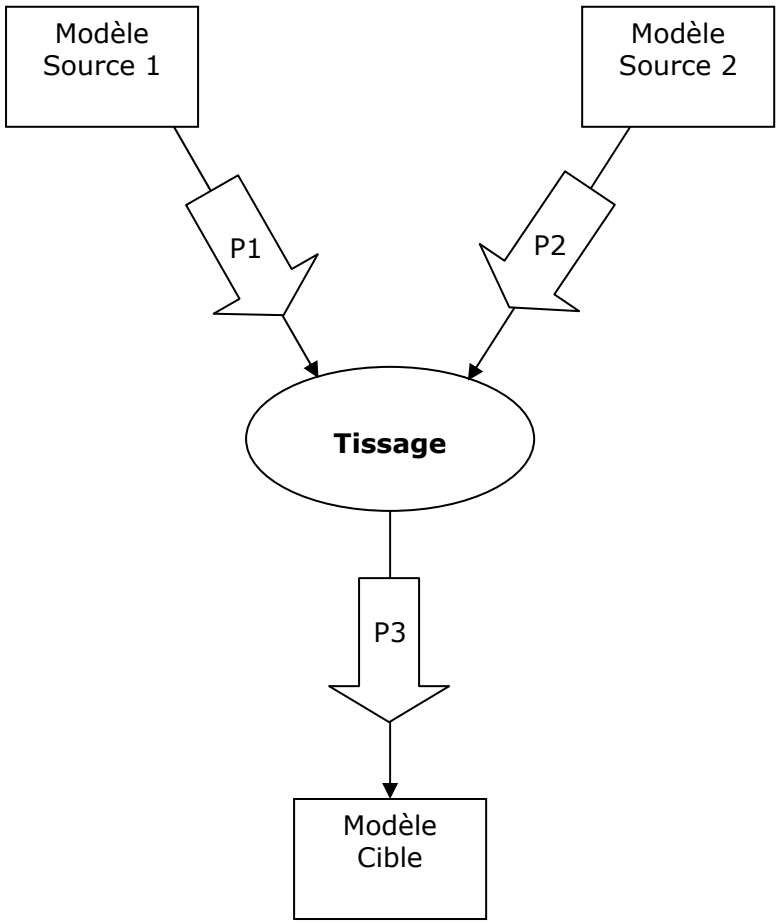


Figure 9 : Couplage de Y de deux Processus

La Figure 8 illustre le couplage de processus en Y. Cette représentation indique que le processus P3 démarre avec un modèle qui résulte de l'intégration de modèles obtenus au terme des processus P1 et P2. On peut parler de tissage entre processus pour détailler les opérations d'échange entre les processus.

Dans notre problématique, on peut illustrer ce couplage de processus dans plusieurs exemples énumérés ci-dessous :

Exemple 1 :

- P1 : représentation de la partie analogique du système
- P2 : représentation de la partie numérique du système
- P3 : modèle mixte du système

Exemple 2 :

- P1 : représentation fonctionnelle du système
- P2 : représentation organique du système
- P3 : modèle en bloc structuré du système

Exemple 3 :

- P1 : représentation en tâches à effectuer
- P2 : représentation des moyens disponibles
- P3 : modèle des plannings

Ce couplage peut prendre diverses formes opérationnelles :

1. La définition de nouvelles entités (variables agrégées) construites à partir des deux modèles suivant des règles de couplage, de partitionnement. Ces nouvelles variables sont nécessairement des variables agrégées dépendantes des deux modèles en amont dont elles sont issues.

2. La co-simulation de modèles indépendants, éventuellement par le biais de la définition d'un langage commun nouveau permettant une simulation.

1.3.2 Le rôle de la co-simulation

Avec la croissante complexité des systèmes dans des domaines comme l'automobile et l'aéronautique, l'intégration de logiciels et de matériels devient une problématique de plus en plus difficile à résoudre. La conception de systèmes complexes requiert souvent la coopération de plusieurs équipes avec des domaines d'expertise différents mettant en jeu des outils de développement et de validation variés. Il devient maintenant nécessaire de pouvoir valider l'ensemble du comportement d'un système très tôt dans son cycle de conception. Cela amène donc à des problèmes survenant lors de l'intégration au sein de la modélisation du système, par plusieurs modèles écrits dans des langages de modélisation différents et utilisant des outils de simulation différents.

Dans notre optique de modélisation de systèmes hétérogènes, nous souhaitons favoriser l'utilisation d'un seul et unique langage, et un seul et unique outil de simulation. La transformation de modèle est alors une étape inévitable si certains modèles provenant de certains domaines d'application sont écrits dans des langages de modélisations différents. Cependant, il existe aussi d'autres alternatives qui peuvent s'avérer, suivant les cas, plus rapides à mettre en place et plus adaptées pour répondre à certaines contraintes liées à la conception système comme la contrainte temporelle du "Time To Market" [Her02].

1.3.2.1 La cosimulation Système

La cosimulation permet de simuler un système complet, tout en ayant des modèles et des outils de simulation différents : L'étape de transformation de modèle n'est alors plus indispensable.

Les outils et les langages sont de plus souvent très adaptés et spécifiques à leur domaine d'application. Ils sont généralement développés et utilisés par les mêmes personnes et répondent ainsi parfaitement à leurs besoins. La cosimulation permet de garder intacts ces environnements, mais nécessite par contre le développement d'interface entre les outils, entre les modèles, mais aussi de créer des modules de conversion et d'interfaçage.

Il arrive souvent que soient intégrées, par défaut, dans certains outils des interfaces possibles avec d'autres outils. Par exemple le logiciel SABER [Sab] autorise une cosimulation avec les logiciels MATLAB [Mat] et ModelSim [Mod]. Cette solution permet de profiter des performances des outils dans chacun de leurs domaines d'application. Ainsi, dans le cas d'une cosimulation SABER – MODELSIM, l'intérêt est de profiter des nombreux modèles numériques et du cœur de simulation performant de MODELSIM. Cependant, cette solution reste assez lourde à mettre en place et est moins performante en terme de temps de simulation et de précision des résultats qu'une solution avec un unique outil de simulation.

Le développement d'interface entre outils peut aussi être réalisé par l'utilisateur mais cela dépend des possibilités prévues par chacun des éditeurs des outils. En général cela se traduit par la présence ou non d'API (Application Programming Interface), accessibles par l'utilisateur du logiciel avec ou sans accord tacite de l'éditeur de l'outil.

En général, la cosimulation s'effectue entre deux simulateurs de façon point à point. Néanmoins il existe d'autres solutions qui fonctionnent grâce à la mise en place d'un bus de cosimulation et d'une architecture ouverte permettant la communication entre plusieurs simulateurs hétérogènes. Il n'est pas nécessaire d'avoir tout le matériel en local, le bus de cosimulation peut très bien passer par le réseau. On peut ainsi optimiser les ressources CPU et améliorer les temps de simulation en partitionnant le système en vue de la simulation sur plusieurs machines (cf. Figure 9).

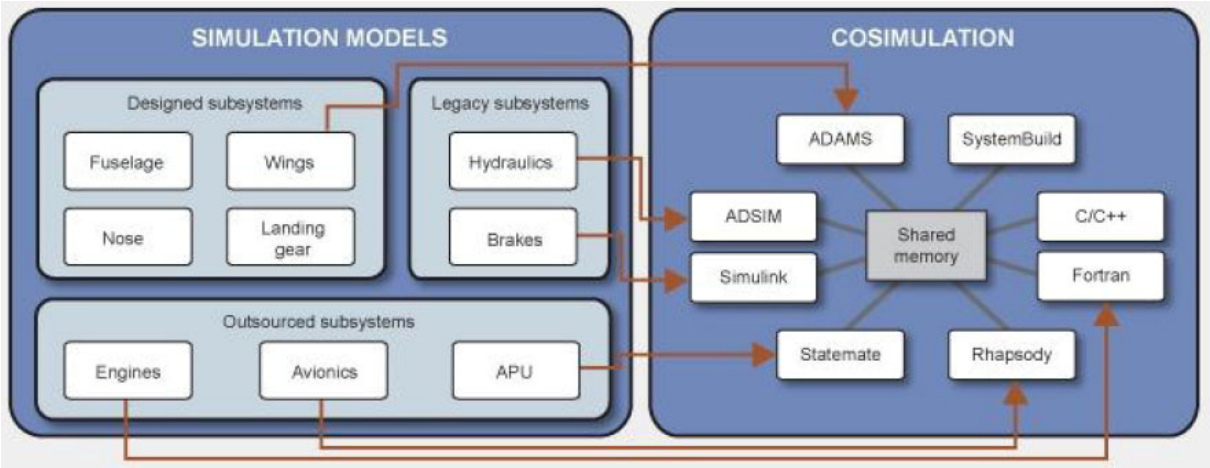


Figure 10 : Partitionnement du Système sur plusieurs outils en vue de la simulation (Source [Cos06])

Pour illustrer ce type de cosimulation, nous pouvons citer la technologie de cosimulation SVX utilisée par la société MENTOR GRAPHICS notamment dans le logiciel SystemVision [Sys50] pour effectuer des cosimulation avec le logiciel MatLab Simulink [Mat] (cf.Figure 10), mais aussi pour partitionner une modélisation VHDL-AMS d'un système entre plusieurs ordinateurs et outils SystemVision et ainsi améliorer les temps de simulation.

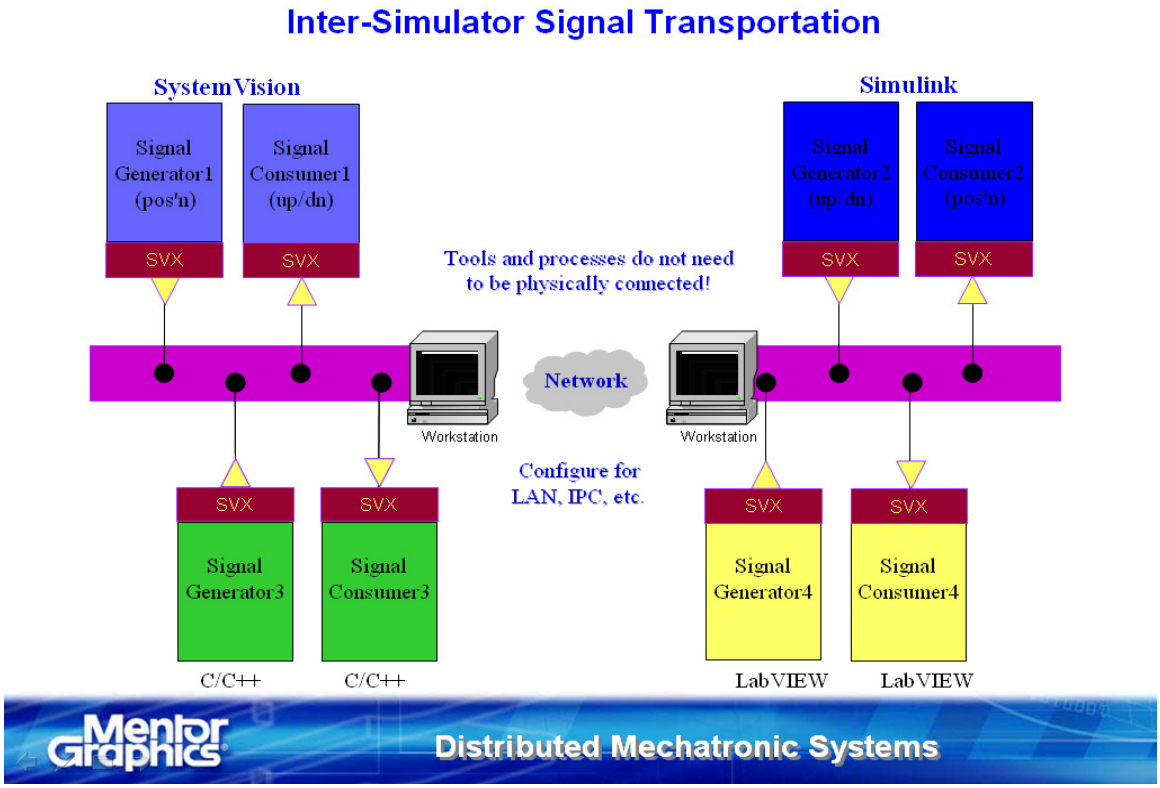


Figure 11 : Technologie de Cosimulation SVX

Pour la même problématique de cosimulation, nous pouvons aussi citer la solution proposée par la société CHIASTEK qui a aussi ce type de service basé sur un bus de cosimulation dans son outil COSIMATE [Cos06] (cf.Figure 11).

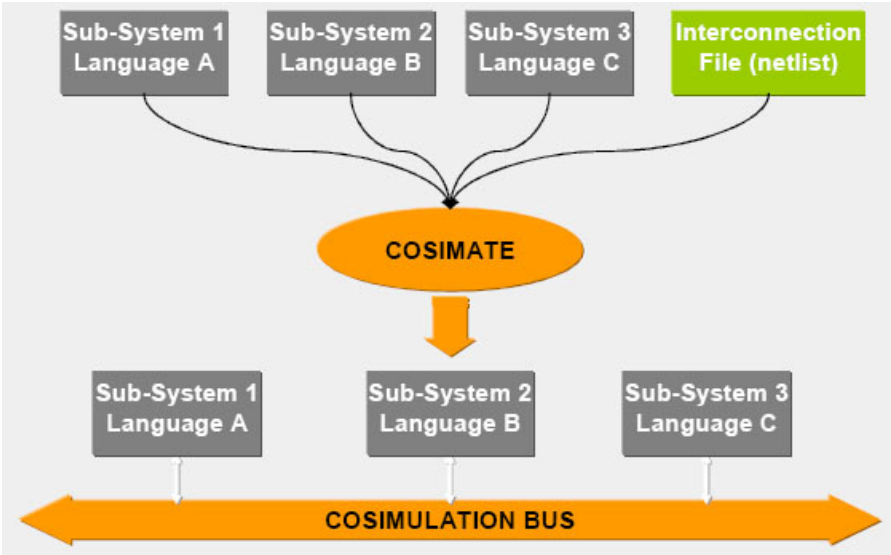


Figure 12 : COSIMATE Cosimulation

1.3.2.2 Cosimulation matériel-logiciel

Nous nous focalisons dans cette partie sur la cosimulation dans le cadre de la validation logicielle. Dans le flot de conception classique, le logiciel embarqué est, en général, validé tardivement dû à la nécessité de disposer du modèle matériel réel du processeur embarqué. L'intérêt est alors de pouvoir exécuter conjointement logiciel et matériel au cours d'une simulation. J.Rowson propose deux définitions de la cosimulation [Row94] :

- Une cosimulation de bas niveau qui fait intervenir la forme assembleur (binaire) du logiciel embarqué,
- une cosimulation de haut niveau qui ne nécessite que le modèle écrit en C du logiciel embarqué. Avec l'application écrite en C, la lecture et le débogage fonctionnel sont bien plus aisés qu'avec le code binaire. Il n'est plus nécessaire de réaliser l'architecture du processeur, ni le jeu d'instruction. Par contre seule la validation fonctionnelle de l'algorithme est effectuée.

On peut voir sur la Figure 12, l'enchaînement des étapes aboutissant à la validation complète du logiciel et à l'intégration du processeur dans le système (cf. [NVH00]).

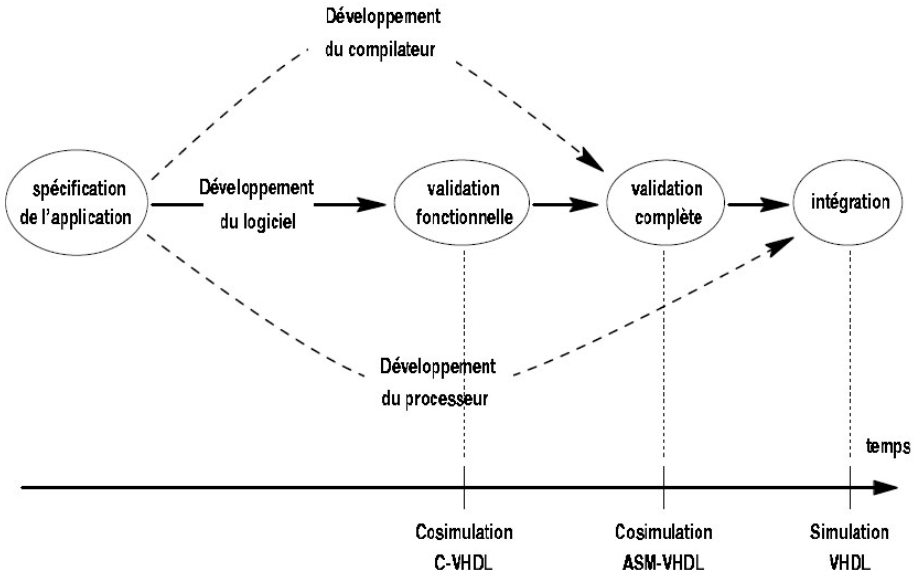


Figure 13 : Etapes de validation du logiciel dans le cadre d'une cosimulation C-VHDL (Source [NVH00])

Dans ce cadre de cosimulation matériel-logiciel au niveau assembleur, il existe par exemple l'outil Seamless de MENTOR GRAPHICS [Sea] qui fournit une interface de co-simulation pour connecter un simulateur de jeu d'instructions à un simulateur matériel (VHDL, Verilog, System Verilog, System C).

La co-simulation est donc une alternative possible qui peut s'avérer plus adaptée à la situation lorsque les moyens, en terme de temps de conception et de ressources disponibles (modèles, outils, expertises des concepteurs), le permettent. C'est une possibilité à intégrer dans notre démarche de conception, même si notre problématique principale est de pouvoir modéliser un système hétérogène avec un seul et même langage de modélisation.

1.3.3 Les Bibliothèques de modèles

Avec des délais de conception de plus en plus courts et l'augmentation constante de la complexité des systèmes à modéliser, il est essentiel d'optimiser au maximum les méthodes de travail. Il est notamment nécessaire d'intégrer dans le flot de conception le concept de réutilisation « reuse » notamment au niveau des modèles créés qui s'ils présentent un caractère générique et sont correctement documentés, pourront être réutilisés dans un autre projet et contexte. Les modèles créés et validés doivent être sauvegardés de manière à être retrouvés et réutilisés facilement dans un autre contexte.

La sauvegarde de modèles validés notamment sous la forme de bibliothèque de modèles, est une problématique que l'on retrouve au sein des outils de conception et de simulation. Ces outils permettent bien souvent de créer un modèle utilisateur et de le stocker dans une de leurs bases données, mais ils proposent aussi un certain nombre de modèles d'ores et déjà créés pouvant être réutilisés par le concepteur en tant que brique de base par exemple dans son nouveau projet. Ces modèles sont en général regroupés au sein de bibliothèques de modèles pouvant être de plusieurs formes. Certains modèles sont accessibles par exemple par le biais de symbole à placer sur une schématique, avec un lien vers un code source, cela s'apparente à une bibliothèque de symbole. La forme sous laquelle le modèle est sauvegardé peut aussi varier. Ainsi le modèle peut aussi être sauvegardé non seulement sous la forme de code source mais par exemple sous la forme de modèle compilé. L'instanciation de ce modèle au sein d'un modèle structurel ne requerra pas une nouvelle compilation de celui-ci lors de la compilation du modèle structurel dans lequel il est réutilisé. Cela permet donc un gain de temps au niveau de la compilation en plus de la réutilisabilité.

Notre problématique de création de modélisation de systèmes hétérogènes, va nous amener à devoir utiliser une base de donnée une fois notre modèle validé. Il convient donc de s'intéresser à la structure de base de données dans le cadre de la conception système. Le concept de base de données a été développé depuis plusieurs années toujours dans l'optique de pouvoir accéder facilement et rapidement à l'information que l'on souhaite. Nous pouvons citer différents types de base de données :

- Bases de données hiérarchiques : les données y sont stockées sous la forme d'une arborescence. Les relations entre les données sont de type 1 vers N.
- Bases de données réseaux : c'est une amélioration des bases hiérarchiques permettant notamment d'établir des relations entre toutes les données et non plus entre un niveau N et N – 1.
- Bases de données relationnelles : les données sont ici stockées dans des tables appelées aussi relations.
- Bases de données orientées objet : les données sont représentées sous la forme d'objets.
- Bases de données XML : Ces bases de données s'appuient sur la structure du langage XML (eXtensible Markup Language : Langage de Balisage Extensible) pour stocker les données et les repérer. L'organisation des données reste cependant hiérarchique.

Chaque fichier XML doit correspondre à une grammaire précise, qui notamment définit les différentes balises utilisées au sein du fichier pour organiser les informations. Cette grammaire peut être définie au sein d'un document de définition (DTD : Document Type Definition) ou bien sous la forme d'un autre fichier XML avec un langage de description de document XML (XSD : XML Schema Description).

Ces techniques apparaissent très adaptées pour le stockage d'informations liées à un modèle. Nous pouvons notamment citer plusieurs outils mettant en jeu cela. Ainsi dans un logiciel comme Paragon [PAR04], dont l'objectif est de mettre à disposition un environnement de conception de modèles, tous les modèles créés sont stockés sous la

forme de fichiers XML conforme à une DTD. Elle est représentée sur la Figure 13. On peut y voir les différentes balises définies qui ici vont permettre de stocker sous la forme d'un fichier XML les informations (paramètres génériques, ports entrée/sortie, variables, équations) relatives à un modèle de système ici analogique.

```
<!ELEMENT MODEL (INTERFACE, BODY)>
<!ATTLIST MODEL NAME CDATA #REQUIRED>
<!ATTLIST MODEL COMMENTS CDATA #IMPLIED>
<!ELEMENT  INTERFACE (PARAMETER, PORT)>
<!ATTLIST PARAMETER name CDATA #REQUIRED>
<!ATTLIST PARAMETER value CDATA #IMPLIED>
<!ATTLIST PARAMETER type  (real|integer|time) #REQUIRED >
<!ATTLIST PARAMETER unit CDATA #IMPLIED>
<!ATTLIST PARAMETER range CDATA #IMPLIED>
<!ATTLIST PORT name CDATA #REQUIRED>
<!ATTLIST PORT mode (in|out|inout) #REQUIRED>
<!ATTLIST PORT nature (electrical|mechanical|thermal|optical|magnetic)
#REQUIRED>
<!ATTLIST PORT type (terminal|quantity|signal) #REQUIRED>
<!ELEMENT BODY (MODEL_EXPRESSIONS, BRANCH,MACROMODEL)>
<!ELEMENT MODEL_EXPRESSIONS (math_dtd)>
<!ELEMENT BRANCH (QUANTITY, EQUATION)>
<!ATTLIST BRANCH name CDATA #REQUIRED>
<!ATTLIST BRANCH from CDATA #REQUIRED>
<!ATTLIST BRANCH to CDATA #REQUIRED>
<!ATTLIST QUANTITY name CDATA #REQUIRED>
<!ATTLIST QUANTITY nature (through|across) >
<!ATTLIST QUANTITY type CDATA #IMPLIED >
<!ATTLIST QUANTITY unit CDATA #IMPLIED >
<!ATTLIST EQUATION type (simulataneous|conditional) >
<!ATTLIST EQUATION value (math_dtd) >
<!ELEMENT MACROMODEL (MACROMODEL PARAMETER) >
<!ATTLIST MACROMODEL name CDATA #REQUIRED >
<!ATTLIST MACROMODEL entity CDATA #REQUIRED >
<!ATTLIST MACROMODEL architecture CDATA #REQUIRED >
<!ATTLIST MACROMODEL PARAMETER.name CDATA #REQUIRED>
```

Figure 14 : Définition de type de document pour modèle Paragon

De la même manière, on peut citer l'exemple du logiciel de simulation SystemVision de la société Mentor Graphics [Sys50], qui permet notamment de créer des modèles de systèmes hétérogènes et de les simuler. Le stockage et la manipulation des informations relatives à ces modèles se fait sous la forme de fichiers XML avec une grammaire exprimée via un fichier XSD (XML Schema Description) dont un extrait est représenté sur la Figure 14.

De part leur nature, les fichiers XML des modèles sont manipulables facilement. Il existe notamment des langages de requête permettant de retrouver rapidement une information particulière et de la traiter. Nous pouvons citer *XPath* et *XQuery*.

La définition des différentes balises est en relation directe avec la conception de meta modèle, notion que l'on a définit dans l'introduction. Le meta modèle va modéliser un langage de modélisation donné, en précisant notamment les différents concepts de modélisation mis en jeu et leurs relations. Cela sera explicité au cours du deuxième chapitre notamment dans le paragraphe 2.2.2 et celui consacré sur la création de meta modèle 2.3.

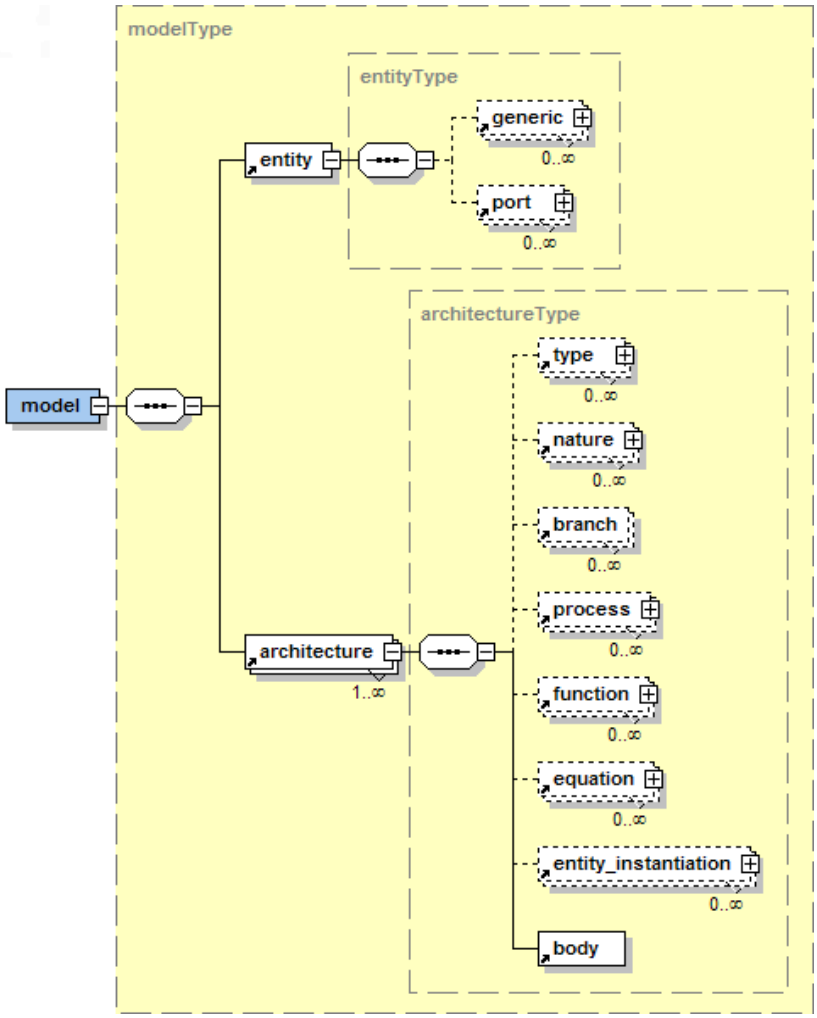


Figure 15 : Extrait du fichier XSD (XML Schema Description) utilisé par SystemVision

1.4 La Modélisation Fonctionnelle des Systèmes

Au sein de notre processus de conception décrit dans les paragraphes précédents reposant sur les recommandations de l'EIA-632 [Eia98], la modélisation fonctionnelle est un processus particulier qui part du cahier des charges et de la formalisation progressive des exigences fonctionnelles pour arriver à :

- Une solution Logique.
- Une solution Physique.

L'aboutissement de cette démarche de modélisation fonctionnelle est le prototype virtuel. Il sera obtenu en y intégrant, par le biais d'un couplage de processus, les choix technologiques que l'on aura faits pour le système.

Les systèmes hétérogènes impliquent par définition plusieurs disciplines, mais aussi plusieurs métiers et domaines de compétence. Les fonctions misent en œuvre peuvent être variées, allant de l'électronique à l'informatique en passant par exemple par le domaine des télécommunications.

Notre objectif est la modélisation des systèmes numériques, analogiques et mixtes, mais il faut tout d'abord être d'accord sur chacun de ces termes. Chaque ingénieur viendra avec sa définition et ce suivant son expérience et son domaine d'activité dans lequel il travaille. La principale différence entre les systèmes numériques et analogiques est la représentation qui est faite des valeurs et du comportement par rapport au temps [TAP03]. Ainsi :

- Un système va être considéré comme numérique s'il manipule et enregistre les informations quantifiées dans le temps en des valeurs discrètes. Celles-ci sont généralement exprimées par des équations booléennes logiques ou par des processus communicants et déclenchés par des évènements [PLV05].

- Un système va être considéré comme analogique, si les informations manipulées ont leurs valeurs qui varient de façon continue dans le temps. Ainsi les équations différentielles ou algébriques sont souvent utilisées pour décrire l'évolution des variables dans un tel système.

De plus, comme les systèmes numériques, les systèmes analogiques peuvent être considérés comme une entité ou un ensemble de constituants (par exemple une résistance ou une capacité par rapport à un filtre ou un amplificateur dans le domaine de l'électronique).

Après avoir posé ces définitions, il faut à présent analyser la manière dont ces systèmes interagissent. Un système mixte est la combinaison de systèmes analogiques et numériques. Dans un tel système, les parties analogiques et numériques interagissent via des convertisseurs numérique-analogique et analogique-numérique. Citons un système composé d'un capteur de température dont la sortie va être échantillonnée, puis traitée par un ordinateur, qui a son tour va commander un élément chauffant. Entre les deux étages du système sont intégrés des convertisseurs permettant l'interaction.

Cette discussion ne se résume pas aux systèmes électroniques. Il faut également l'étendre aux systèmes mécaniques, optiques, hydrauliques, thermiques et électromagnétiques par exemple. Ainsi, un système peut se présenter sous forme multidisciplinaire. Cela peut alors rendre le système très complexe à appréhender dans son ensemble, ce qui implique d'avoir des méthodes pour traiter cette complexité.

Le modèle du système que l'on souhaite est intimement lié à l'aspect selon lequel on observe ce système. On peut notamment classer ces différentes vues suivant trois domaines : Comportemental, Structurel et Géométrique :

- Le domaine Comportemental se préoccupe des fonctionnalités du système. C'est le domaine de description avec le plus d'abstraction. Il n'indique pas comment la fonction est implémentée.

- Le domaine Structurel représente comment le système est décomposé en sous systèmes interconnectés.

- Le domaine Géométrique représente comment le système se situe dans l'espace physique, comme par exemple la répartition physique des composants sur une carte électronique.

Chaque domaine peut être divisé en niveaux d'abstraction. La Figure 15 (décrites par [GK83]) représente, dans le cas de systèmes numériques, les trois axes d'abstraction précédemment décrits (structurel, comportemental, et générique). Dans la Figure 16, ce graphique a été adapté par [TAP03] aux systèmes analogiques.

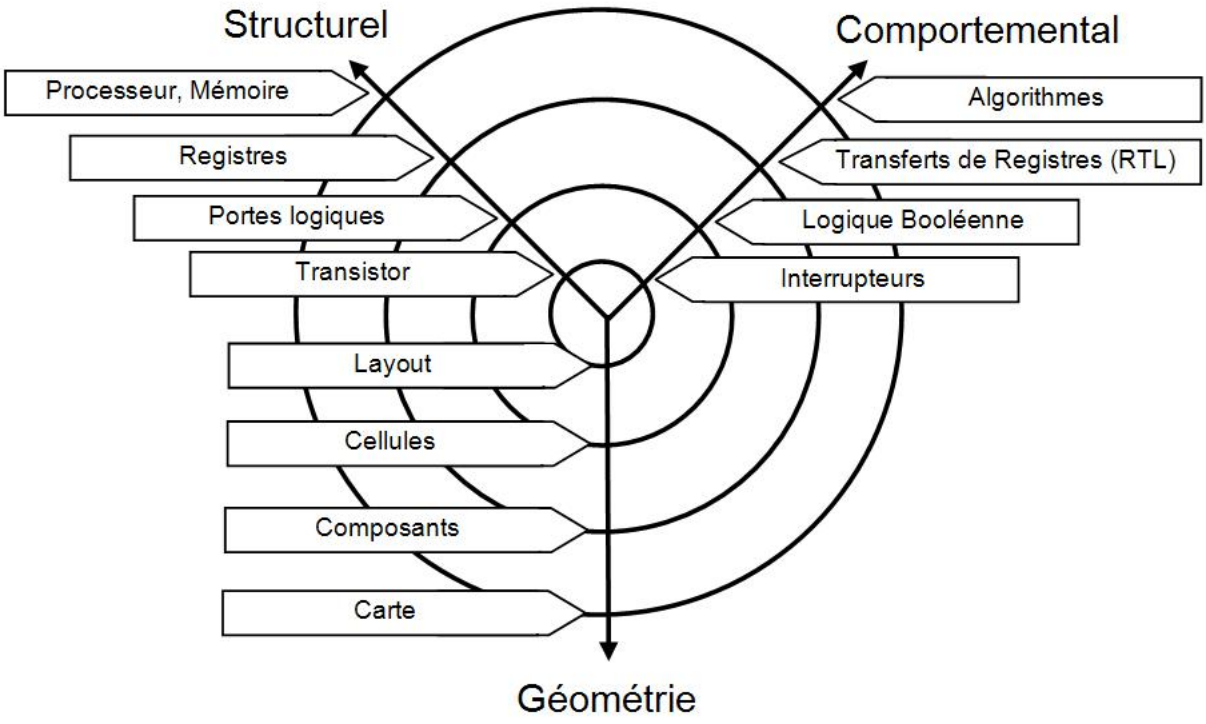


Figure 16 : Les différents niveaux d'abstraction lors de la conception de systèmes numériques [GK83]

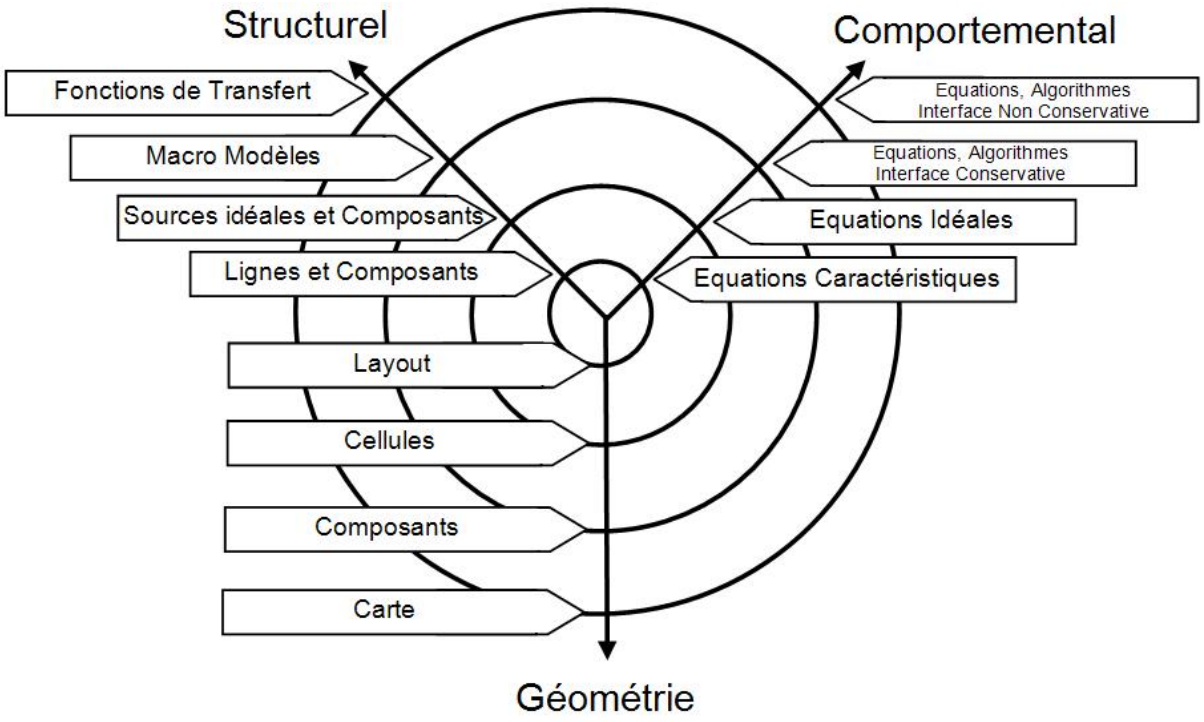


Figure 17 : Les différents niveaux d'abstraction lors de la conception de systèmes analogiques [GK83]

Notre problématique de thèse est de pouvoir modéliser dans un langage adapté les systèmes hétérogènes à plusieurs niveaux d'abstraction, pouvant être régis en tout ou partie, par les lois de conservation de l'énergie. Nous souhaitons être capable d'adopter aussi bien une vue structurelle du système qu'une vue comportementale, la vue géométrique pourra quant à elle être renseignée via des annotations au sein du modèle du système.

1.5 Cartographie des Langages

Dans la continuité de nos recherches concernant la méthodologie de conception de systèmes hétérogènes, et après avoir explicité les différents types de modèles existants, il convient de choisir quel langage pourra répondre au mieux à nos attentes pour être le support de la modélisation du système.

L'expression d'un modèle va se faire par le biais d'un langage de modélisation. Par définition, c'est un langage artificiel qui peut être utilisé pour exprimer des informations, des connaissances, ou des systèmes dans une structure qui est définie par un ensemble de règles.

Un langage de modélisation peut être graphique ou textuel :

- Un langage de modélisation graphique utilise des techniques de diagramme avec notamment des symboles représentant des concepts et des lignes qui connectent les symboles et qui représentent les relations entre ceux-ci.

- Un langage de modélisation textuel quant à lui va utiliser des mots clés normalisés accompagnés de paramètres qui constitueront des expressions interprétables notamment par un outil informatique dédié à ce langage. L'ensemble des instructions et de leurs règles d'écriture sont regroupées au sein d'une grammaire. Un moyen d'expression d'une grammaire est par exemple le formalisme BNF (Backus-Naur Form).

Nous allons nous efforcer, par le biais de la présentation des langages susceptibles de couvrir les processus de développement orientés modèles, d'évaluer celui qui pourra nous permettre d'exprimer le modèle d'un système hétérogène. Ce type d'étude à notamment été réalisé par le groupe de travail WP1 au sein du projet TOPCASED [Top06].

1.5.1 Langages pour la simulation numérique

En électronique, les langages de modélisation utilisés sont les langages de description de matériel communément appelé HDL (Hardware Description Language). Il s'agit des langages informatiques permettant une description formelle des circuits électroniques. Ils peuvent décrire les opérations effectuées par le circuit (fonctionnalités), mais aussi son organisation (structure), et tester le circuit et le vérifier par le biais de la simulation. En électronique numérique, les langages de description de matériel se sont très vite développés, en étant notamment intégrés dans le flot de conception de composants tels les ASICs (Application Specific Integrated Circuit) et les FPGAs (Field Programmable Gate Array). Les HDLs peuvent être utilisés de deux façons. D'une part on peut les utiliser pour modéliser un circuit intégré comme un processeur (le langage peut être synthétisable ou non). Le langage permet alors de modéliser le comportement du composant avant que celui-ci ne soit construit réellement. D'autre part, il est possible de les utiliser pour programmer des composants, tels les FPGAs (le langage doit être nécessairement synthétisable).

1.5.1.1 Le Verilog

Au départ, le langage Verilog était un langage propriétaire développé par la société Cadence Design Systems. Il a été normalisé en 1990 et normalisé par l'IEEE en 1995 (IEEE 1364).

Ce langage est un langage de description de matériel, mais possède une syntaxe proche du langage de programmation C au niveau de la sémantique, les concepts de modélisation y sont différents.

Le langage Verilog est particulièrement adapté dans les descriptions de bas niveaux, mais manque un peu de souplesse dans le comportemental de très haut niveau (par exemple il ne sait pas modéliser les entités abstraites telles que des systèmes d'exploitation, des canaux de communications, etc...).

1.5.1.2 VHDL (Very High Speed Integrated Circuit (VHSIC) Hardware Description Language)

Plus particulièrement utilisé dans les entreprises européennes (contrairement au Verilog plus utilisé dans les entreprises américaines), le langage de description de materiel VHDL a été normalisé une première fois par l'IEEE en 1987 (IEEE 1076-1987). Il est destiné à décrire le comportement et/ou l'architecture d'un module de logique matérielle (une fonction combinatoire et/ou séquentielle. La syntaxe du VHDL est originaire du langage ADA, dont les mots clefs ont été adaptés à la conception de matériel. L'une des particularités du VHDL provient du fait qu'il est possible d'exprimer facilement le parallélisme à l'intérieur d'un circuit [Her02].

1.5.2 Langages pour la simulation mixte

Les besoins en simulation s'élargissant aux systèmes mêlant plusieurs domaines physiques, les deux langages précédents se sont enrichis pour devenir des langages de modélisations mixtes. Ce sont ces types de langages qui sont le plus à même d'être intégré dans notre processus de conception en tant que langage du prototype virtuel.

1.5.2.1 Verilog-AMS

Le Verilog-AMS [Acc98] a été créé sous la tutelle d'Accellera (Organisation de normalisation EDA) afin de mettre en place les extensions analogiques mixtes du Verilog (IEEE-1364). La première version était Verilog-A LRM sortie en juin 1996 puis Verilog-AMS LRM en août 1998. Le langage Verilog-AMS permet de faire la description comportementale des systèmes analogiques et mixtes. Tout comme le VHDL-AMS, le Verilog-AMS peut être applicable aux systèmes électriques et non électriques. Ce langage permet de faire des descriptions de systèmes, en utilisant des concepts tels que les noeuds, les quantités de branche, et les ports. Les signaux de type analogique et numérique peuvent être présents dans le même module.

1.5.2.2 MAST

Le langage MAST est un langage propriétaire développé en 1984 et complètement lié au simulateur SABER développé à l'origine par la société Analogy et appartenant actuellement à la société SYNOPSYS [Sab]. Depuis plus de vingt ans, en tant que précurseur des langages de modélisation, le MAST domine le marché des langages de description de matériel à signaux et technologie mixte, de part le fait que l'outil Saber est implémenté notamment dans de nombreuses entreprises automobiles et aéronautiques. Ce langage propose néanmoins des mécanismes de modélisation obsolètes et non adaptés notamment pour la modélisation de système numérique. **Le langage MAST est exclusif à l'outil Saber, et présente le défaut de ne pas être normalisé**. Il permet cependant d'avoir au sein même du code du modèle des instructions adressant directement le cœur du simulateur afin d'aider à la convergence (« control section »). Ainsi **l'inconvénient majeur de Saber, par rapport à notre vision de plate-forme coopérative, est son langage propriétaire, MAST**, qui ralentit sa diffusion. Dernièrement, suite au rachat de l'outil par la société Synopsys et grâce à une réaction

commerciale naturelle face à la montée en puissance de VHDL-AMS, une nouvelle initiative a été lancée. Il s'agit d'une proposition OpenMAST™ [Syn04]dont l'objectif est de faciliter l'accès au code des modèles écrits en MAST. Ce langage ne reprend cependant qu'une sous partie de la grammaire du MAST originel.

1.5.2.3 VHDL-AMS

Le langage VHDL-AMS (VHSIC-Hardware Description Language – Analog and Mixed Systems) normalisé en décembre 1999 sous la référence IEEE 1076.1-1999 [Vhd99], a été développé comme une extension du langage VHDL (IEEE 1076-1993) décrit ci-dessus (1.5.1.2). Le VHDL-AMS permet la modélisation et la simulation de circuits et de systèmes logiques, analogiques et mixtes (cf. [Her02] et [Tai02]).

En tant que sur-ensemble du VHDL, le VHDL-AMS profite des capacités de ce langage à modéliser des objets abstraits manipulant des signaux quantifiés à des moments discrets dans le temps. Aux instructions concurrentes et aux instanciations des modèles du VHDL, le VHDL-AMS ajoute les instructions simultanées qui permettent de manipuler des valeurs à temps continu transportées par les objets « QUANTITY », et le concept de synchronisation des noyaux de simulations numériques et analogiques utiles lorsque l'on veut forcer des points de simulation à des temps donnés. Les instructions simultanées mettent en jeu des équations différentielles pouvant être simples, sélectives ou conditionnelles et qui relient les quantités. Le VHDL-AMS dispose aussi de l'objet « TERMINAL » jouant le rôle de nœud de connexion pour le domaine analogique. L'objet « TERMINAL » est associé à une « NATURE » qui définit un domaine physique. Afin que l'on puisse résoudre l'ensemble des équations différentielles, le critère de solvabilité doit être validé localement et les conditions initiales doivent être spécifiées.

Citons quelques simulateurs pouvant manipuler des modèles écrits dans ce langage qui sont déjà disponibles sur le marché :

- AdvanceMS [Adv] ANACAD (Mentor Graphics),
- SystemVision [Sys50] version 5.0 de Mentor Graphics,
- SIMPLORER 7.0 [Ans] développé par ANSOFT,
- SMASH™ 5.8 [Sma06] de DOLPHIN Integration.

1.5.2.4 Modelica

Modelica est un langage ouvert dont le développement et la promotion sont organisés par l'association Modelica [Mod07]. La première version de ses spécifications a été finalisée en septembre 1997 (version 1.0). La version la plus récente des spécifications est la version 2.2 (Mars 2005). C'est un langage de modélisation orienté objet qui permet de modéliser des systèmes complexes pouvant être multidisciplinaires (thermique, mécanique, hydraulique, électronique,…).

Les modèles y sont mathématiquement décrits à l'aide d'équations pouvant être algébriques, différentielles ou discrètes. Les algorithmes de résolution pour les systèmes équations y sont très efficaces et permettent la manipulation de modèles complexes pouvant compter plusieurs milliers d'équations. Ce langage est d'ailleurs utilisé dans des simulations de type hardware-in-the-loop (simulation mêlant prototype réel et prototype virtuel).

Une étude a été réalisée dans la référence [CEM02] par plusieurs personnes dont certaines sont impliquées dans le développement d'un outil supportant ce langage (Dymola). Ils mettent évidemment en avant les qualités que nous avons citées plus haut sur ce langage.

Ils réalisent aussi un comparatif avec le langage VHDL-AMS. Ils mettent particulièrement l'accent sur les différences entre les deux langages, notamment les

possibilités d'approche orienté objet comme la définition de classes, la notion d'héritage, qui n'ont pas leurs équivalents en VHDL-AMS (surcharge d'opérateur).

Ils soulignent aussi qu'il est possible d'intégrer dans des modèles modelica des informations concernant leur représentation graphique, facilitant leurs utilisations dans des outils différents.

Un point plus important est aussi évoqué et concerne la gestion de la simulation mixte (analogique-numérique). La simulation d'un modèle mixte VHDL-AMS met en jeu le couplage de deux simulateurs distincts dédiés respectivement à la partie discrète et à la partie continue, qui vont se synchroniser à certaines dates T (dépassement de seuil, évènements). Modelica voit quand à lui à tout instant l'ensemble des équations différentielles, algébriques et discrètes comme un seul et unique système d'équations ce qui rend notamment la synchronisation automatique entre les parties discrètes et continues. Cela met néanmoins en avant qu'il n'y a pas dans Modelica une prise en charge adaptée du domaine discret comme le fait le VHDL-AMS. Celui-ci possède en effet de part son héritage du VHDL, des instructions bien adaptées à ce type de modélisation pour preuve son utilisation depuis de nombreuses années dans le flot de conception des circuits intégrés. De plus le VHDL-AMS possède par le biais du VHDL un accès à la synthèse logique (dans sa version synthétisable).

Modelica fait parti des langages candidats au support de la solution physique. Il sera comparé à d'autres candidats au sein de la problématique à la fin de ce chapitre (cf. paragraphe 1.6.2).

Citons quelques simulateurs pouvant manipuler des modèles écrits dans ce langage qui sont déjà disponibles sur le marché de façon open source ou commerciale :

- Dymola [Dym07] outil commercial développé par Dynasim,
- MathModelica [Mmo07] System Designer outil commercial développé par MathCore,
- OpenModelica Project [Opm07] outil gratuit en développement (dernière version 1.4.3).

1.5.2.5 Tableaux comparatifs des langages MAST, Verilog-AMS et VHDL-AMS :

Il s'avère que ces langages de description de matériel semblent répondre aux principaux critères que l'on recherche pour la modélisation de systèmes pluridisciplinaires et multi abstractions.

Le Tableau 1, développé sur les quatre pages suivantes, compare de manière détaillée les différents langages de description de matériel mixtes précédemment présentés. Il s'appuie sur une version préliminaire développée par [PLV05]. Nous avons actualisé les informations, notamment au niveau des outils de simulation employés qui ont été améliorés au niveau de la couverture des normes par exemple. Nous avons ajouté sur le tableau, le langage propriétaire MAST, car il est très utilisé dans le monde de l'industrie et cela permet de voir ses forces et ses faiblesses vis-à-vis des nouvelles normes.

Tableau 1 : Comparatif de quelques langages de description de matériel mixtes

Type de Fonctionnalités	Fonctionnalités	VHDL-AMS	MAST	VERILOG-AMS
Aspects du Langage	**Définition**	Norme IEEE 1076.1-1999 Extension du VHDL (IEEE 1076)	Langage Propriétaire lié à l'outil SABER	Accelera standard Version 2.2 (November 2004) Extension du Verilog (IEEE 1364)
	Héritage	Origine ADA Non prise en compte de la casse	Non prise en compte de la casse	Sémantique proche du C Prise en compte de la casse
	Modularité	Entité active, Architecture (simple ou multiple), Package, Configuration	Un seul fichier avec sections (header, paramètre, when, value, équation, control section)	Module avec vues internes externes et interface
	Généricité	Paramètres Génériques Instruction « generate »	Paramètres Génériques	Paramètres Génériques Instruction « generate »
	Gestion de Bibliothèques	Oui (unité de conception pré-compilée)	Non	Non
Expression de la Structure	**Ports**	Discret ou continu Conservatif ou flot de signal		
	Structure	Instanciation de composants de manière hiérarchique		
	Sémantique Partie Conservative	Nature = Domaine d'Energie Ss-Types = attribut des natures Non Prédéfinis	Prédéfinis (thermal_c, electrical,…)	Disciplines = Domaine d'Energie Nature = attribut des disciplines Disciplines prédéfinies

Type de Fonctionnalités	Fonctionnalités	VHDL-AMS	MAST	VERILOG-AMS
Expression du Comportement	**Objets**	Terminal, Quantity, Signal, Variable, Constant	var, ref, val, state, branch number, struc thermal_c electrical	Variable, fil, registre
	Instructions	Concurrentes, Séquentielles, Continues (Simultanées et Procédurales)	Séquentielles, Continues (Simultanées)	Concurrentes, Séquentielles, Continues (Simultanées et Procédurales)
		Instructions concurrentes avec instructions simultanées	Séquentielles (sections Value, When) Simultanées (section équation) Procedural (section value)	Instructions continues dans un bloc analogique (un par module)
	Expression des Equations Algébriques Différentielles (DAE)	• Support des formes explicites et implicites des équations • Equations Simultanées • Instruction Procedural • Dérivé et intégration /temps -> attribut quantité 'dot, 'integ (plusieurs degrés possibles) • Fonctions Maths dans Package séparé IEEE 1076-2 • Fonctions Foreign (interfaçage autres codes et simulateurs)	Forme explicite des équations Support limité des formes implicites d'équation. Dérivation : d_by_dt (un seul ordre) Fonctions Maths prédéfinis Import Fonction C	Forme explicite des équations Support limité des formes implicites d'équation. Formulation Procedurale Dérivation : ddt (un seul ordre) Fonctions Maths prédéfinis
	Gestion des discontinuités	Les discontinuités doivent explicitement être annoncées dans le modèle Réinitialisation par l'utilisateur dans le code après une discontinuité supportée (instruction break)	Control Section Instructions agissant directement sur le simulateur Saber	Les discontinuités doivent explicitement annoncées dans le modèle L'ordre de la discontinuité doit être spécifié

Type de Fonctionnalités	Fonctionnalités	VHDL-AMS	MAST	VERILOG-AMS
Sémantique Partie Conservative	**Domaines d'Energie**	Nature = Domaine d'Energie Ss-Types = attribut des natures Non Prédéfinis Quantités de Branche	Prédéfinis (thermal_c, electrical,…) Branch variable	Disciplines = Domaine d'Energie Nature = attribut des disciplines Disciplines prédéfinies Fonction d'accès et objets de branche
	Formulation	Formulation sous forme d'équations simultanées	Value section Equation section	Formulation orientée circuit avec source et sonde de mesure
		Instructions concurrentes avec instructions simultanées	Séquentielles (sections Value, When) Simultanées (section équation)	Instructions continues dans un bloc analogique (un par module)
Sémantique Partie Flot de Signal	**Interface du modèle**	Quantité libre directionnel dans le port map	Ports Input et Output	Port d'un module associé à une disciplines avec une seule nature (signal-flow discipline)
	Blocs Fonctionnels	Transformation de Laplace et en Z	Fonction MAST : transfer_function Fonctions C externes	Transformation de Laplace et en Z
Contrôle de la Simulation	**Solvabilité**	Critère de solvabilité vérifié lors de la compilation des unités de conception	Pas de critère de solvabilité	Pas de critère de solvabilité
	Pas de Temps	Le pas de temps peut être fixé		
	Tolérance	Annotation sous forme de chaîne de caractère non formellement lié au simulateur		Tolérance de type Spice définie dans les natures

Type de Fonctionnalités	Fonctionnalités	VHDL-AMS	MAST	VERILOG-AMS
Aspect Mixte	**Interfaces**	Les interfaces A/N et N/A peuvent se faire via des attributs (‘ramp, ‘slew, ‘above) Synchronisation simulateur : A -> N ‘above N -> A break on Pas d’association de port directe	Fonction de seuil analogique threshold , rend un événement booléen Insertion d’interface automatique	Filtre d’interface N/A et A/N et détecteur d’évènement Insertion automatique de module de connexion (intra ou inter disciplines)
	Interaction Comportementale	Accès aux signaux dans un contexte continu Accès aux quantités dans un contexte discret	Variables continues accessibles dans when section (mais certaines limitations d_by_dt non autorisé) Signaux accessibles dans value section	Accès aux nœuds et variables discrets dans un contexte continu Accès aux nœuds et variables continus dans un contexte discret
Exemples d’Outils supportant le langage (Mars 2007)		SystemVision 5.0 Advance MS v2005.3 Mentor Graphics	Saber v-2004.03	Cadence AMS Simulator

1.5.3 Langages pour le codesign matériel/logiciel

L'approche codesign matériel/logiciel se place dans une méthode mêlant une conception simultanée du **matériel numérique et du logiciel**. L'objet du codesign [DMG97] est de réaliser les objectifs de la conception au niveau système en regroupant et exploitant la synergie du matériel et du logiciel par le biais d'une conception concourante.

1.5.3.1 Le langage SystemC

SystemC est un langage de description de matériel issu du langage C++, lui permettant de modéliser non seulement des systèmes matériels, mais aussi des systèmes logiciels, matériel-logiciel, ou non-partitionnés.

Il a été développé en commun par plusieurs entreprises (ARM Ltd, Cadence Design Systems, CoWare, Mentor Graphics,…) et a été normalisé IEEE en 2005 (IEEE 1666-2005).

Ce langage est basé sur un ensemble de bibliothèques créées en langage C++ qui permettent de faire de la modélisation logiciel/matériel par le biais de spécifications exécutables et de plusieurs niveaux d'abstraction pour un même système.

SystemC est très peu adapté au bas niveau. Il a été conçu pour modéliser les systèmes de niveaux intermédiaires à très haut. Il sait modéliser des entités propres aux OS telles que des sémaphores, ... et permet de modéliser simplement des canaux de communication abstraits (ethernet, UMTS par exemple). Il « laisse » les très bas niveaux (transistors) à des langages tels Verilog, VHDL et autres langages dédiés (SPICE, ...).

1.5.3.2 SystemVerilog

SystemVerilog est une extension du langage Verilog (IEEE 1363-2001). Il se veut le premier langage de description et de vérification de circuits. C'est une nouvelle norme définie en 2005 (IEEE 1800-2005). Sa principale caractéristique le différenciant des autres langages vient du fait qu'il offre des fonctionnalités avancées de conception pour fournir des capacités de bancs de tests et d'assertion détaillées pour la vérification intégrée de haute performance.

1.5.4 Langages pour la conception de système logiciel

Les différents langages et méthodes mis en œuvre dans le cadre de la conception de logiciel ont permis de voir émerger et de mettre en place un certain nombre de concepts intéressants dans l'optique d'une conception amont.

1.5.4.1 UML 2.0

UML (pour Unified Modeling Language) est un langage de modélisation graphique soumis et recommandé pour la première fois par l'OMG [OMG] en 1997. Il est dédié à la conception de systèmes logiciels à haut niveau.

Il a pour objectif, cinq activités principales du processus de conception [Bro03] [Ham05] :

- Une description graphique du système selon plusieurs points de vue,
- La spécification des besoins et de la mise en œuvre,
- La visualisation pour faciliter la compréhension et la communication parmi les partenaires de la conception avant la réalisation du système,
- La représentation graphique de systèmes complexes,
- La documentation de la totalité du projet, dès les spécifications jusqu'aux tests de fonctionnement.

Il est structuré sur un meta modèle définissant les éléments de modélisation (concept manipulé par le langage) et la sémantique de ces éléments (définitions et sens de leurs utilisations) (cf. 2.3 La Conception de Meta Modèle).

C'est un langage formel organisé autour de diagrammes. La version actuelle d'UML est la version 2.0. Une nouvelle version se propose d'améliorer l'organisation générale du langage, de simplifier sa syntaxe et sa sémantique, de fournir des éléments plus adaptés pour la modélisation comportementale des systèmes. Ces améliorations sont proposées sous la forme de modifications des anciens diagrammes (UML 1.5) et aussi avec la proposition de nouveaux diagrammes. Ainsi, la version actuelle d'UML 2.0 dispose de 13 diagrammes officiels (contre 9 dans la version 1.5).

Les diagrammes sont répartis en trois catégories :

- Les diagrammes structuraux,
- Les diagrammes comportementaux,
- Les diagrammes d'interactions (qui peuvent être vus comme une sous catégorie des diagrammes de comportement).

La Figure 17 schématise ces différents diagrammes sous forme d'un diagramme de classe.

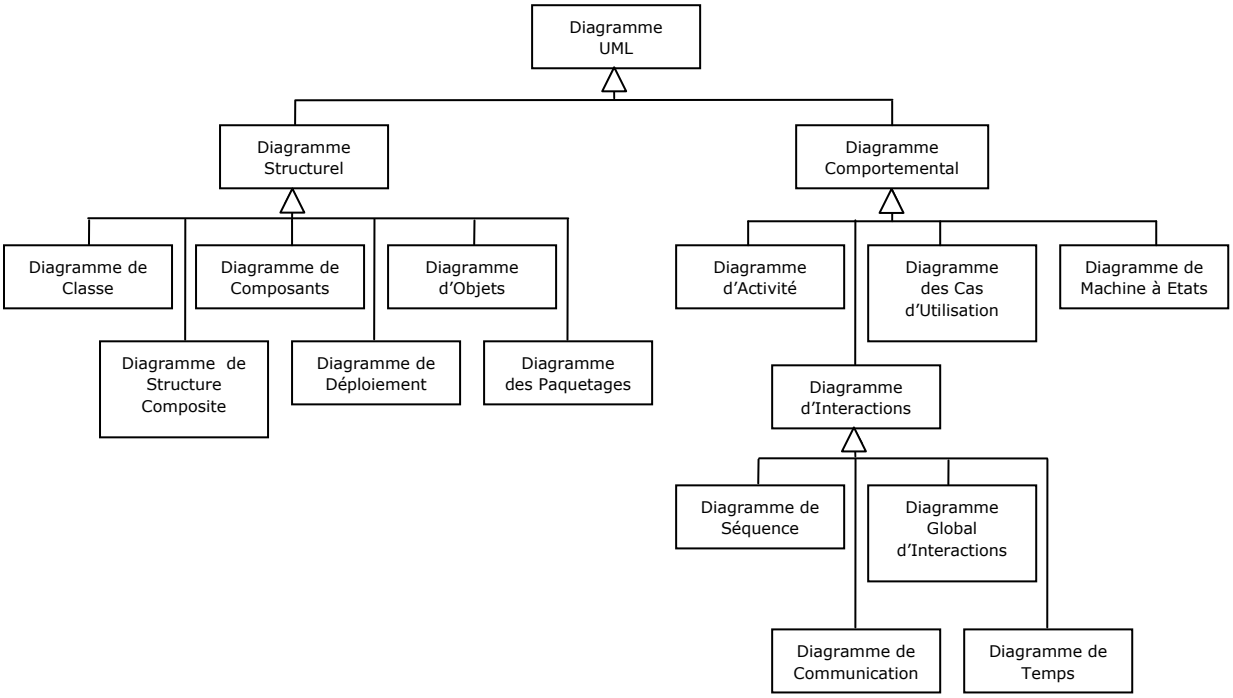


Figure 18 : Les différents diagrammes d'UML 2.0 sous forme de diagramme de classe

L'OMG ne propose cependant pas de méthodologie associée à sa recommandation UML 2.0. La méthodologie à employer pour utiliser l'ensemble des diagrammes UML pour la conception et la modélisation de système reste donc à définir. Il existe actuellement de nombreux outils qui supportent la recommandation UML 2.0 et qui permettent la modélisation et la génération de code [Arg] [Vis] [Ecl] [Omo].

UML 2.0 est un langage de modélisation pour le domaine logiciel qui est à présent utilisé avec succès, notamment lors des phases de conception. Pour combler les lacunes de UML dans le domaine de l'ingénierie système, un groupe de travail a été formé en 2003 pour créer le langage SysML.

1.5.4.2 SysML

SysML vise à enrichir le langage UML en ajoutant des notions et des concepts propre à l'ingénierie système [Sys03]. Il supporte ainsi la spécification, l'analyse, la conception, la vérification et la validation de systèmes. Les améliorations par rapport à UML sont notamment :

- La sémantique de SysML est plus flexible. SysML lève les restrictions « logiciel » d'UML et ajoute deux nouveaux diagrammes : le diagramme d'exigence (utilisable dans un contexte de gestion des exigences) et le diagramme de contraintes paramétriques (utilisable dans un contexte d'analyse de performance et de qualité).
- Les diagrammes de classes et de structure composite ont été modifiés pour donner lieu au diagramme de bloc.

La Figure 18 représente les différents diagrammes disponibles dans le langage SysML en comparaison avec ceux disponibles dans la recommandation UML 2.0.

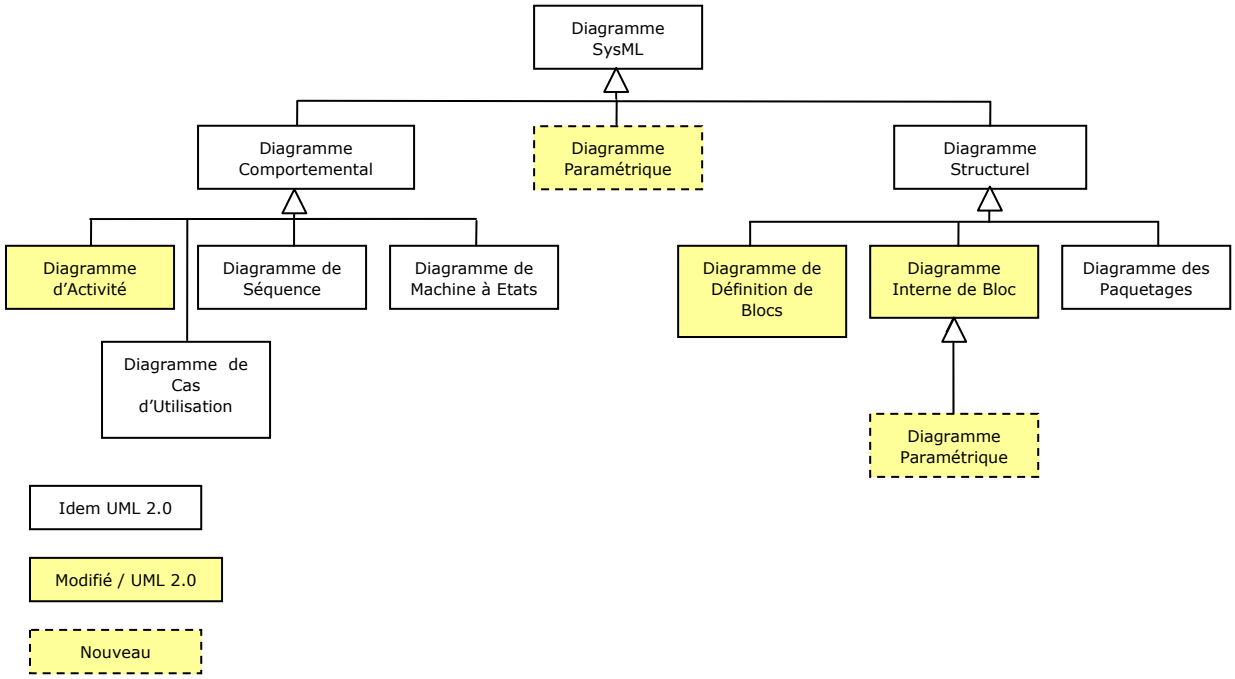


Figure 19 : Les différents diagrammes SysML 1.0

1.5.5 Les Outils dédiés

1.5.5.1 Logiciel Pspice

C'est une norme pour la simulation électronique analogique utilisé depuis son introduction en 1985. Il est actuellement accessible en tant que module fonctionnel de l'outil Orcad de la société Cadence [Orc06]. PSpice est un simulateur complet pour la conception analogique. Avec ses modèles internes et ses bibliothèques largement rependues et développées, les systèmes à hautes fréquences jusqu'aux circuits intégrés de basse puissance, peuvent être simulés. La plupart des fabricants de composants électroniques fournissent aujourd'hui les modèles PSpice de leur gamme de composants. PSpice est un simulateur complet pour la conception analogique. Dans la bibliothèque de PSpice, des modèles peuvent être édités, mais les utilisateurs peuvent également créer leurs modèles pour de nouveaux dispositifs à partir des fiches techniques. « PSpice A/D

Basics » est un simulateur de signaux mixtes. C'est une version plus élaborée de PSpice qui peut être employée pour simuler des systèmes mixtes sans limite théorique de taille, contenant des parties analogiques et des éléments numériques. Cependant, il est mal adapté lorsque le système à modéliser devient de plus en plus complexe et requiert une vue hiérarchique.

1.5.5.2 MatLab/Simulink

MatLab est un environnement de calcul numérique créé par la société The MathWorks [Mat]. Il permet une manipulation simple des matrices, de tracer des foncions et des données, d'implémenter des algorithmes, de créer des interfaces utilisateurs, et de s'interfacer facilement avec d'autres programmes (cf. 1.3.2.1 Cosimulation).

MatLab est construit autour d'un langage communément appelé M-code. Le code est saisi dans une fenêtre de commande. La séquence de commandes peut alors être sauvegardée dans un script, ou encapsulée dans une fonction.

MatLab peut être complété par de multiples outils appelés plugins ou toolbox :

- Communications Toolbox,
- Control System Toolbox,
- Neural Network Toolbox,
- Optimization Toolbox,
- Robust Control Toolbox,
- Statistics Toolbox,
- System Identification Toolbox,
- Virtual Reality Toolbox,
- Excel Link,
- Simulink [Mata].

L'outil Matlab permet de modéliser, simuler et analyser des systèmes dynamiques multi-domaines. Son interface principale repose sur un diagramme de blocs graphiques, avec des bibliothèques de blocs prédéfinis permettant la modélisation système. L'outil profite de la puissance de calcul numérique offerte par l'environnement MatLab. Il est principalement utilisé dans les applications de contrôle et dans la manipulation de signaux numériques.

Une de ses principales caractéristiques est que les connexions ne sont que de type « flot de signal » (signal flow). Dans ce cas de figure, les sorties ne sont pas influencées par les entrées (impédance de sortie nulle et d'entrée infinie). [GO04] réalise une comparaison entre une modélisation sous MatLab/Simulink et sous VHDL-AMS d'un composant piézoélectrique. Il constate que le VHDL-AMS, de part son type de connexion conservatif (échange bidirectionnel de l'énergie) et son écriture des équations physiques, permet une modélisation plus aisée et plus directe. En terme de résultats de simulation, ils sont identiques. Nous en concluons que la co-simulation entre un outil VHDL-AMS et Simulink serait une solution adéquate, en profitant d'un côté de l'efficacité de modélisation des algorithmes sous forme de schéma blocs sous Simulink, et d'un autre côté de la richesse des techniques de modélisation de systèmes multi physiques proposés par la norme IEEE 1076.1-1999 VHDL-AMS.

Pour être complet au niveau des possibilités d'utilisation de Simulink au sein d'un flot de conception système, il convient de citer certains logiciels pouvant être couplés à Simulink dans une optique de génération de code. Ainsi, en utilisant un autre produit de the MathWorks "Real-Time Workshop" [Matb], Simulink est capable de générer du code C pour des systèmes embarqués à partir d'algorithmes créés sous Simulink. Avec le logiciel "Simulink HDL Coder" [Matc], du code VHDL et Verilog synthétisables peuvent être générés à partir de Simulink.

La hiérarchisation des équipes de développement fait que chaque niveau du processus de conception système n'est pas conçu par les mêmes personnes, et chaque population a ses propres habitudes. Par exemple, la description de plus haut niveau d'une chaîne de traitement de communication (exemple UMTS) sera développée par des personnes du traitement du signal, qui représentent généralement leurs algorithmes en code C ou en Matlab. C'est le niveau en dessous qui sera traité par les électroniciens, qui travailleront alors en Verilog, VHDL ou SystemC.

1.6 Notre Problématique

1.6.1 Proposition d'un processus générique de conception

Comme nous l'indiquons en introduction de ce mémoire, nous nous référons aux recommandations de l'EIA-632 pour construire notre approche de conception à l'intérieur d'une démarche globale de développement normalisée. Celle-ci fait intervenir treize grands processus déclinés en trente trois exigences. Ces processus sont naturellement interconnectés, nécessitant dans leur mise en œuvre des procédures de tissage comme nous l'avons introduit en 1.3.1. Un axe important d'effort est aujourd'hui de construire sur la base des textes de la norme, une représentation graphique plus formalisée de cette norme. Notre collègue S. Rochet en propose une formalisation en SPEM [Roc06] et une extension sous la forme d'une modélisation à trois niveaux : générique, métier et projet.

Pour argumenter notre problématique, seul nous intéresse, à l'intérieur du processus générale de développement, le processus de conception associé à l'évaluation technique indispensable. Ces deux processus sont reproduits sur la Figure 19 et la Figure 20 :

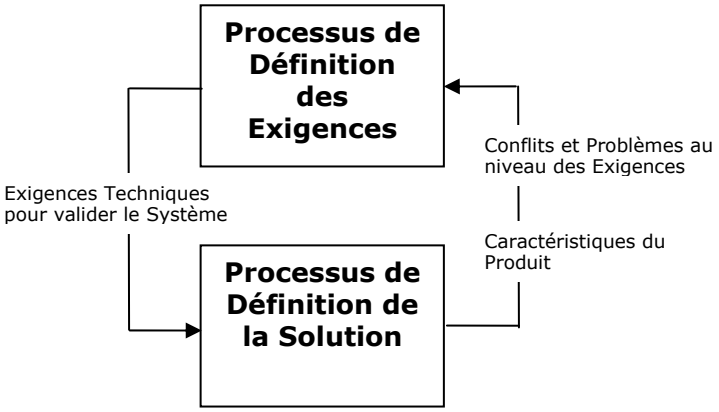


Figure 20 : Processus de Conception Système selon l'EIA-632

Nous restons dans cette démarche générique qui prévoie les étapes suivantes, à partir du cahier des charges :

- Formalisation des exigences
- Solution Logique
- Vérification Formelle
- Solution Physique
- Simulation et Validation

Cette philosophie générale est reprise de façon graphique sur la Figure 21 que nous avons élaboré. On y retrouve les langages UML et SysML dans la branche "fonctionnelle" du cycle en Y. Ils vont permettre d'exprimer les spécifications du système à modéliser. Le

point de départ est le cahier des charges. Par transformations successives et en utilisant des diagrammes UML (diagramme de contexte, diagramme de cas d'utilisation, diagramme de séquence), on aboutit à une solution logique exprimable notamment sous le formalisme HiLeS. On peut retrouver cette démarche appliquée notamment dans les thèses [Ham05] [Mau05].

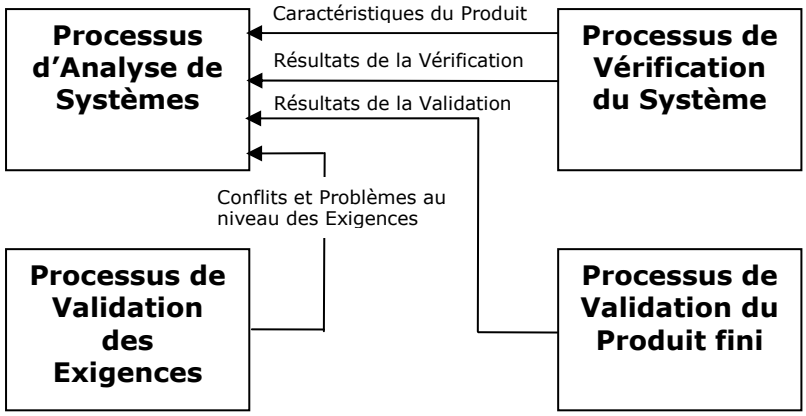


Figure 21 : Processus d'Evaluation Technique selon l'EIA-632

Notre problème est de construire la Solution Physique, à partir de la Solution Logique vérifiée et des contributions de toute nature permettant de choisir notamment les technologies. La Solution Physique va donc être un point de convergence de plusieurs modèles qu'il faudra intégrer ensemble et/ou transformer. **Notre problématique est donc d'étudier les moyens pour converger vers cette solution physique, mais aussi d'en trouver un langage de modélisation pour support.**

1.6.2 VHDL-AMS au centre du Prototypage Virtuel

L'étude des différents langages de modélisation dans le paragraphe 1.5, nous a permis de voir d'apprécier un certain nombre de langages disponibles à l'heure actuelles et susceptibles de répondre à nos attentes en temps que support de la solution physique. Le reprend les principales caractéristiques qui nous semblent primordiales pour le support de la solution physique. Il en ressort que les mécanismes de modélisation de la norme VHDL-AMS (multi-domaine, multi-abstraction, multi-architecture) apparaissent les plus adaptés à notre problématique. C'est pourquoi nous proposons le langage de description de matériel VHDL-AMS normalisé IEEE (IEEE 1076.1-1999) comme candidat au langage de modélisation du prototype virtuel.

Ce langage de haut niveau permet d'écrire un cahier des charges abstrait et simulable. Une architecture de système peut être faite de manière fonctionnelle en associant des modèles existants ou à créer. Les modèles peuvent être décrits sous forme comportementale ou structurelle.

Le fait d'être une norme, le VHDL-AMS autorise l'échange entre collaborateurs ou de fournir à des clients, des modèles de haut niveau dans le cadre d'une bibliothèque IP (Intellectual Property). Il est aussi possible de décrire les composants à très bas niveau en écrivant sous forme d'instructions simultanées les équations différentielles issues des études physiques. On peut donc ajouter dans les modèles des imperfections et ainsi voir l'influence sur les performances de la modification d'un paramètre.

VHDL-AMS permet de modéliser l'environnement du système, optimisant ainsi son étude et sa mise au point. Le VHDL-AMS prend en charge la conception mixte et permet

de mettre en jeu plusieurs domaines physiques. Ainsi une modélisation peut prendre en compte les phénomènes électriques, optiques, mécaniques, thermiques et leurs couplages. Un autre atout du VHDL-AMS est de proposer une approche multi-abstraction permettant ainsi d'associer des modèles de haut niveau et des modèles de bas niveau dans le même modèle global ayant une approche structurelle. Ainsi, on peut détailler certaines parties plus finement, alors que d'autres parties sont modélisées de manière plus abstraite, requérant ainsi un temps de calcul inférieur (à la simulation).

Langages / Caractéristiques	Verilog-AMS	MATLAB	Modelica	MAST	VHDL-AMS
Norme	NON	NON	NON	NON	IEEE 1076.1-1999
Présentation	Extension du Verilog (IEEE 1364)	Logiciel calcul numérique + Simulink	Langage de modélisation orienté objet	Langage propriétaire lié à l'outil SABER	Extension du VHDL (IEEE 1076)
Modularité	Mono-Architecture	Mono-Architecture	Mono-Architecture	Mono-Architecture	Entité Active Multi-Architecture
Equations algébro-différentielles	OUI Forme explicite ordre 1	OUI Forme implicite ordre 1	OUI Forme implicite ordre 1	OUI Forme explicite ordre 1	OUI Forme implicite ordre N
Multi-Disciplines	OUI	NON	OUI	OUI	OUI
Conservatif / Flot de signal	OUI / OUI	NON / OUI	OUI / OUI	OUI / OUI	OUI / OUI
Multi-Abstraction	OUI	NON	NON	OUI	OUI
Gestion des discontinuités	OUI	NON	OUI	OUI Control Section	OUI Instruction Break
Dépendance à un Outil	NON	OUI	NON	OUI	NON

Tableau 2 : Comparatif de cinq candidats au support de la solution physique

Un des intérêts du VHDL-AMS est de construire une bibliothèque de modèles génériques réutilisables afin d'augmenter la productivité. A la suite de sa normalisation en 1999, de nombreux projets ont été mis place autour de ce langage [DSK04]. Les laboratoires de recherches et les industriels ont unis leurs efforts afin de développer les bibliothèques de modèles dans ce langage. Ces efforts ont aussi permis d'améliorer constamment les outils dédiés à ce langage et notamment les simulateurs de modèle. Les critères importants des concepteurs sont en effet pour ces outils : la couverture de la norme, une interface conviviale, une gestion des bibliothèques de modèles efficace, des simulations performantes en termes de précision et de temps de mise en œuvre.

Mon travail de thèse s'inscrit dans cette proposition de modélisation système en VHDL-AMS. Nous avons représenté cette démarche sur la Figure 22 et sur la Figure 23. Elle met en œuvre les recommandations de l'ingénierie système (EIA-632 et ingénierie guidée par les modèles) avec plusieurs modèles qui par transformations successives permettent de converger à partir des spécifications (en vert) vers d'une part une solution logique qui a pour support le formalisme HiLeS (en bleu) et d'autre part vers une solution physique que nous associons au prototype virtuel (en rouge).

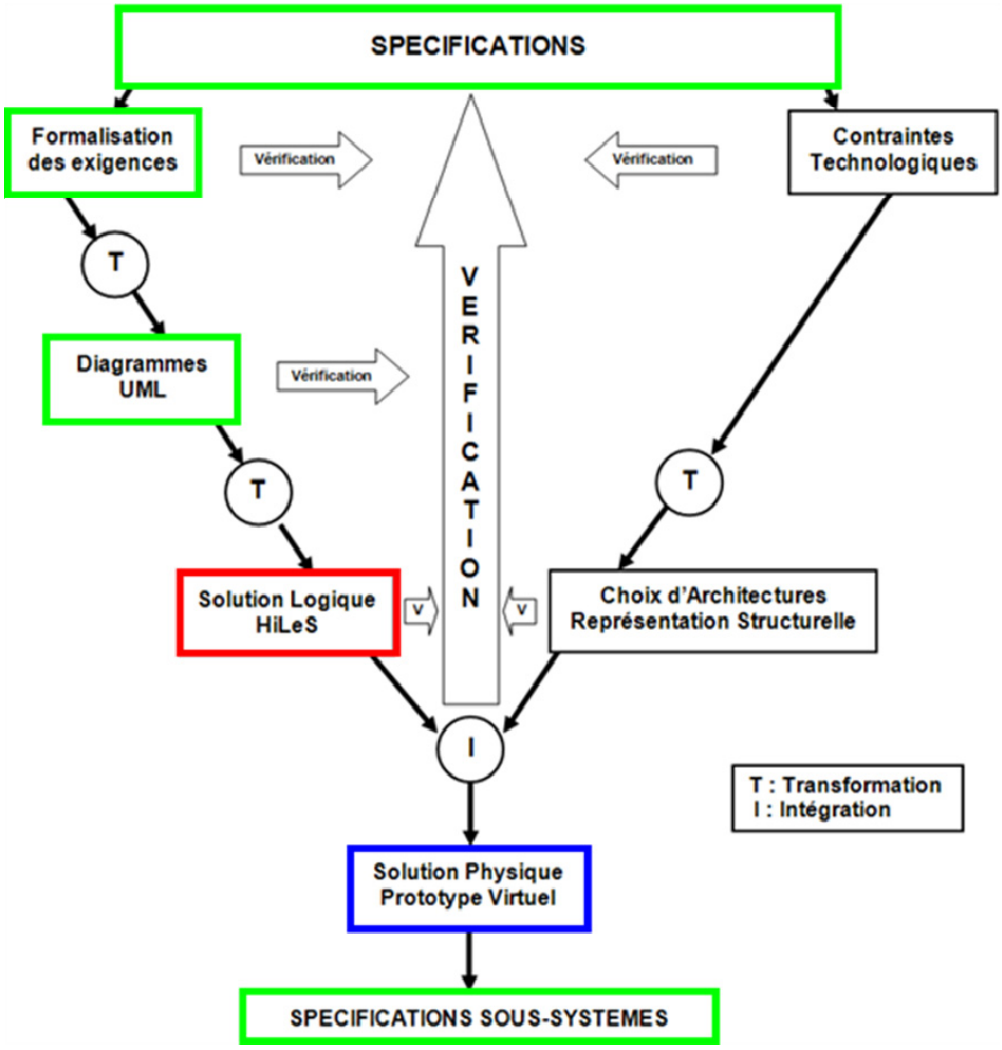


Figure 22 : Notre proposition d'un processus de conception

Notre travail va consister à :

- Créer l'interface entre la solution logique et la solution physique souhaitée. Cela implique notamment la traduction de certains éléments de la solution logique en VHDL-AMS, comme par exemple la traduction en VHDL des réseaux de Petri qui font parti du formalisme HiLeS utilisé pour la solution logique.

- Intégrer au sein d'une modélisation VHDL-AMS faisant office de prototype virtuel, des modèles écrits dans des langages de modélisation différents.

Sur ce dernier point, il convient de préciser que de nombreux industriels utilisant un langage de modélisation propriétaire à un outil, ont un réel besoin de connaître les moyens à mettre en œuvre en terme de méthodologie à appliquer, d'experts à impliquer, et en temps de migration pour obtenir des modèles VHDL-AMS.

Nous verrons au cours de cette thèse quels sont les moyens à mettre en œuvre et nous proposerons une méthodologie qui sera testée par le biais de plusieurs exemples d'applications pertinents, pris dans le monde de la recherche et de l'industrie. Ces différentes modélisations nous permettrons d'évaluer si notre choix d'utiliser le VHDL-AMS comme langage de modélisation support de la solution physique est valide.

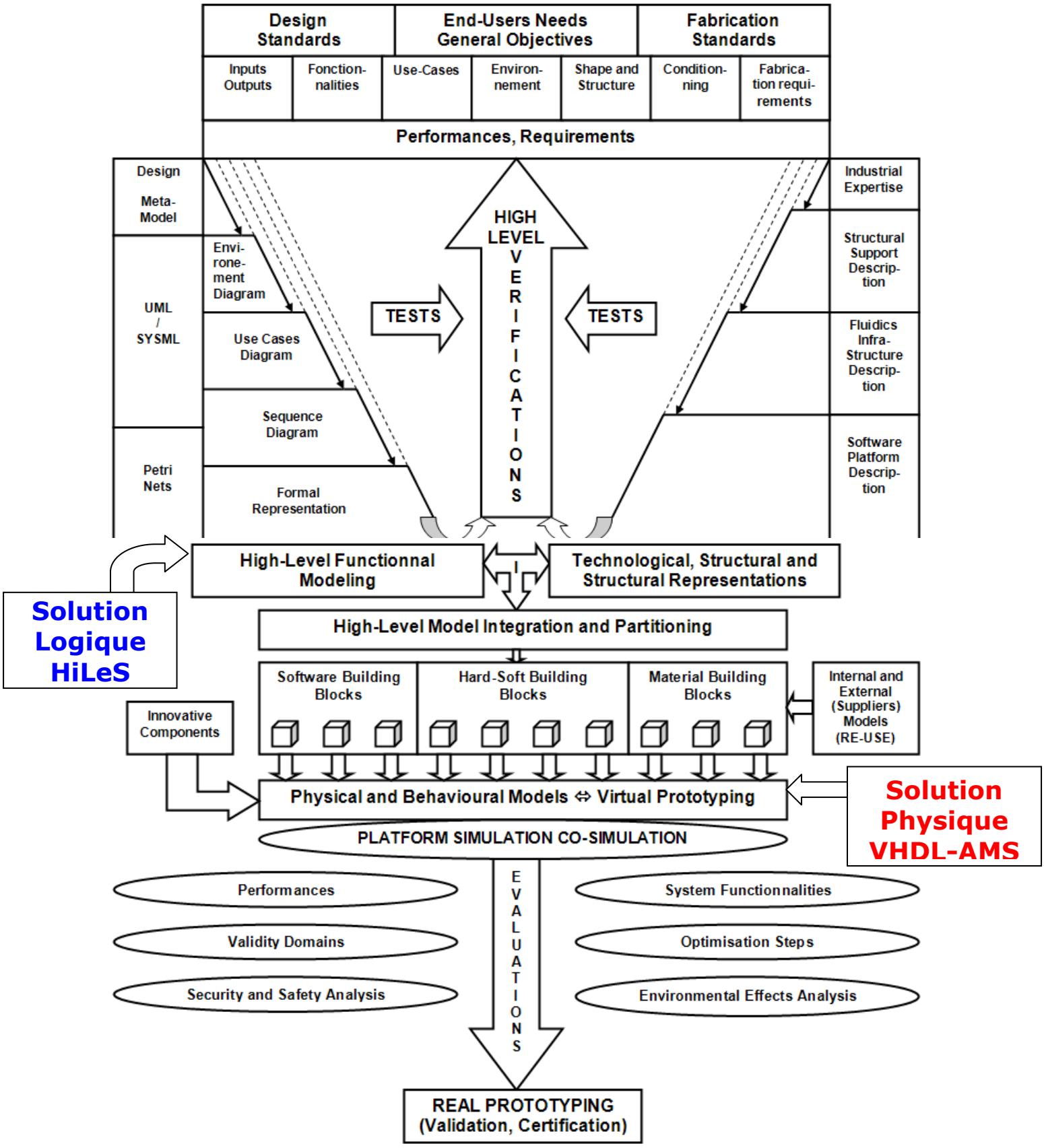


Figure 23 : Proposition détaillé de notre Processus de conception générale d'un système

1.7 Conclusion

Dans ce premier chapitre, nous avons cadré notre problématique de modélisation de système hétérogène au sein de l'ingénierie système dont les principes sont issus de la norme EIA-632. Nous avons montré que l'ingénierie système était une ingénierie guidée par les modèles avec notamment le MDA (Model Driven Architecture) préconisé par l'OMG (Object Management Group). C'est aussi une ingénierie multiprocessus, donnant notamment lieu à une solution logique et une solution physique. Notre position de travail se place très clairement au niveau de la création d'une solution physique que l'on associe au prototypage virtuel.

Cela nous a amené à aborder tout d'abord la modélisation système de manière générale en faisant état des différents modèles que l'on peut rencontrer et réaliser. Les modèles associés aux transformations de modèles, que nous avons aussi évoqués dans ce chapitre, définissent les processus élémentaires de la conception système. La transformation de modèles est donc un acteur important du processus de conception. Ce thème sera approfondi au cours du second chapitre, dans l'optique de converger vers la solution physique.

Le corps de ce premier chapitre nous a permis de souligner les qualités qui sont nécessaires au langage de modélisation pour être le support de la solution physique. Ainsi il a été souligné l'importance :

- d'avoir une norme pour satisfaire notamment la réutilisabilité (reuse),
- de pouvoir supporter une simulation ou si besoin une cosimulation de système, dont la modélisation peut avoir un caractère hétérogène, multi-abstraction, et multi-architectures,
- de pouvoir être interfacé facilement avec la solution logique,
- de pouvoir être la cible de transformations de modèle avec des modèles sources mettant en jeu des langages de modélisation différents,
- d'être compatible avec une gestion en base des données des modèles créés.

L'étude des différents langages de modélisation présentée dans le paragraphe 1.5, nous a permis de voir que le langage de description de matériel VHDL-AMS normalisé IEEE (IEEE 1076.1-1999) était un bon candidat comme langage de modélisation du prototype virtuel. Ce langage de modélisation nous permet en effet :

- De réaliser de modéliser des systèmes multi disciplinaires et multi abstractions.
- De mettre en œuvre si besoin des cosimulations,
- De permettre la généricité et la réutilisation des modèles dans d'autres contextes, grâce notamment au fait d'être une norme IEEE (IEEE 1076.1-1999),
- D'être interfacé facilement avec une solution logique. Des travaux de transformations automatiques ont été d'ores et déjà réalisés, mettant notamment en jeu le formalisme HiLeS support d'une solution logique et le VHDL-AMS [Ham05][Alb05][Mau05].

Nous tâcherons dans les chapitres suivants de valider notre choix de langage de modélsiation et d'étudier les moyens à mettre en œuvre pour y converger.

CHAPITRE 2

2. CREATION ET TRANSFORMATION DE MODELE

Au cours du premier chapitre, nous avons montré que notre problématique de modélisation système basée sur l'application de la norme VHDL-AMS, s'insérait parfaitement dans un processus de conception guidé par les modèles. Nous avons retenu comme norme de l'ingénierie système l'EIA-632, avec pour volonté de s'appuyer sur les recommandations de l'OMG en Ingénierie guidée par les Modèles (MDA/MDE). Dans ce contexte, notre proposition a été d'utiliser le VHDL-AMS comme le langage de modélisation et de simulation de la solution physique, au sens des recommandations de l'EIA-632. Dans cette optique, nous avons montré qu'il y avait des avantages fondamentaux très attractifs de pouvoir cosimuler directement des systèmes mixtes (numérique-analogique) et accueillir des modélisations de disciplines différentes : Electrique, Electronique, Thermique, Mécanique, Fluidique. Cette aptitude peut être étendue par des techniques de cosimulation outils (cas de Cosimate [Cos06] avec l'utilisation d'un bus de données) mettant en relation dynamique des simulateurs mettant en jeu des modèles écrits dans des langages différents (entre deux outils différents comme par exemple SystemVision [Sys50] et Matlab/Simulink [Mat] ou au sein d'un même outil comme SeamLess [Sea] avec une cosimulation VHDL et C).

Cependant nous n'aborderons pas dans ce travail de thèse l'utilisation du langage VHDL-AMS comme héritier du travail de modélisation amont, réalisé pour l'établissement de la solution logique en conception système selon l'EIA-632. Nous rappellerons néanmoins que les résultats obtenus par des travaux récents [Ham05] [Mau05] [Gui03] [Alb05], ont montré que dans la plateforme HiLeS [Ham05], qui propose une solution logique dans un formalisme à base de réseaux de Petri, la traduction automatique du système de commande en VHDL-AMS, pouvait être réalisée de différentes façons.

Il nous reste donc à montrer que le VHDL-AMS peut être un langage cible pour d'autres sources de modélisation, c'est-à-dire soit par modélisation directe, soit par modélisation en appliquant des règles de transformations à partir de modèles d'ores et déjà créés dans un autre cadre.

Si les modèles sont les principaux acteurs du processus MDA [MDA], la transformation de modèle en est le mécanisme principal [MCV05]. Il convient tout d'abord de définir les principales caractéristiques et les différents langages et outils qui supportent ces transformations. Cela va nous permettre grâce à cette taxonomie et cette présentation des différentes techniques de transformation, de choisir la transformation de modèle la plus adéquate à nos besoins. C'est l'objectif de ce chapitre qui :

- Précisera de façon plus approfondie la notion de Transformation de Modèles,
- Etablira le meta modèle du formalisme VHDL-AMS,
- Explorera les transformations de modèles au niveau meta modèle.

2.1 Taxonomie des Transformations de Modèles

Dans l'optique de décider quelle approche de transformation de modèle est la plus adaptée pour répondre à notre problème, nous allons tout d'abord nous intéresser aux différents types de transformations de modèle existants, pour ensuite en faire ressortir

les différentes caractéristiques et propriétés principales qui seront pour nous des critères de choix pour les transformations de modèle vers le modèle VHDL-AMS cible.

2.1.1 Types de Transformations

Les différents types de transformations que l'on peut rencontrer, dépendent de ce que l'on a en entrée et de ce que l'on souhaite obtenir en sortie. Dans la référence [MCV05], est tout d'abord précisé que, suivant que l'entité en entrée est un programme (code source, code machine) ou un modèle, on utilise le terme de transformation de programme ou transformation de modèle. Néanmoins, la transformation de modèle englobe la transformation de programme étant donné qu'un modèle peut avoir plusieurs niveaux d'abstraction pouvant aller d'un modèle d'analyse abstrait (en passant par un modèle plus concret de conception) pour aller vers un modèle très concret de code source. Ainsi les transformations de modèle incluent aussi les transformations entre des modèles de niveaux d'abstraction différents (design vers code ou code vers design dans le cas d'un projet de reverse engineering).

2.1.1.1 Transformation endogène et exogène

Dans le contexte de transformation de modèles, ceux-ci peuvent être exprimés par le biais de langages de modélisation (par exemple en UML ou en VHDL-AMS pour des modèles de conception, ou en assembleur pour des modèles de code source). La syntaxe et la sémantique des langages de modélisation sont exprimées par un méta modèle, concept défini dans l'introduction générale qui sera approfondi au cours du paragraphe 2.3. Prenant en compte ces langages de modélisation dans lesquels les modèles source et cible de la transformation sont exprimés, une première distinction doit être précisée entre les transformations endogènes et exogènes.

Les transformations endogènes mettent en jeu des modèles exprimés dans un même langage (possédant donc le même méta modèle). La Figure 22 représente ce type de transformation avec un modèle d'entrée et de sortie conformes à un même méta modèle.

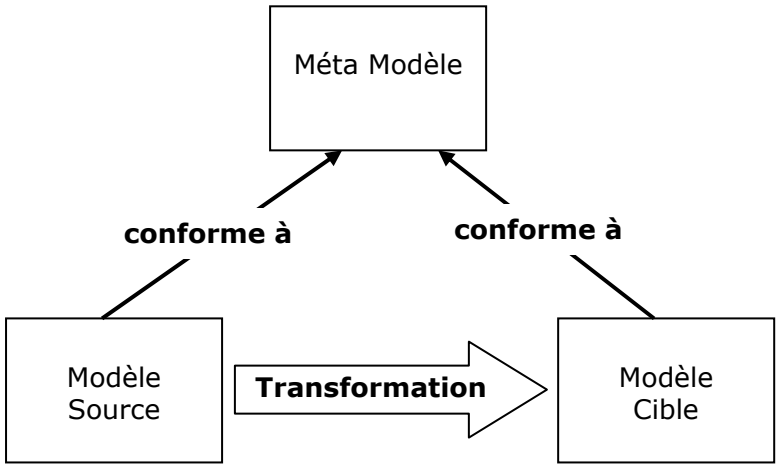


Figure 24 : Principe de la Transformation de Modèle Endogène

Il peut ainsi s'agir :

- D'une optimisation : la transformation a pour but d'améliorer par exemple la qualité d'exécution (en terme de performance) du modèle, tout en préservant la sémantique du modèle,

- D'une restructuration : la transformation entraine un changement de la structure interne du modèle afin d'améliorer la qualité de certaines caractéristiques comme la modularité et la réutilisation, sans modifier le comportement du modèle,

- D'une simplification : la transformation va permettre de simplifier la complexité de la syntaxe du modèle.

La transformation exogène, quant à elle, met en jeu des modèles exprimés dans des langages différents. La Figure 23 dépeint ce type de transformation avec les modèles d'entrée et de sortie ayant des meta modèles différents.

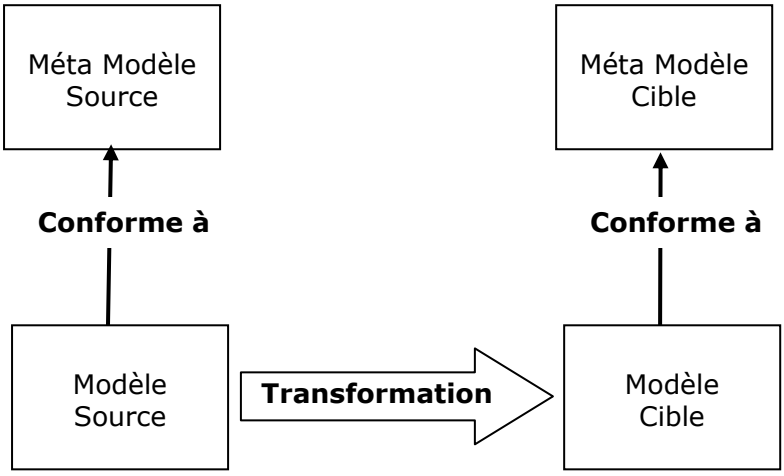


Figure 25 : Principe de la Transformation de Modèle Exogène

Comme pour la transformation endogène, on peut citer plusieurs exemples de transformation exogène :

- La migration : la transformation d'un modèle écrit dans un certain langage en un modèle écrit dans un autre langage, tout en gardant le même niveau d'abstraction.

- La synthèse : la transformation d'un modèle de haut niveau, très abstrait (spécification, modèle fonctionnel) vers un modèle de bas niveau plus concret (code généré, exécutable).

- Le « reverse engineering » : il s'agit d'une transformation de synthèse inverse qui permet d'extraire des spécifications de haut niveau d'un modèle de bas niveau.

2.1.1.2 Transformation horizontale et verticale :

Une deuxième distinction importante est faite entre les transformations. Elle se base cette fois-ci sur les niveaux d'abstraction des modèles sources vis-à-vis des modèles cibles. Il s'agit des transformations horizontales et verticales.

Une transformation horizontale présente des modèles sources et cibles ayant le même niveau d'abstraction. La Figure 24 dépeint ce type de transformation en représentant la fusion de deux modèles de même niveau d'abstraction afin d'obtenir un troisième modèle.

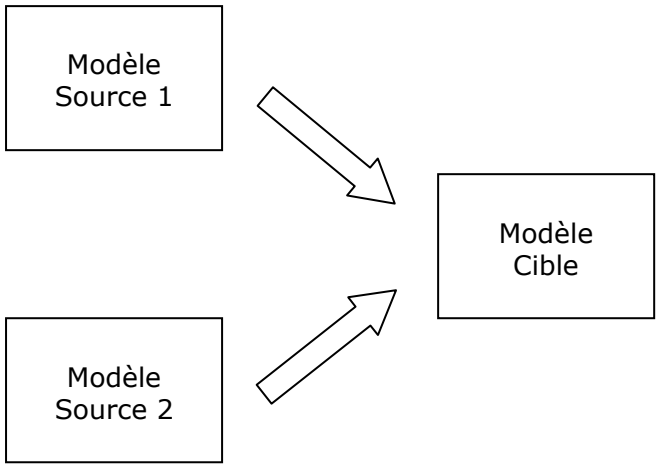


Figure 26 : Principe de la Transformation de Modèle Horizontale (Fusion de modèles)

Contrairement à une transformation horizontale, une transformation verticale présente des modèles sources et cibles de niveaux d'abstraction différents. La Figure 25 représente un exemple de transformation verticale que l'on a d'ores et déjà rencontrée, puisqu'il s'agit de la conception en Y à la base de la méthodologie MDA [MDA] proposée par l'OMG [OMG]. Elle met en jeu des modèles indépendants à une quelconque plateforme (PIM), des modèles décrivant une plateforme particulière (PDM), et qui par transformation verticale résultent d'un modèle spécifique à une plateforme (PSM).

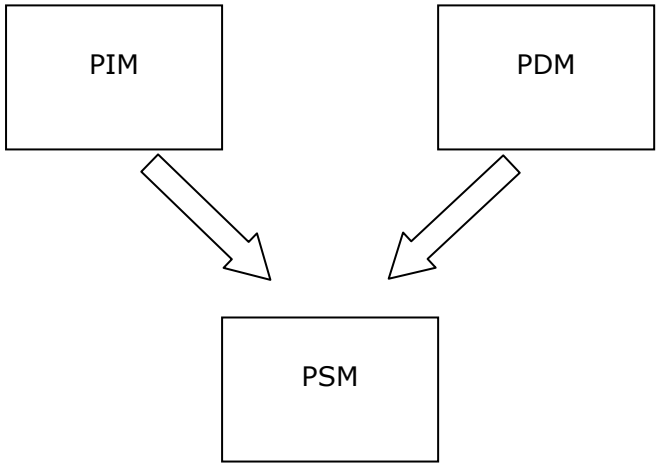


Figure 27 : Transformation de Modèle Verticale (PIM vers PSM)

On peut aussi constater, sur le Tableau 3, que de manière générale, les transformations de modèle possèdent une double dimensionnalité orthogonale entre la caractéristique Endogène/Exogène orthogonale et la caractéristique Horizontale/Verticale.

Transformation	Horizontale	Verticale
Endogène	Restructuration Simplification	Raffinement
Exogène	Migration de Langages Fusion de Modèles	PIM vers PSM Reverse Engineering Génération de Code

Tableau 3 : Double Dimensionnalité des Transformations de Modèles ([MCV05]et [Top06])

2.1.1.3 Conclusions pour notre problématique

Dans notre objectif de créer une modélisation d'un système hétérogène à l'aide du langage de modélisation VHDL-AMS, le modèle cible est bien déterminé. La caractéristique de la transformation à appliquer va donc dépendre essentiellement du ou des modèles sources présents en entrée.

Ainsi dans le cas où l'on part essentiellement des spécifications du système à réaliser, on se place dans le cas d'une transformation verticale exogène qui s'apparente à la transformation PIM vers PSM.

Pour les autres cas où l'on part d'un modèle créé dans un langage de modélisation donné, on se place alors dans une transformation de modèle s'apparentant à une migration de langage, c'est-à-dire horizontale et exogène. Au cours de notre projet, nous avons aussi rencontré le cas de figure d'avoir à transformer un modèle écrit dans le langage MAST (cf. 1.5.2.2) vers le langage VHDL-AMS. Nous nous attarderons plus précisément sur cette transformation au cours du chapitre 3 (cf. 3.4), mais l'on peut d'ores et déjà souligner que ces deux langages sont tous deux des langages de description de matériel possédant en grandes parties des concepts de modélisation commun. On peut donc dire que bien que leur syntaxe soit différente, leur sémantique est proche. La création d'un méta modèle commun aux deux langages regroupant la plupart de leurs concepts est possible, et donc dans ce cas de figure nous nous rapprochons très fortement d'une transformation de modèle endogène. Cette idée de méta modèle commun à tous les langages de description de matériel (MAST, VHDL-AMS mais aussi Verilog-AMS) sera développée au cours du paragraphe 2.4 et du chapitre 3 (cf. 3.4).

2.1.2 Propriétés et Caractéristiques

Après avoir défini les différents types de transformation de modèle, il convient de prendre en compte certaines propriétés et caractéristiques qui vont contribuer aux choix d'une transformation à appliquer à notre problématique plutôt qu'une autre.

Le premier aspect à prendre en compte est le niveau d'automatisation de la transformation. On distingue communément trois niveaux : automatique, manuel et

assisté. La complète automatisation d'une transformation est l'objectif, notamment car elle va répondre plus efficacement à des contraintes économiques en requérant moins de personnes en charge de la transformation et en engendrant un gain de temps. Ce gain de temps vient principalement du fait qu'une fois la transformation certifiée, il n'y aura plus besoin d'avoir des étapes de vérification du bon déroulement de la transformation ni d'étape de validation du modèle obtenu. Cependant il s'avère que bien souvent une intervention extérieure au processus (manuel) va être nécessaire pour spécifier et résoudre une ambiguïté, une imperfection, une contradiction qui peut survenir au cours d'une transformation ayant lieu par exemple entre les spécifications d'un système à modéliser et un modèle fonctionnel de ce système.

Dans notre cas, on se rend compte, que pour une transformation de type verticale qui caractérise notre processus de conception qui part des spécifications afin d'obtenir le modèle de notre système, une automatisation complète est utopique. En effet, un apport du concepteur en termes de choix d'architecture notamment, est nécessaire et est bien souvent résultant de l'expérience et de l'intuition qui ne sont pas modélisables.

Dans le cas où l'on part d'un autre modèle déjà créé où le travail de structure et de fonctionnalités a été déjà réalisé, il existe plusieurs méthodes qui seront plus ou moins automatisables. Lorsque le point de départ est le code du modèle source, si l'on adopte un portage sémantique qui va nécessiter la contribution d'un expert des deux langages, alors le niveau d'automatisation sera manuel voire assisté [VIP06]. De même, si l'on adopte un portage par identification qui va nécessiter la contribution d'un spécialiste des deux langages, le niveau d'automatisation sera aussi manuel voir assisté. Cela sera évoqué plus précisément dans les techniques de transformation que l'on utilisera notamment au niveau du chapitre 3 et de la transformation entre le MAST et le VHDL-AMS. Il existe cependant une autre alternative de portage que l'on a déjà évoquée, notamment dans l'introduction générale et qui se base sur les meta modèles. Nous en reparlerons dans le paragraphe 3.4 du chapitre 3, mais nous pouvons d'ores et déjà dire que cette approche est la plus à même d'être automatisable même si cela peut engendrer certaines pertes sémantiques (transformation exogène) et un code résultant lourd et pas forcément optimisé dans le langage source comme le pourrait être un code réalisé par un expert qui pourra profiter de toutes les possibilités de modélisation du langage cible.

Une seconde caractéristique à prendre en compte est la complexité de la transformation, qui sera faible par exemple dans le cas d'une restructuration (horizontale et endogène) mais qui pourra devenir plus complexe dans le cas d'un analyseur et un générateur de code. L'ensemble de techniques et d'outils à utiliser pour mettre en œuvre la transformation ne sera alors pas la même.

Enfin, la préservation des propriétés prouvées est aussi une caractéristique à prendre en compte car au sein d'un processus de conception guidé par les modèles tel que l'on a pu décrire au cours du premier chapitre, on trouve un enchaînement de plusieurs types de transformations et il va être important de s'assurer que certains aspects du modèle source vont être préservés dans le modèle cible [MCV05]. Par exemple, dans le cas de la restructuration d'un modèle, il va être nécessaire de s'assurer que le comportement global du modèle source sera préservé après la transformation. Plusieurs critères de transformations sont à prendre en compte : Réversibilité (Règle unidirectionnelles et bidirectionnelles), Traçabilité (Point de Contrôle), Incrémentabilité (modifications dans le modèle source qui impliquent la mise en jeu de seulement certaines transformations à appliquer sur le modèle cible), Réutilisabilité (Modularité).

2.2 Techniques de Transformations

Nous avons défini précédemment le type et les caractéristiques que l'on souhaiterait retrouver dans la ou les transformations de modèles répondant à notre problématique de modélisation de système hétérogène en VHDL-AMS. Il faut à présent choisir la technique de transformation de modèle à utiliser.

2.2.1 Méthodes Disponibles

Il existe aujourd'hui différentes méthodes de transformations mises en œuvre. Un travail effectué par le groupe WP5 du projet TOPCASED [Top06] s'est notamment penché sur une taxonomie des transformations et sur un état de l'art des différentes techniques de transformations possibles [Top06]. On retrouve ainsi les techniques de transformations suivantes :

2.2.1.1 Transformation d'arbres

Ce type de transformation se base sur le concept d'arbres. Ce concept est souvent utilisé dans le domaine des langages de programmation lors de la compilation, mais aussi dans le domaine de l'analyse de langage. Un arbre peut être qualifié d'arbre d'analyse (parse tree) (aussi appelé arbre de syntaxe concrète (concrete syntax tree)). Il permet alors de représenter la structure syntaxique d'une chaine de caractère en fonction d'une certaine grammaire formelle. On retrouve ce principe sur la Figure 26, avec par exemple l'arbre d'analyse linguistique d'une phrase en langue française. Le passage par un arbre pourra être une bonne alternative pour traduire un texte écrit dans un langage donné dans un autre langage (Français vers Anglais par exemple).

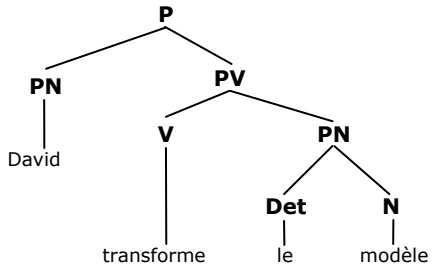


Figure 28 : Exemple d'arbre pour l'analyse d'une phrase en français

Ce type de transformation est aussi très utilisé dans le domaine de la programmation. Nous avons notamment rencontré au cours de notre travail un logiciel de création de modèle qui se basait sur ce principe. En effet ce logiciel Paragon [PAR04], que l'on évoquera plus longuement dans le paragraphe 2.2.2, possède une base de données de modèle en XML. Ces modèles peuvent être générés dans plusieurs langages et le principe de transformation se base sur les arbres. Ainsi dans cette approche, un arbre est généré à partir du document XML, ainsi que le code souhaité. On peut retrouver ces travaux de transformations explicités dans la thèse de V. Chaudhary [Cha02]. Ici le langage de programmation utilisé par le concepteur est le langage orienté objet Python [Pyt07]qui incorpore plusieurs mécanismes d'analyse de documents XML et notamment permet la manipulation d'arbre et la génération de code.

2.2.1.2 Transformation directe

Dans ce type de transformation, le processus de transformation est programmé explicitement, avec une manipulation directe du code source via un programme. Nous avons pour illustrer cela l'exemple du travail de V. Albert [Alb05], qui a écrit un programme informatique en java qui permet de manipuler des modèles de réseaux de Petri pour en générer des modèles VHDL équivalents. Le principe de sa méthode est

cependant basé sur le concept de meta modèle et de transformation au niveau meta modèle. En effet, il a créé le meta modèle du formalisme des réseaux de Petri ainsi que celui du VHDL. Toutes ces règles de transformations ont été écrites en Java. Ici les meta modèles ne servent que de référence pour le programmeur. Le programme ne manipule que les modèles d'entrée et de sortie. Nous verrons dans le paragraphe 2.2.3, qu'il existe d'autres approches de transformation basées sur les meta modèles qui nécessitent d'apporter les modèles d'entrée et de sortie mais aussi leur meta modèle respectif. Nous reviendrons sur la question des meta modèles au paragraphe 2.3.

2.2.2 Technique avec un Langage Pivot

Les différentes techniques explicitées précédemment se sont avérées relativement spécifiques à une transformation donnée entre deux types de modèle. La réutilisation de ces transformations dans un autre cas de figure nécessiterait de recommencer l'ensemble des travaux quasiment à zéro. Nous nous trouvons ici dans une transformation de type 1 vers 1. Cela peut devenir problématique si l'on doit transformer N langages vers N autres. On voit clairement ici qu'il serait très intéressant de pouvoir insérer un langage intermédiaire qui permettrait de décomposer les transformations de façon N vers 1 + 1 vers N. Ce langage intermédiaire est ce que l'on nomme le langage pivot. Il va donc permettre d'optimiser les processus de transformations entre plusieurs langages. On peut aussi souligner que le fait d'avoir un langage pivot commun va permettre d'optimiser certains outils notamment de vérification qui seront plus à même de manipuler certains langages. On peut notamment citer le langage XML qui permet le stockage de donnée par le biais de balises personnalisables. Le fait de converger vers ce langage permet d'avoir un certain nombre d'outils à disposition permettant de manipuler les modèles écrits dans ce langage.

Au cours de notre travail de transformation de modèles écrits dans un langage de description de matériel comme le MAST ou le VHDL-AMS, nous avons été amenés à évaluer différents outils pouvant contribuer à notre démarche. Dans ce cadre, nous avons rencontré un outil de transformation qui a un principe de fonctionnement s'apparentant à une technique de transformation basée sur un **langage pivot**. Cet outil est un projet universitaire nommé Paragon, qui est toujours en cours de développement. Cet outil a pour origine le laboratoire américain MSCAD (Mixed Signal Computer Aided Design) [MFC03] [FCM04]. Nous avons utilisé la version 1.1.3 incluant la fonctionnalité d'importation de modèle MAST mise à disposition en 2004. L'objectif de l'outil est de mettre à disposition un environnement de conception de modèles indépendant des langages de description de matériel (via une interface graphique intuitive) qui permet ensuite de les générer dans un langage description de matériel donné. Le principe de transformation de l'outil peut être apparenté à la démarche de transformation de modèle basée sur les méta modèles, comme annoncé dans le paragraphe 1.3.3. Il s'avère en effet qu'une base de donnée XML (eXtensible Markup Language) est générée automatiquement par Paragon à partir de modèles HDL en entrée via un interpréteur de code (cf. Figure 27). Chaque fichier XML doit correspondre à une grammaire précise définie pour chaque application dans une DTD (Document Type Definition). Celle-ci est en fait fortement liée au concept de méta modèle.

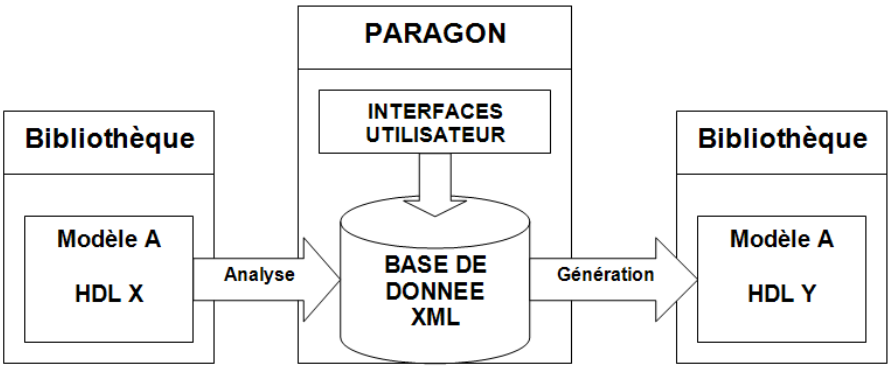


Figure 29 : Principe de Fonctionnement du logiciel Paragon

Dans le cas de Paragon, la transformation de modèle se fait par l'intermédiaire d'un meta modèle pivot (celui correspondant à la DTD de la base de donnée XML). Ce méta modèle reprend les différents concepts qui sont en commun entre les différents langages de description de matériel (entité, ports, paramètres génériques, équations, etc…).

Il faut aussi noter que pour le moment, seuls les modèles MAST analogiques peuvent être actuellement interprétés par Paragon. Certaines interprétations peuvent être imprécises, c'est pourquoi des étapes de vérifications doivent être insérées dans la méthodologie.

Quant à la partie de génération du code, comme évoqué dans le paragraphe 2.2.1.1, l'outil, écrit dans le langage de programmation Python, manipule la base de donnée XML pour la transformer en un arbre qui sera le point d'entrée du processus de génération de code. Il est à noter ici que quelque soit le langage de modélisation désiré en sortie, les différentes étapes de transformations (outils, langage pivot) sont les mêmes, mettant en avant l'avantage à utiliser une méthode avec langage pivot.

Cet outil fait parti de notre première méthodologie mise en place au début de nos travaux pour répondre à une problématique de création de modèle VHDL-AMS. Cette méthodologie sera explicitée en détail au sein du paragraphe 3.3 du chapitre 3.

2.2.3 Approche Hybride (ATL)

Développé depuis 2003, un outil libre et gratuit répond bien à la problématique de transformation de modèles au niveau des meta modèles. Il s'agit d'ATL (ATLAS Transformation Language) qui a pour vocation de permettre l'expression de règles de transformation de modèle et de pouvoir les exécuter. ATL a été développé au sein de l'Université de Nantes (INRIA) et plus particulièrement au sein du laboratoire de recherche LINA [ATL06]. ATL fait parti d'une plateforme nommée AMMA (ATLAS Model Management Architecture) qui regroupe non seulement ATL mais aussi trois autres projets en développement qui sont :

- AMW (ATLAS Model Weaver) pour le tissage entre modèle (Traçabilité),
- AM3 (ATLAS MegaModel Management Tool) pour la gestion des mega modèles qui spécifient les ressources disponibles (modèles, méta modèles, transformations de modèle, outils) au sein d'une plateforme de conception guidée par les modèles donnée,
- ATP (ATLAS Technical Projectors).

La plateforme AMMA ainsi que de nombreux concepts notamment la notion de langage spécifique à un domaine (DSL), sont explicités dans cette référence [BJK06].

ATL est intégré comme "plugin" à la plateforme de conception Eclipse [Ecl]. Au départ ATL a été intégré dans la partie d'Eclipse concernant les projets en développements (en incubation) sur la modélisation (GMT : Generative Modeling

Technologies). Néanmoins depuis le 15 janvier 2007, ATL fait maintenant parti de la section M2M d'Eclipse (Model-to-Model) et est donc recommandé en tant qu'outil de transformation modèle à modèle, preuve de sa maturité et de son nouveau statut d'outil industriel.

En tout cas, l'esprit MDA [MDA] qui nous anime, est parfaitement illustré dans ATL à travers la structure de son langage qui implémente notamment la norme QVT (Query/View/Transformation) proposé par l'OMG pour la transformation de modèle. De plus, l'approche de transformation de modèle est basée sur les règles de transformation définies au niveau méta modèle comme le montre la Figure 28 pris d'après ses spécifications. Les différents constituants de cette figure seront explicités dans le paragraphe suivant..

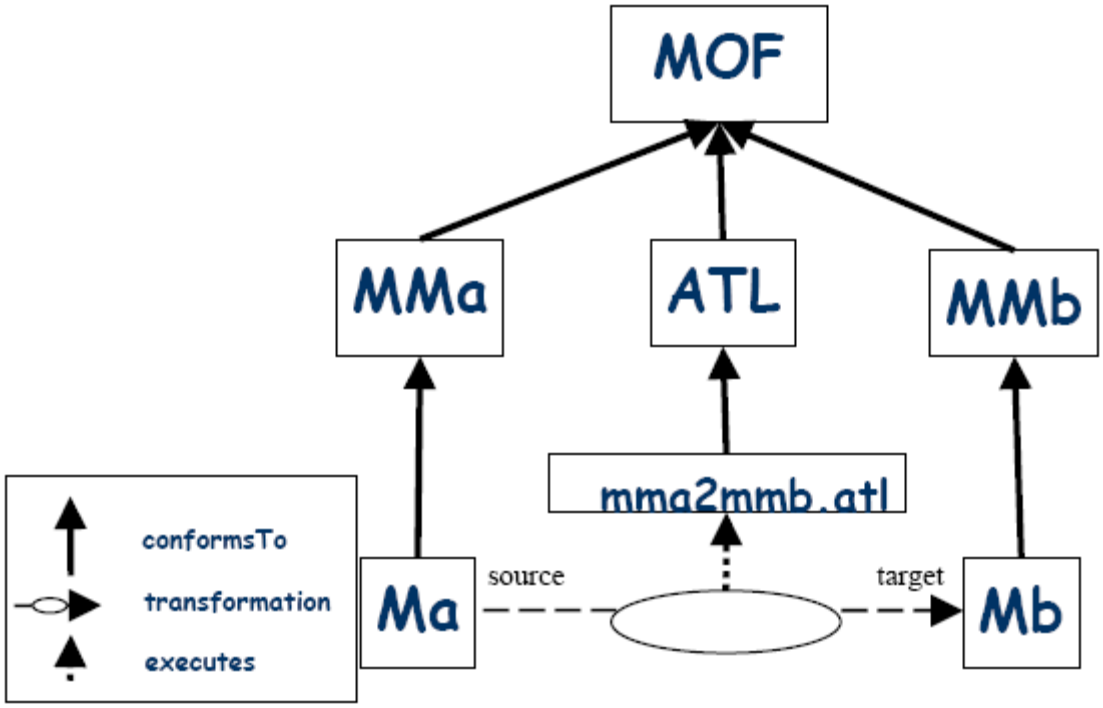


Figure 30 : Approche de Transformation de Modèle par l'outil ATL

Le caractère hybride d'ATL vient du fait que c'est un langage de transformation à la fois déclaratif et impératif. Ainsi il est possible d'écrire une règle de correspondance entre deux méta modèles (matched rules déclaratif)) mais aussi de faire appel à certaines règles au sein du code (called rules (impératif)). Néanmoins le style préconisé et le plus utilisé est le style déclaratif avec une expression simple du mapping entre les méta modèles d'entrée et de sortie.

ATL intègre le langage OCL (Object Constraint Language) qui est à la base un profil UML et qui fait parti du MOF afin d'aider à la création des meta modèles. Ce langage permet d'écrire sous forme de code des pré et/ou post conditions pour une opération et notamment d'insérer certaines contraintes à l'application d'une règle.

La transformation de modèles via des règles de transformations écrites au niveau meta modèle, peut être appliquée à notre problématique en utilisant notamment ATL comme langage de la transformation et les outils développés autour de ce langage permettant la réalisation de cette transformation (via un plugin sous la plateforme Eclispse). Cela nécessite la création de meta modèles adéquats qui sera explicitée dans les paragraphes suivants de ce chapitre 2. Il faut pour cela créer des règles de

transformation et utiliser la plateforme Eclipse, ce qui sera explicité dans le paragraphe 3.4.

2.3 La Conception de Meta Modèle

2.3.1 Définition d'un Meta Modèle

Un grand nombre de techniques de transformations de modèles, en particulier celles applicables au processus de conception guidé par les modèles normalisé par l'OMG [OMG] et dans lequel nous plaçons notre problématique, mettent en jeu le concept de meta modèle.

La première définition que l'on peut faire d'un meta modèle est celle de modèle de modèle. Il s'avère cependant que cela ne peut être valable dans toutes les situations. Un exemple souvent cité [Béz06] est le tableau de Magritte "La Trahison des Images (Ceci n'est pas une pipe) " représenté sur la Figure 29.

Figure 31 : Représentation du tableau de Magritte "La Trahison des Images (Ceci n'est pas une pipe) " (1928)

Cette figure représente le tableau qui à son tour représente l'objet réel. On peut alors remarquer que la figure est un modèle de modèle et donc suivant notre première définition un meta modèle. Cependant il manque un critère essentiel qui doit lier le modèle à son meta modèle, celui de la conformité.

Pour le tableau de Magritte, nous sommes situés simplement dans le cas d'une représentation d'une représentation. Le meta modèle va lui donner une signification au modèle. Un autre exemple simple de meta modèle est la légende d'une carte. Elle permet, en effet, de donner une signification à une carte routière de la France qui est une représentation de la France selon un certain point de vue (modèle géographique).

En d'autres termes, on peut dire qu'un système est représenté par un modèle, lui-même conforme à un meta modèle (cf. Figure 30).

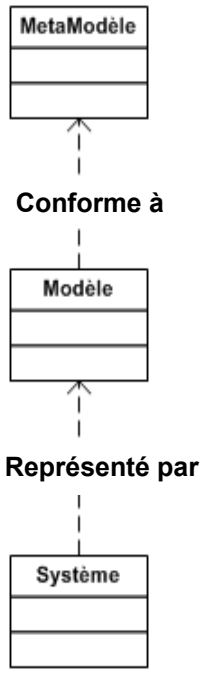


Figure 32 : Notion de Meta Modèle

Enfin, le choix d'identifier et d'associer un meta modèle au modèle exécutable, va contribuer utilement :

- A écrire un éditeur parfaitement rigoureux assurant la conformité du modèle vis-à-vis des règles de modélisation décrites dans le meta modèle,
- A permettre, ce qui nous importe ici, l'interopérabilité des modèles, en facilitant l'écriture de la transformation entre deux langages.

2.3.2 La notion de Meta Meta Modèle

Bien évidemment, on a le plus grand intérêt, du point de vue de l'interopérabilité, de mettre tous les outils d'un processus donné en conformité avec les mêmes références conceptuelles : c'est le rôle de la meta meta modélisation.

L'ingénierie dirigée par les modèles implique quatre niveaux de modélisation. Ils sont représentés sur la Figure 31 sous la forme d'une pyramide avec à la base le niveau M0 dédié au système à modéliser.

Le système est représenté par un modèle se situant au niveau M1. Suivant le point de vue adopté vis-à-vis du système à modéliser, ce modèle peut être une expression des spécifications du système via un diagramme de contexte UML, il peut aussi être une solution logique sous forme d'une schématique HiLeS ou une solution physique sous forme d'un modèle VHDL-AMS. Ces modèles sont réalisés en conformité avec les meta modèles respectifs des différents langages de modélisation utilisés. Ainsi on retrouve au niveau M2 les meta modèles d'UML, d'HiLeS et du VHDL-AMS dans notre exemple. Ces meta modèles sont à leurs tours conformes à un meta meta modèle occupant le niveau M4 et étant conforme à lui-même ne nécessitant aucun niveau de modélisation supérieur.

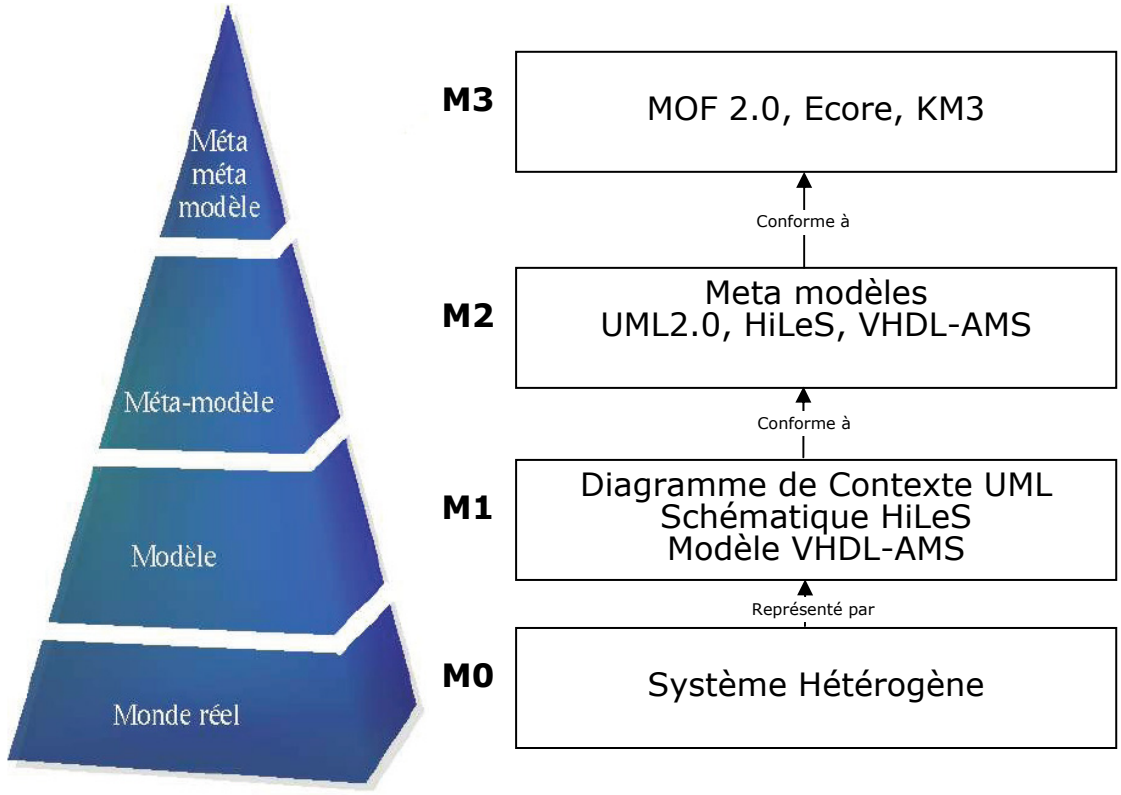


Figure 33 : Organisation Pyramidale du MDE (Model Driven Engineering)

L'OMG, dans le cadre de l'architecture dirigée par les modèles (MDA), a défini, comme meta meta modèle, le MOF (Meta Object Facility) dont la version la plus récente à la rédaction de ce mémoire est la version 2.0 [MOF06]. Dans le cadre plus général de l'ingénierie dirigée par les modèles dont le MDA en est une héritière, on trouve plusieurs espaces technologiques qui ne se résument pas au MDA et au MOF qui est communément appelé approche modèle (ModelWare) [Fav04]. Il existe aussi des espaces technologiques comme celui des grammaires (BNF Backus-Naur Form), des données (SQL), des documents (DTD pour document XML) par exemple. Dans chaque espace, les concepts de modèle, meta modèle et transformation ont une incarnation différente. Ainsi, ce qui est appelé meta modèle dans le MDA en conformité avec le MOF, correspond à une grammaire en conformité par la norme BNF. Le MOF, comme le BNF, est communément appelé langage spécifique à un domaine (DSL) que nous avons d'ores et déjà abordé dans le paragraphe concernant ATL (cf. 2.2.3) [BJK06].

Dans notre problématique de modèle de conception, nous nous plaçons dans l'espace technologique MDA. Il existe au sein de l'espace « ModelWare », plusieurs meta meta Modèle qui heureusement sont très proches, et les meta modèles qui leur sont conformes peuvent être transformés les uns vers les autres, via des langages de transformations comme ATL qui est parfaitement adapté à cela. Ainsi, outre la norme MOF, on peut citer la norme Ecore qui est définie comme le meta meta modèle du processus de modélisation de la plateforme de conception Eclipse [Ecl]. Un autre meta meta modèle, développé par les équipes responsables du langage de transformation ATL, est le KM3 qui s'avère être plus simple que les normes MOF et Ecore [JBK06]. En comparaison, le meta meta modèle KM3 est composé de 14 classes alors que pour l'Ecore on en distingue 18 et pour le MOF 1.4, 28. Ce meta meta modèle répond à une demande des utilisateurs qui avaient besoin de pouvoir créer leur propre méta modèle adapté à leurs applications et de pouvoir les manipuler facilement et de les transformer en d'autres méta modèles en utilisant des outils éprouvés comme ATL.

2.3.3 Création d'un Meta Modèle

Comme expliqué dans le paragraphe précédent, le meta modèle que l'on souhaite créer doit être conforme à un meta meta modèle. Pour notre problématique de création et de transformation de modèle, nous nous plaçons bien dans l'espace technique lié au modèle (« ModelWare »). Le meta meta modèle qui nous sert de base pour notre meta modèle est le MOF 2.0.

Le meta modèle modélise les différents concepts présents dans un langage de modélisation et les relations qui existent entre eux. Le meta meta modèle MOF met à notre disposition un certain nombre d'artefacts nous permettant de modéliser les différents concepts du langage de modélisation que l'on souhaite meta modéliser, ainsi que les relations qui relient ces différents concepts. La Figure 32 met en jeu un exemple simplifié permettant de voir comment se construit un meta modèle. Il représente ici un modèle UML avec simplement une classe « client » ayant un attribut « name » de type chaîne de caractère (string). Il représente le niveau M1, le niveau M0 étant le système réel c'est-à-dire ici le client à proprement parlé non représenté ici. Ce modèle UML met en jeu le concept de classe et le concept d'attribut. La relation entre ces deux concepts est qu'une classe peut contenir un ou plusieurs attributs. Le meta modèle simplifié d'UML que l'on peut voir sur la Figure 32 (M2), est conforme au MOF (M3) en cela que nous avons utilisé, pour créer notre meta modèle, deux classes (classe et attribut) et une association ici de type composition.

Nous allons principalement utiliser, pour modéliser nos meta modèles, les concepts de classe, d'attribut, et pour les associations celle de composition comme utilisé dans l'exemple de la Figure 32, mais aussi les associations de type généralisation (héritage).

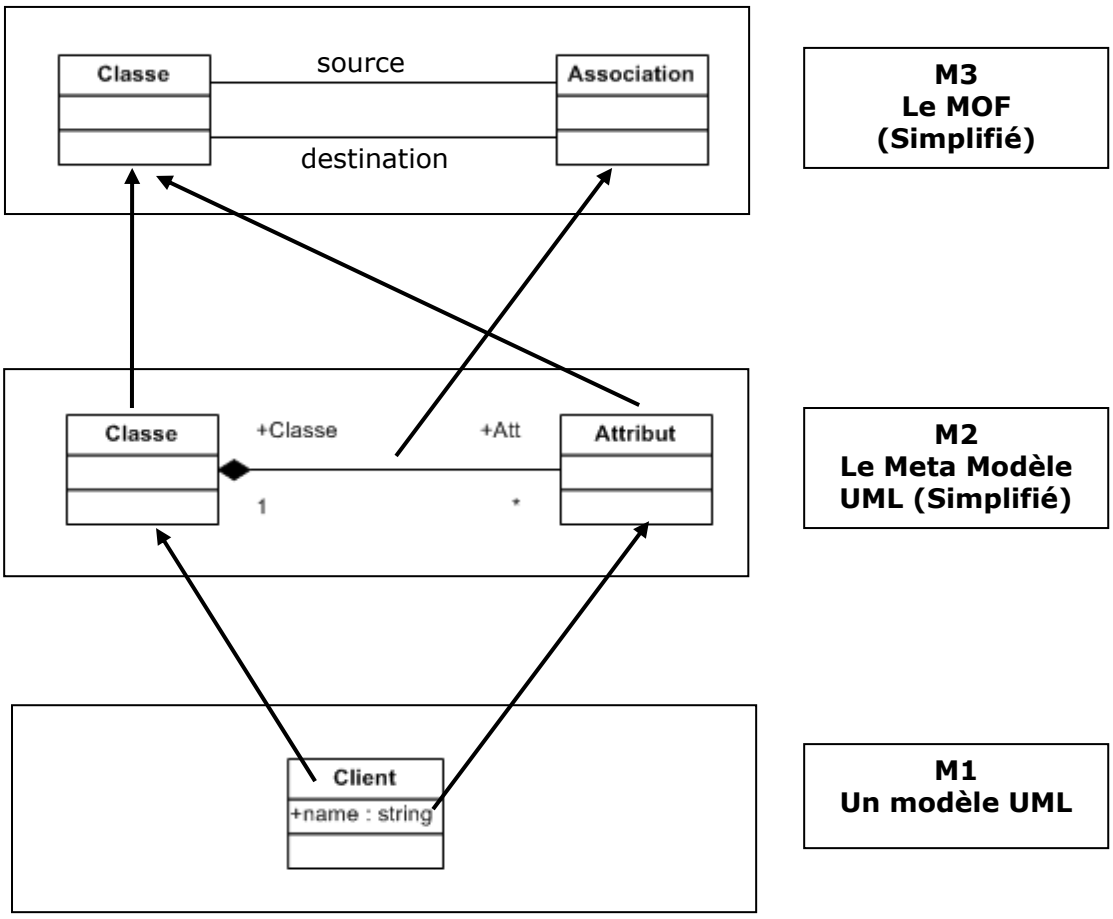


Figure 34 : Exemple simple de Meta Modélisation pour UML en conformité avec le MOF

2.3.4 Meta Modèle HiLeS

Nous allons appliquer la démarche précédente à un formalisme qui s'inscrit dans notre démarche générale de conception (représenté sur la Figure 21) pour y décrire la solution logique du système au sens de l'EIA-632. Il s'agit du formalisme HiLeS qui a été implémenté au cours de la thèse de J.C. Hamon [Ham05] sur des premières propositions de E. Jimenez [Jim00]. HiLeS est un outil candidat à la modélisation logique des systèmes. Son implémentation dans la plateforme TOPCASED [Top06] passe par l'établissement d'un meta modèle du formalisme et d'un éditeur spécifique.

Plusieurs constituants de base composent ce formalisme. Avec eux, le concepteur peut réaliser les descriptions de haut-niveau des systèmes modélisés avec des structures hiérarchiques et avec une représentation formelle de ces états de fonctionnement sous la forme de Réseaux de Petri, autrement dit, avec une architecture fonctionnelle dans laquelle sont bien différenciées les séquences de contrôle-commande et les fonctions exécutées à chaque état.

Un ensemble de blocs (structurels et fonctionnels) et des canaux de communication composent la description d'un système sous HiLeS (cf. Figure 33).

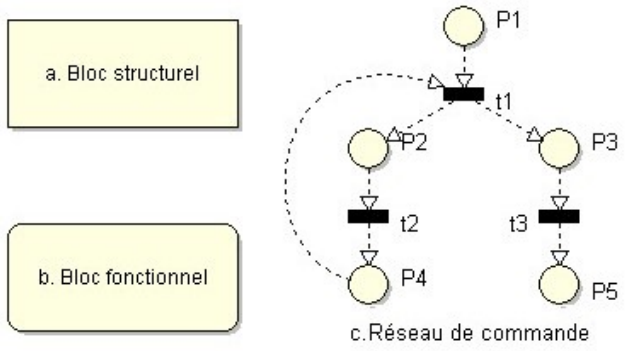


Figure 35 : les blocs de base du langage HiLeS et son modèle de commande

Les blocs structurels (cf. Figure 33a) (rectangles) permettent la décomposition structurale et hiérarchique du système. Ils admettent tout type de canaux d'entrée-sortie. Les blocs structurels peuvent contenir d'autres blocs structurels, des blocs fonctionnels ou des réseaux de commande organisés de façon hiérarchique. D'autre part, les blocs fonctionnels (cf. Figure 33b) (rectangles avec les coins arrondis) décrivent le comportement du système sous la forme de systèmes d'équations différentielles, algébriques ou logiques.

Le modèle de commande d'un système décrit HiLeS est basé sur des réseaux de Petri (cf. Figure 33c) ordinaires et temporisés. Le flot des jetons commande et séquence l'exécution des blocs structurels et fonctionnels. Le déclenchement d'une transition a une durée supérieure ou égale à zéro.

Les arcs des réseaux de Petri sont représentés par des flèches pointillées pour décrire le flot de commande. Les jetons utilisés ne sont pas colorés, indiquant qu'ils informent des activités événementielles mais qu'ils ne portent pas d'information.

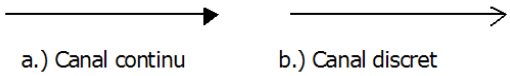


Figure 36 : Type de canaux définis dans HiLeS

Les blocs entre eux et leur réseau de contrôle sont reliés par des canaux. Les canaux transportent deux modes d'information: des modes continus (cf. Figure 34a) et des modes d'événement (cf. Figure 34b).

La Figure 35 présente l'architecture HiLeS par intégration du réseau de commande et de blocs. Les flots de données et de commande sont indépendants et concourants. Les réseaux de Petri représentent l'exécution des commandes de contrôle et la synchronisation des blocs. Des canaux discrets entrant ou sortant du réseau sont utilisés pour représenter l'interaction de commandes et de données.

L'interaction des réseaux de Petri avec des blocs fonctionnels décrit le comportement, l'exécution des instructions de contrôle–commande, et l'évolution des activités dans la structure hiérarchique du système.

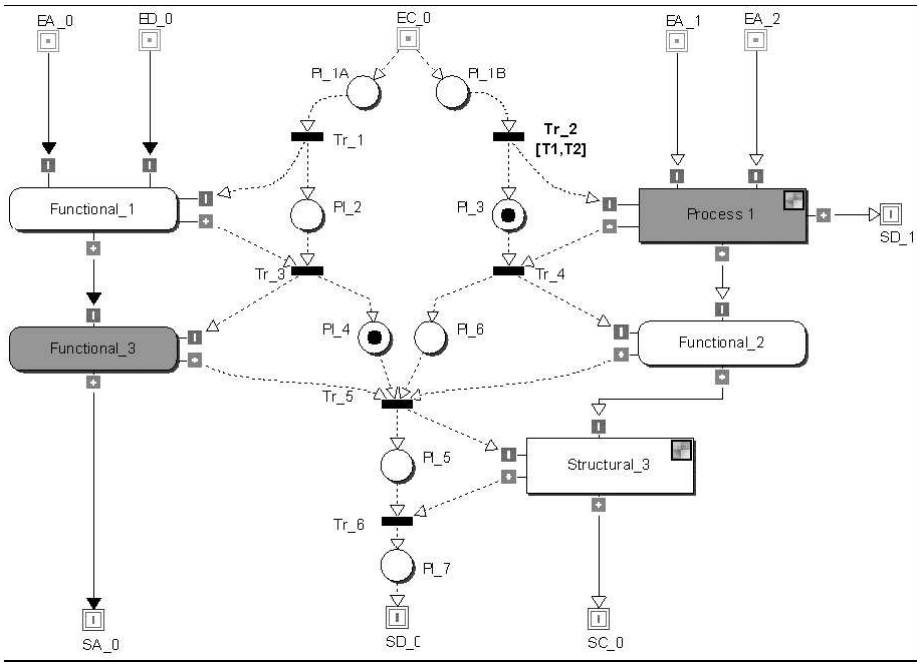


Figure 37 : Intégration du réseau de commande et blocs dans une architecture HiLeS

Avec ces concepts biens définis, ainsi que leurs relations, nous pouvons à présent créer un meta modèle de ce formalisme HiLeS en conformité avec le meta meta modèle MOF.

Notre proposition de meta modèle pour le formalisme HiLeS, est représentée sur la Figure 36. Le formalisme utilisé pour représenter ce meta modèle est le MOF. Ainsi les différents concepts utilisés dans HiLeS sont représentés sous forme de classe. On retrouve les différents types d'associations (notamment de composition et de généralisation) d'ores et déjà explicités dans le paragraphe 2.3.3. Nous retrouvons sur cette figure les différents concepts du langage HiLeS généralisés en trois grands concepts : les éléments de type bloc, les éléments de type liaison et les éléments de type réseau de Petri. Le meta modèle fait apparaître un certain nombre de liens entre les

différents concepts. Ainsi on s'aperçoit que les éléments de réseau de Petri ne peuvent être reliés à des canaux discrets ou continus, mais seulement à des arcs de réseaux de Petri. Il existe des contraintes qui ne sont pas représentées graphiquement sur ce meta modèle mais qui sont tout de même en vigueur. Elles sont exprimées via le langage OCL, profil UML que nous avons présenté dans le paragraphe 2.2.3. Dans notre cas il existe une contrainte, notamment sur le fait qu'un arc de Petri en relation d'une part avec un bloc, ne peut alors être en relation qu'avec une transition pour son autre extrémité.

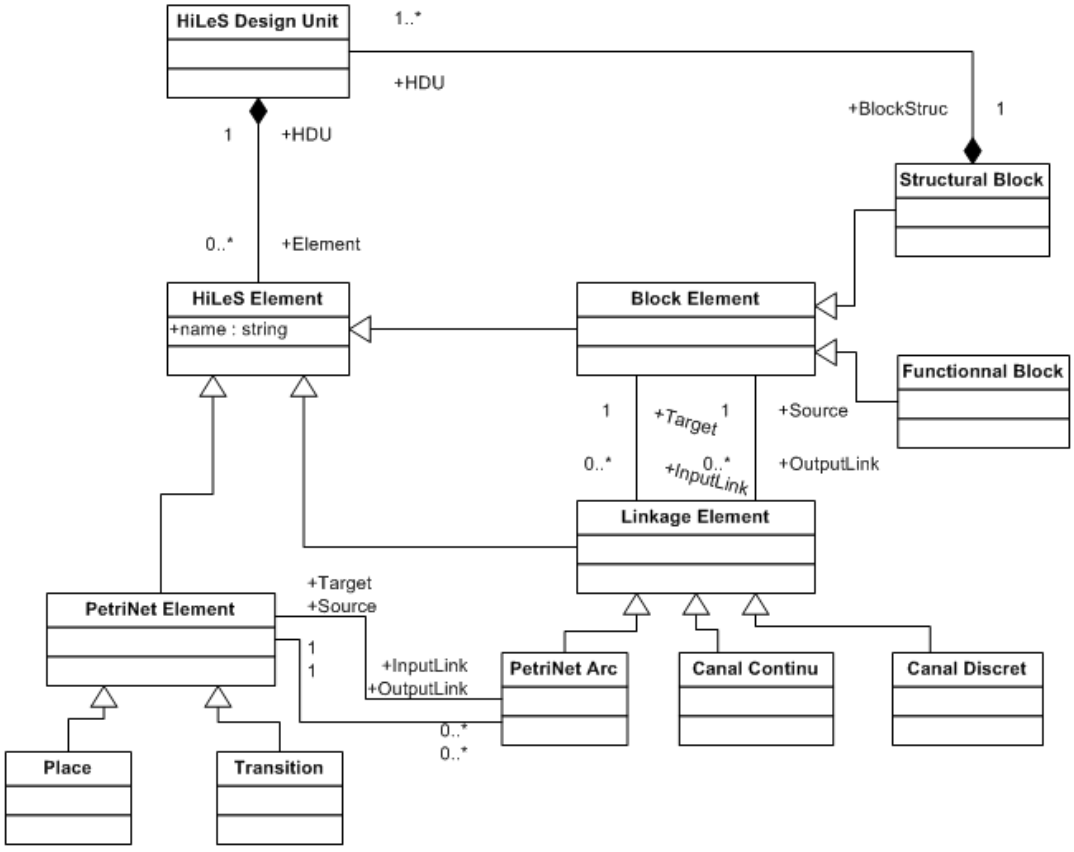


Figure 38 : Meta Modèle HiLeS

Une fois créé, ce meta modèle du langage HiLeS, va permettre :

- D'une part de pouvoir intégrer notre formalisme de modélisation au sein d'une plateforme comme Eclipse par le biais d'un éditeur basé sur ce meta modèle.
- D'autre part, HiLeS étant dans notre processus de conception, le support de la solution logique (selon l'EIA-632), ce meta modèle va servir de source dans notre problématique de passage de la solution logique vers la solution physique vers laquelle nous convergeons.

2.4 Proposition d'un Meta Modèle pour VHDL-AMS

Comme conclu dans le premier chapitre et introduit dans le deuxième chapitre, nous nous intéressons dans ce travail de thèse à la solution physique du système que l'on souhaite concevoir. Nous avons choisi le langage de description de matériel normalisé VHDL-AMS comme support de cette solution physique. Ce langage est donc notre langage cible pour les transformations que l'on souhaite mettre en place. On a pu tout au long de ce deuxième chapitre mettre en avant différents types de transformations, notamment la transformation basée sur les meta modèles. Cette

transformation est une des transformations que l'on souhaite mettre en place notamment dans le cadre d'un passage de la solution logique vers la solution physique, mais aussi dans le cadre d'une transformation entre un langage de description de matériel quelconque vers le VHDL-AMS. Afin de converger vers le langage VHDL-AMS et dans l'optique d'une transformation basée sur les meta modèles, il convient de faire le meta modèle de ce formalisme.

2.4.1 Etat de l'art Meta Modèle

De nombreux meta modèles ont été créés. On les trouve très souvent regroupés dans des bibliothèques de meta modèles appelées communément zoo. Il existe des zoos mettant en jeu des meta meta modèles différents (MOF, Ecore ou KM3). On peut par exemple citer les bibliothèques de meta modèles présentes dans le cadre du groupe projet AM3 (ATLAS MegaModel Management) de la plateforme Eclipse. On y trouve ainsi les meta modèles de formalismes tels que le langage C, Java, les réseaux de Petri, exprimés sous plusieurs meta meta modèles [Zoo07].

Malgré ces multiples meta modèles présents dans ces bibliothèques, il s'avère néanmoins qu'il n'existe pour l'instant pas de meta modèle du langage VHDL-AMS, ni de meta modèles d'autres langages de description de matériel (MAST, Verilog-AMS) dans ces bibliothèques. Néanmoins, des travaux ont déjà été réalisés dans ce sens concernant le langage VHDL (IEEE 1076-1993) avec notamment les travaux de V. Albert [Alb05], qui a réalisé le meta modèle du VHDL que nous pouvons voir sur la Figure 37, ainsi que le meta modèle des réseaux de Petri dans l'optique d'une transformation de modèle entre ces langages.

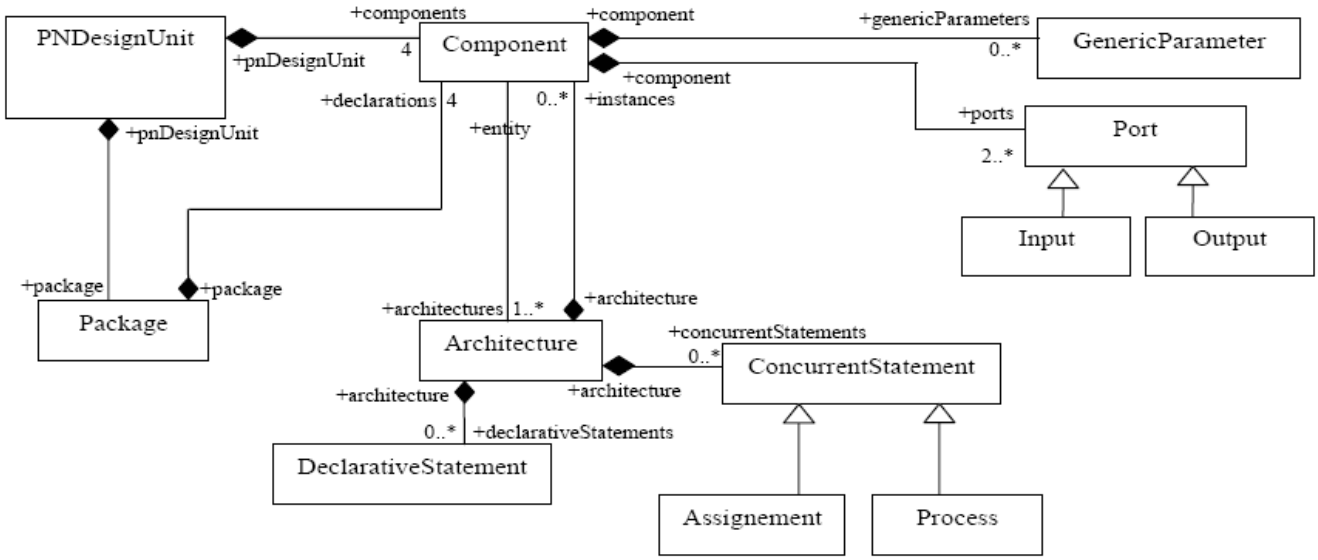


Figure 39 : Meta Modèle du formalisme VHDL proposé par V. Albert [Alb05]

Dans la même optique, on peut aussi citer les travaux de J. Delatour [SDC06], qui a aussi réalisé un meta modèle du langage VHDL (cf. Figure 38), dans le cadre de la conception d'un système numérique avec génération de code VHDL.

On peut toutefois constater que ces deux meta modèles du même langage ne sont pas rigoureusement identiques. Ils utilisent pourtant le même meta meta modèle mais celui-ci ne donne que les outils nécessaires à la représentation du meta modèle, il ne donne pas la méthode permettant de le construire. Il s'avère en fait que la création d'un meta modèle dépend de l'utilisation que l'on souhaite en faire. Ainsi dans le cadre d'une

transformation de modèle exogène, le meta modèle représentera les différents concepts du formalisme de façon plus ou moins complète. Dans le cas d'une transformation entre les réseaux de Petri vers le langage VHDL, seuls quelques concepts de modélisation du langage VHDL sont nécessaires. Ainsi on peut simplifier le meta modèle de ce langage. Dans un autre contexte, par exemple dans le cadre de la conception d'un éditeur de modèle, le meta modèle sera par contre le plus complet possible et le plus représentatif de la norme du langage.

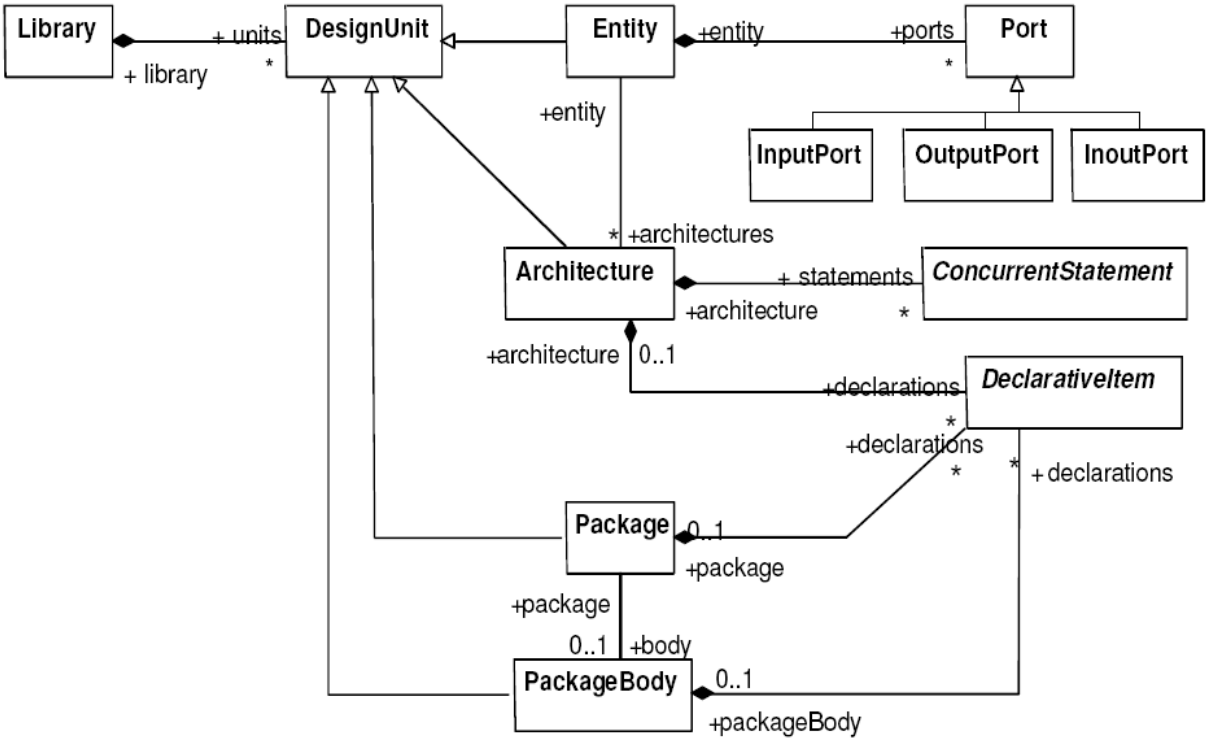


Figure 40 : Meta Modèle du formalisme VHDL proposé par J. Delatour [SDC06]

Le meta modèle du langage VHDL-AMS que l'on souhaite définir, va avoir une utilisation multiple. Dans un premier lieu, il permettra de vérifier la conformité d'un modèle VHDL-AMS que l'on aura créé. Dans un deuxième lieu, avec notre optique de convergence vers ce langage pour la solution physique, le langage sera la cible de la transformation de modèles écrits dans d'autres langages de description de matériel et de la solution logique. Ainsi le meta modèle sera utilisé dans un cadre de transformation de modèle avec définition de règles de transformations au niveau meta. Nous allons dans la suite de ce paragraphe évoqué la création d'un meta modèle de ce langage dans le cadre d'une vérification de conformité de modèles VHDL-AMS. Pour le cas d'utilisation pour une transformation donnée, cela sera abordé au cours du chapitre 3, dans le paragraphe 3.2 traitant du passage entre la solution logique et la solution physique, et dans le paragraphe 3.4 décrivant la transformation entre un langage de description de matériel particulier vers le VHDL-AMS.

2.4.2 Création du meta modèle VHDL-AMS

Dans le paragraphe 2.3.4, nous avons explicité notre proposition de meta modèle pour notre formalisme support de la solution logique : HiLeS. Pour notre meta modèle du langage VHDL-AMS, nous allons reprendre le même meta meta modèle, à savoir MOF. Etant donné que nous sommes cette fois-ci en présence d'un langage de modélisation normalisé (IEEE 1076.1-1999), notre point de départ pour définir les différents concepts de modélisation du langage et leurs interactions va être la norme officielle IEEE qui est en accès libre [Vhd99], ainsi que sa grammaire syntaxique concrète exprimée suivant la norme BNF que l'on peut trouver notamment dans les références [Vhd00] et [Her00].

Contrairement au formalisme HiLeS, le VHDL-AMS est un langage de programmation complètement syntaxique, ce qui va impliquer une démarche de création de meta modèle différente. Des travaux ont d'ores et déjà été réalisés sur le meta modélisation de langages de programmation par A. Sebatware [Seb06] et J. Malenfant [Mal02]. Nous allons appliquer dans un premier temps cette démarche pour réaliser notre meta modèle.

Un langage de programmation est défini principalement par sa syntaxe. Les règles syntaxiques et les règles lexicales sont exprimées à l'aide d'une grammaire. La norme BNF est utilisée pour exprimer les grammaires. Il existe deux types de syntaxe pour les langages de programmation : la syntaxe concrète relative à la forme et la syntaxe abstraite relative au fond.

Ce point de départ est la grammaire de la syntaxe concrète du langage. Ce point de départ ne parait pas incongru car comme nous l'avions précisé dans le paragraphe 2.3.2, la grammaire est le pendant du meta modèle dans un autre espace technologique, dans lequel le meta meta modèle est la norme BNF (Backus Naur Form) qui elle est le pendant du MOF.

Pour illustrer la démarche de conception de notre meta modèle, nous allons nous appuyer sur l'exemple du modèle d'une résistance électrique en VHDL-AMS, présenté sur la Figure 39. On retrouve les mots clés du langage (en bleu). Les ports d'entrée/sortie et les paramètres génériques sont définis au niveau de l'entité. La partie comportementale avec l'équation physique qui régit la résistance est définie dans l'architecture. Il s'agit ici de la syntaxe concrète.

```
library IEEE;
use IEEE.electrical_systems.all;

entity resistor is
  generic (     res : resistance);
  port (        terminal p1, p2 : electrical);
end entity resistor;

architecture ideal of resistor is
  quantity v across i through p1 to p2;
begin
  v == i*res;
end architecture ideal;
```

Figure 41 : Modèle d'une Résistance en VHDL-AMS

Un certain nombre de règles syntaxiques ont été nécessaires pour écrire correctement ce modèle VHDL-AMS, référencées dans la grammaire concrète du langage. Nous avons puisé dans la grammaire BNF du VHDL-AMS (cf. Tableau 4) les différentes règles nécessaires à la réalisation de la partie comportementale (architecture) du modèle ci-dessus. Les règles sont écrites en gras, avec en italique les différents mots clés du langage relatifs à la syntaxe concrète du langage. D'après la norme BNF, les crochets permettent de connaître les membres facultatifs et les barres verticales symbolisent des OU logiques.

architecture_body	::=	*architecture* **identifier** *of* **entity_name** *is* **architecture_declarative_part** *begin* **architecture_statement_part** *end;*
architecture_declarative_part	::=	{ **block_declarative_item** }
block_declarative_item	::=	**subprogram_declaration** | **subprogram_body** | **type_declaration** | **subtype_declaration** | **constant_declaration** | **signal_declaration** | **shared_variable_declaration** | **file_declaration** | **alias_declaration** | **component_declaration** | **attribute_declaration** | **attribute_specification** | **configuration_specification** | **disconnection_specification** | **step_limit_specification** | **use_clause** | **group_template_declaration** | **group_declaration** | **nature_declaration** | **subnature_declaration** | **quantity_declaration** | **terminal_declaration**
quantity_declaration	::=	**free_quantity_declaration** | **branch_quantity_declaration** | **source_quantity_declaration**
branch_quantity_declaration	::=	*quantity* [**across_aspect**] [**through_aspect**] **terminal_aspect** ;
across_aspect	::=	**identifier_list** [**tolerance_aspect**] [*:=* **expression**] *across*
through_aspect	::=	**identifier_list** [**tolerance_aspect**] [*:=* **expression**] *through*
identifier_list	::=	**identifier** { , **identifier** }
architecture_statement_part	::=	{ **architecture_statement** }
architecture_statement	::=	**simultaneous_statement** | **concurrent_statement**
simultaneous_statement	::=	**simple_simultaneous_statement** | **simultaneous_if_statement** | **simultaneous_case_statement** | **simultaneous_procedural_statement** | **simultaneous_null_statement**
simple_simultaneous_statement	::=	**simple_expression** == **simple_expression** [**tolerance_aspect**] *;*

simple_expression	::=	[**sign**] **term** { **adding_operator term** }
term	::=	**factor** { **multiplying_operator factor** }
factor	::=	**primary** [** **primary**] | *abs* **primary** | *not* **primary**
primary	::=	**name** | **literal** | **aggregate** | **function_call** | **qualified_expression** | **type_conversion** | **allocator** | (**expression**)
name	::=	**simple_name** | **operator_symbol** | **selected_name** | **indexed_name** | **slice_name** | **attribute_name**
simple_name	::=	**identifier**

Tableau 4 : Grammaire Concrète VHDL-AMS (Norme BNF)

Nous avons déduit de ces différentes règles syntaxiques un arbre syntaxique appelé arbre de dérivation. Les meta symboles de la norme BNF (crochet, barre,…), ainsi que les mots clés spécifiques au langage, peuvent être supprimés pour donner lieu à un arbre de dérivation représentant la syntaxe abstraite des règles. Dans la Figure 40, nous avons représenté l'arbre syntaxique abstrait des différentes règles mentionnées dans la figure précédente en l'appliquant au modèle de la résistance électrique. Lorsque que dans la règle plusieurs choix sont possibles, seul le choix utilisé dans le modèle est représenté dans l'arbre.

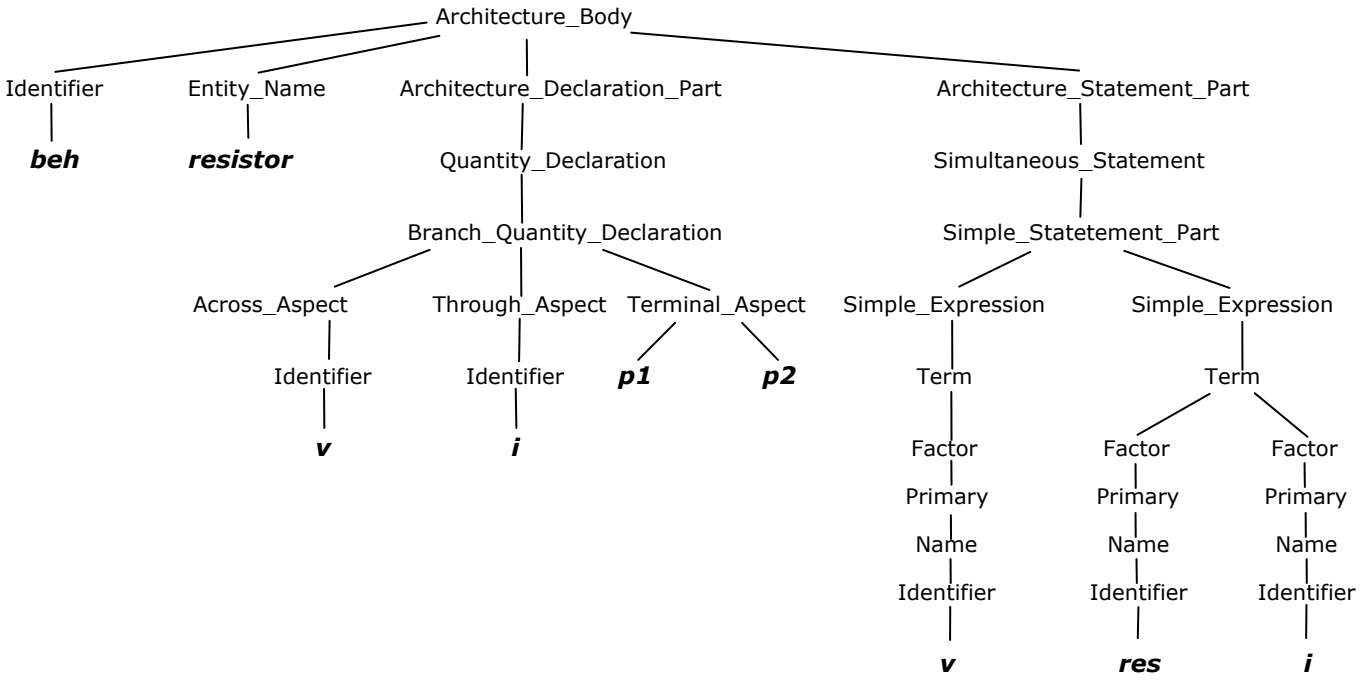


Figure 42 : Arbre syntaxique abstrait ou arbre de dérivation (appliqué au modèle de la résistance)

Le passage entre la syntaxe abstraite et le meta modèle du langage a été démontré dans le travail de A. Sebatware [Seb06], s'appuyant sur les travaux de J. Malenfant [Mal02] qui a montré que l'on pouvait représenter la syntaxe de manière fonctionnelle en utilisant notamment la notion de domaine syntaxique correspondant aux types de nœuds

que l'on retrouve dans l'arbre de la syntaxe abstraite. La démarche consiste à définir une classe MOF pour chaque domaine syntaxique. Pour chaque règle, un ET logique va être transformé en concept de composition conforme au MOF, tandis que le OU logique va être transformé en concept de généralisation conforme au MOF.

Conformément à cette démarche, notre proposition de meta modèle VHDL-AMS est représentée sur la Figure 41 de manière non exhaustive, certaines classes n'étant pas ici développées par un souci de clarté.

Cependant l'application systématique de cette démarche entraine des meta modèles relativement lourds et souvent moins faciles à manier par « l'humain » que les grammaires abstraites correspondantes [Mal02]. Nous avons développé ce meta modèle dans un souci d'étude de conformité des modèles créés en reprenant les normes utilisées dans l'ingénierie des modèles dans lequel nous plaçons notre démarche de conception. Ce meta modèle pourra ainsi être manipulé par des outils ou des plateformes répondant à des normes identiques tels que la plateforme Eclipse [Ecl], ou encore le projet Topcased [Top06].

Nous verrons néanmoins au cours du troisième chapitre (cf. 3.4), que ce meta modèle n'est pas en l'état très adapté à une utilisation dans un cadre de transformation de modèle avec écriture de règles au niveau meta modèle. Il sera donc modifié pour répondre à cette problématique, notamment de transformation entre langages de description de matériel.

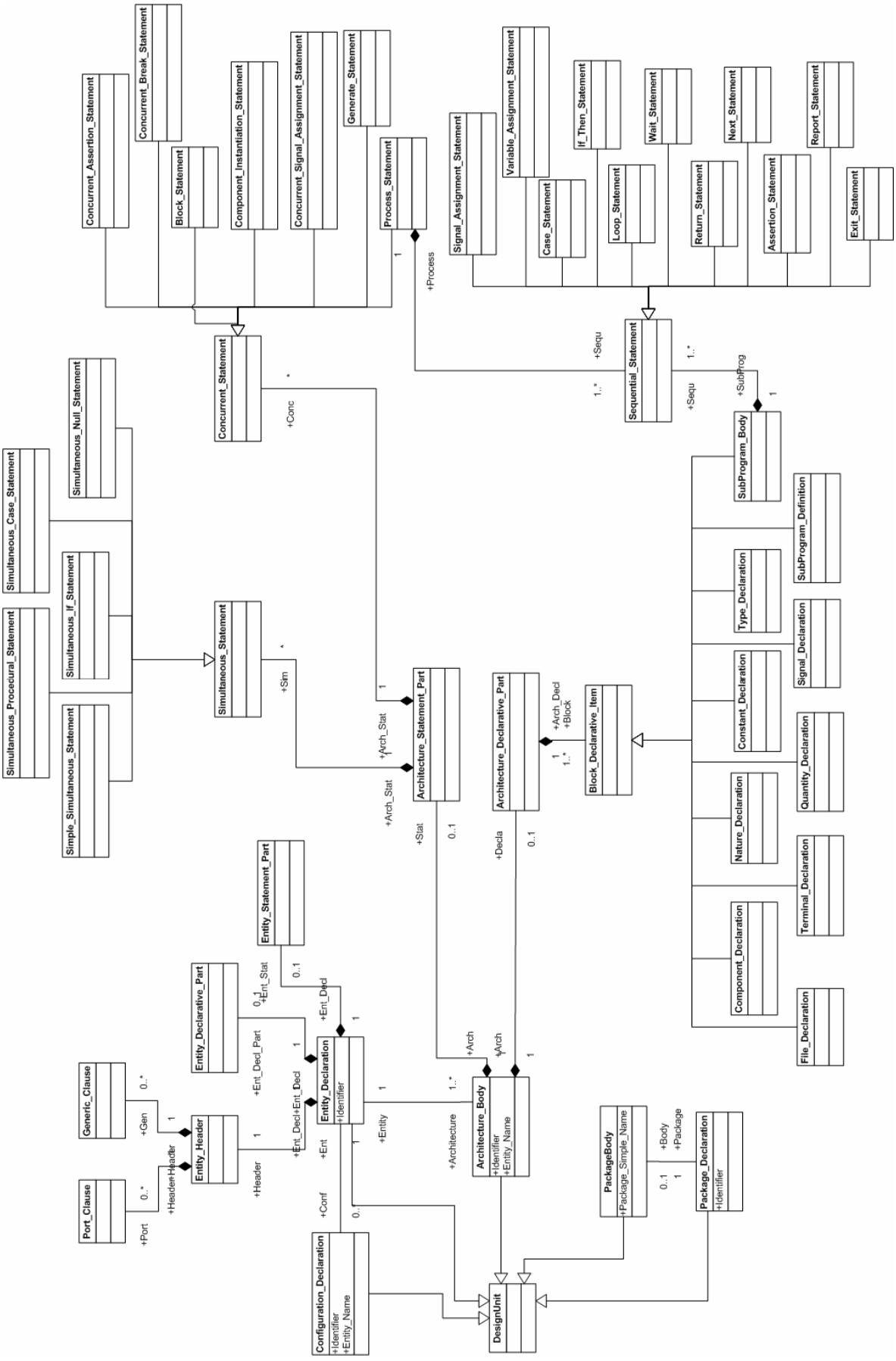


Figure 43 : Meta Modèle VHDL-AMS (premiers niveaux) (conforme au MOF)

2.5 Conclusion

Comme nous l'avons indiqué au cours du premier chapitre, notre problématique de création de modèle de système hétérogène se place dans un contexte d'ingénierie guidée par les modèles. Ce processus se base sur le concept de transformation de modèle présenté dans ce deuxième chapitre. Notre point de convergence est la création d'un modèle de notre système dans le langage de description de matériel standardisé IEEE 1076-1.1999 VHDL-AMS, qui est notre candidat pour être le support de la solution physique, au sens des recommandations de la norme EIA-632.

Dans cette optique, nous avons étudié dans ce deuxième chapitre, les propriétés et caractéristiques essentielles des transformations, ainsi que les différentes techniques possibles qui peuvent être mises en place. Cela nous a permis d'identifier au mieux ce qu'il nous fallait pour répondre à notre problématique :

Il ressort que suivant les cas que nous allons rencontrer, nous allons avoir des propriétés de transformations différentes. Ainsi dans le contexte d'un passage de la solution logique vers la solution physique, nous nous retrouvons ici avec une transformation de type vertical avec un caractère exogène. Cependant ce n'est pas le seul cas de figure que nous pouvons rencontrer lors de l'étape de convergence vers la solution physique. En effet il se peut que l'on ait aussi à faire migrer un certain nombre de modèles d'ores et déjà écrits dans un langage de description de matériel quelconque vers le VHDL-AMS. Dans ce cas là, la transformation est de type horizontal et exogène, c'est-à-dire qu'il n'y a pas d'apport d'informations au niveau du modèle (enrichissement). La transformation est exclusivement une migration de modèle entre deux langages. Il est aussi ressorti de cette étude que pour notre problématique, il était important qu'il ressorte des caractéristiques de traçabilité des exigences, mais aussi que le travail de création puisse être pérennisé en apportant une propriété de réutilisabilité.

Concernant les techniques de transformations étudiées dans ce second chapitre, deux peuvent répondre à notre attente :

La première transformation que l'on peut qualifier de directe, met en jeu notamment un langage pivot. Nous l'avons appliquée dans une première approche que nous avons mise en place au début de nos travaux de thèse, qui sera explicitée au troisième chapitre (paragraphe 3.3). Elle utilise un outil nommé Paragon, introduit dans ce chapitre, et qui se base sur le langage pivot XML.

La deuxième transformation pouvant répondre à nos attentes, est basée sur les meta modèles et implique l'écriture de règles au niveau meta modèle. Des solutions permettant de mettre en place cette transformation existent déjà, nous avons notamment étudié dans ce chapitre le langage de transformation ATL. Dans cette optique, nous nous sommes intéressés dans le chapitre à la notion de meta modèle, que nous avons définie, puis appliquée à notre support de la solution logique HiLeS et à notre support de la solution physique VHDL-AMS. Les meta modèles de ces formalismes ont été proposés qui vont pouvoir être utilisés, outre dans un cadre de transformation que l'on abordera dans le troisième chapitre, mais aussi dans un cadre d'étude de conformité des modèles créés, pouvant notamment servir de base à des éditeurs de modèles dédiés.

CHAPITRE 3

3. INTERETS DES TRANSFORMATIONS VERS LE VHDL-AMS

Nous avons vu, dans le premier chapitre, les possibilités de modélisation et les caractéristiques de simulation mixte qu'offre le langage de description de matériel VHDL-AMS et les outils qui lui sont dédiés. Au sein du deuxième chapitre, nous avons notamment élaboré un meta modèle de ce langage conforme au meta meta modèle MOF. Cela va nous permettre non seulement de vérifier la conformité des modèles que l'on crée en VHDL-AMS, mais aussi de pouvoir l'utiliser dans le cadre de notre processus de conception en l'intégrant aux transformations de modèles permettant de converger vers ce formalisme. Nous sommes donc prêts à écrire des modèles directement dans ce langage, pour faire migrer des modèles vers ce langage et pour vérifier leur conformité vis-à-vis du meta modèle.

Comme nous l'avons explicité au cours des deux chapitres précédents, une étape de notre problématique est de **proposer une solution physique** dans le cadre d'un processus complet de conception d'un système hétérogène. Dans cette optique, notre proposition est maintenant de considérer le langage VHDL-AMS comme langage cible et support privilégié de ce prototypage virtuel.

Pour ce faire, nous allons dans ce chapitre, tout d'abord montrer l'intérêt d'utiliser le VHDL-AMS comme support pour la solution physique, puis nous appliquerons les concepts de transformation et de création de modèles, explicités dans les chapitres précédents en ayant pour cible la solution physique en VHDL-AMS. Nous aborderons les différentes approches que l'on peut mettre en jeu et nous en étudierons la faisabilité et l'efficacité dans l'optique d'une application industrielle impliquant de nombreuses contraintes en termes d'automatisation, de vérification, de déploiement, et d'efficacité.

Ainsi nous aborderons dans ce chapitre les différents cas de figures de création de modèle VHDL-AMS que nous pouvons rencontrer au sein du processus de conception, à savoir :

- L'héritage de la solution logique dans la solution physique,
- La création du modèle à partir des spécifications,
- La transformation de modèle horizontal exogène entre un langage de description de matériel quelconque et le VHDL-AMS.

Nous conclurons finalement ce chapitre sur les aptitudes du langage VHDL-AMS à être le support du prototypage virtuel et sur la stratégie que cela implique de mettre en place.

3.1 VHDL-AMS support de la solution physique

L'objectif de ce paragraphe est d'illustrer la capacité du VHDL-AMS comme support de la solution physique pour modéliser des systèmes hétérogènes. Dans cette optique, nous nous appuyons sur l'exemple d'un dispositif multidisciplinaire de mise à feu sécurisé développé au LAAS-CNRS entre autres par P. Pennarun [Pen06] et C. Rossi [Ros97], qui doit être intégré dans un système de mise à feu de missile. Ce choix est dicté par la proximité du sujet mais aussi par souci de continuité avec la publication de F. Pêcheux, C. Lallement et A. Vachoux [PLV05] sur la modélisation d'un système d'Airbag, qui s'étaient déjà donné pour ambition de traiter la pluridisciplinarité des langages de modélisation VHDL-AMS et Verilog-AMS.

3.1.1 Description du système à modéliser

L'objectif est ici de faire une modélisation VHDL-AMS d'un système hétérogène de mise à feu basé notamment sur les micros interrupteurs pyrotechniques développés par P. Pennarun [Pen06] et C. Rossi [Ros97]. Ce système comprend :

- Une partie électro-mécanique composée notamment d'un accéléromètre qui permet la détection du mouvement du missile et donne ensuite un paramètre de la partie commande,
- Une partie commande qui permet, suivant un comportement de machine à états, de récupérer les requêtes de l'utilisateur (missile armé, missile stérilisé, explosion du missile,…) pour délivrer les commandes adéquates en prenant en compte certaines conditions,
- Une partie alimentation qui permet de délivrer les pulses de puissance nécessaires à l'activation des interrupteurs pyrotechniques lorsque la partie commande le demande,
- Une partie pyrotechnique qui regroupe les différents interrupteurs pyrotechniques ainsi que la partie amorce de la charge explosive.

3.1.2 Spécifications du système de mise à feu de la charge pyrotechnique

La mise à feu d'une charge pyrotechnique dans ses applications civiles ou militaires doit être sécurisée. Les fonctions de sécurisation sont fixées par des normes dépendantes de l'application (voir norme STANAG 4187 [STA]) mais qui, quelque soit le cas, fait appel aux mêmes fonctions de base :

- La fonction principale consiste à alimenter un fil chaud qui va porter le matériau pyrotechnique de l'amorce à sa température d'auto combustion,
- Durant le stockage, l'amorce doit être sécurisée par deux approches couplées et complémentaires :
  - Un obstacle mécanique est interposé entre « l'amorce » et le matériau secondaire.
  - Un court-circuit à la masse évite tout transfert d'énergie sur le fil chaud (Norme : exciter le système avec 1V, 1A sans possibilité de mise à feu) (cf. [STA]).
- Une fonction de stérilisation est souvent prévue de telle sorte que lorsque la décision de stérilisation est prise, il ne soit plus possible de remettre en route le système de mise à feu.

L'originalité de la mise à feu proposée par le LAAS-CNRS est l'utilisation exclusive de composants pyrotechniques pour réaliser toutes ces opérations de sécurisation. Le schéma élémentaire représentant notre système est celui de la Figure 42 :

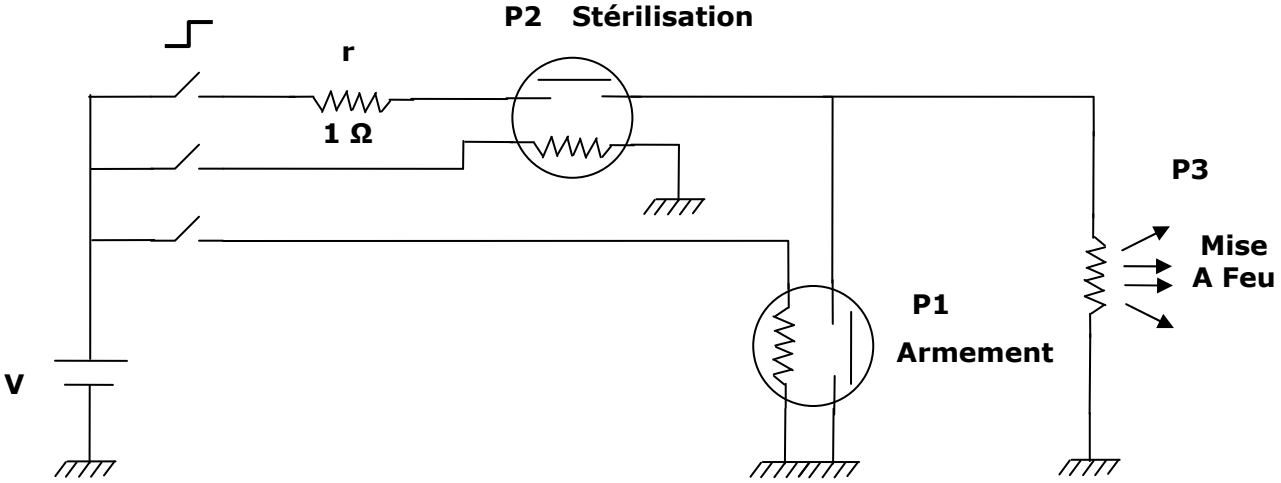


Figure 44 : Schéma élémentaire du système sécurisé de mise à feu

Les différents composants pyrotechniques sont :

- P3 est un fil chaud dans lequel vient circuler un courant pour initier le matériau pyrotechnique
- P1 est le système de sécurité assurant les exigences de la norme : en période de stockage, il court-circuite l'alimentation des convertisseurs de sortie qui annihile toute tentative de mise à feu dans la limite 1 Volt, 1 Ampère. Cela se traduit simplement par une perte de calorie, par effet joule, dans la résistance de dissipation r = 1 Ω
- P2 assure la fonction de stérilisation : le composant pyrotechnique va détruire la connexion entre la source d'alimentation et P3. P1 et P2 sont des interrupteurs pyrotechniques ON-OFF

Un composant original a été mis au point pour réaliser les fonctions P1 et P2 par les travaux de thèse de P. Pennarun (cf. Figure 43) [Pen06]. Il s'agit d'un micro-initiateur sécurisé baptisé PYROSECURE, constitué d'un élément chauffant de type résistif venant initier un matériau pyrotechnique au contact par effet joule. Une piste métallique traverse la membrane. Lorsque le matériau pyrotechnique brûle, la membrane se casse et la piste métallique se rompt. L'interrupteur passe donc de l'état ON à OFF.

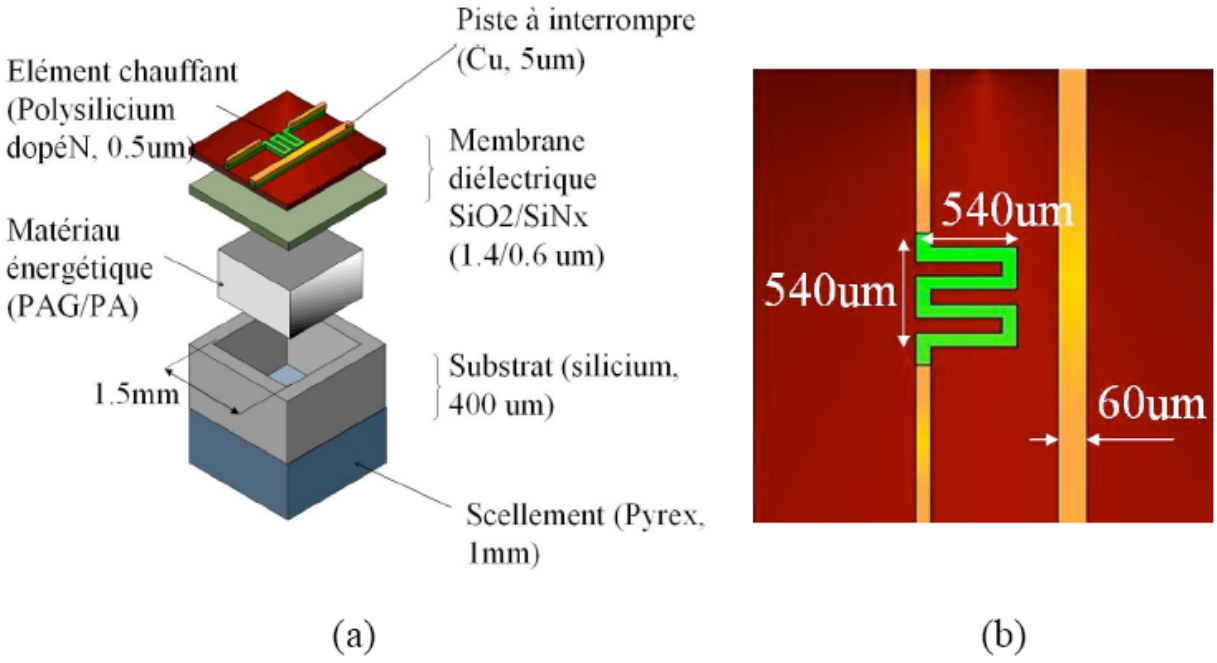


Figure 45 : Dispositif Original PYROSECURE pour la fonctionnalité Interrupteur ON-OFF pyrotechnique : (a) vue éclatée indiquant les couches et leurs épaisseurs (b) vue de dessus avec les cotes (Source [Pen06])

Ce dispositif sera modélisé en VHDL-AMS dans le paragraphe 3.1.3.4, afin de prévoir les transitions de mise à feu compte tenu des variantes technologiques possibles de ce dispositif.

Puissance (mW)	**65**	**150**	**200**	**300**	**500**
$R_{initiateur}$ (Ω)	950	980	925	970	890
Courant appliqué (mA)	8.3	12.4	14.7	17.6	23.7
Temps d'initiation (ms)	**inf**	**97**	**65**	**39**	**24**
Temps d'initiation théorique (ms)	120	72	44	32	14
Energie (mJ)	**inf**	**14.5**	**13**	**11.7**	**12**
Initiations réussies(%)	0	20	40	60	60

Figure 46 : Récapitulatif des tests d'initiation pour le micro initiateur sur membrane SiO2/SiNx rempli avec du PAG/PA (Source[Pen06])

Pour cela, au niveau de l'initiateur, les données expérimentales des tests d'initiation sur les premiers prototypes réalisés par P. Pennarun (cf. Figure 44), servent de référence pour notre modèle d'initiateur qui sera utilisé dans notre système complet. Les points important sont les temps d'initiation obtenus en fonction de la puissance émise.

3.1.3 Modélisation VHDL-AMS du système

La modélisation VHDL-AMS du système reprend l'approche structurelle de la description du système au paragraphe 3.1.1 qui décomposait le système en quatre parties principales : partie Accéléromètre, partie Commande avec interface utilisateur, partie alimentation et la partie pyrotechnique. Chaque sous partie est validée indépendamment dans un banc d'essai adapté pour ensuite être intégrée au système

complet. Cela permet une validation des modèles plus efficace. Dans les paragraphes suivant nous allons développer la création des modèles VHDL-AMS de chaque sous-partie en rappelant les spécifications à vérifier.

3.1.3.1 Modélisation de la partie accéléromètre

La partie accéléromètre est une partie électromécanique. Ce sous système doit, d'après les spécifications, être capable de mesurer l'accélération du missile et fournir l'information à la partie commande (cf. 3.1.3.2) du mouvement ou non du missile. Si le missile n'est pas en mouvement, l'initiation des charges pyrotechniques est inhibée d'après les spécifications.

Pour modéliser l'accéléromètre, nous avons repris les travaux effectués dans la revue [PLV05], qui propose un modèle d'accéléromètre intégré dans un système d'airbag. Pour cette application, l'accéléromètre devait être capable de mesurer une forte décélération (4G). Nous allons utiliser ce modèle, dans le cadre cette fois-ci de la mesure d'une forte accélération lors du démarrage d'un missile (seuil non encore fixé : attente des spécifications accélération missile).

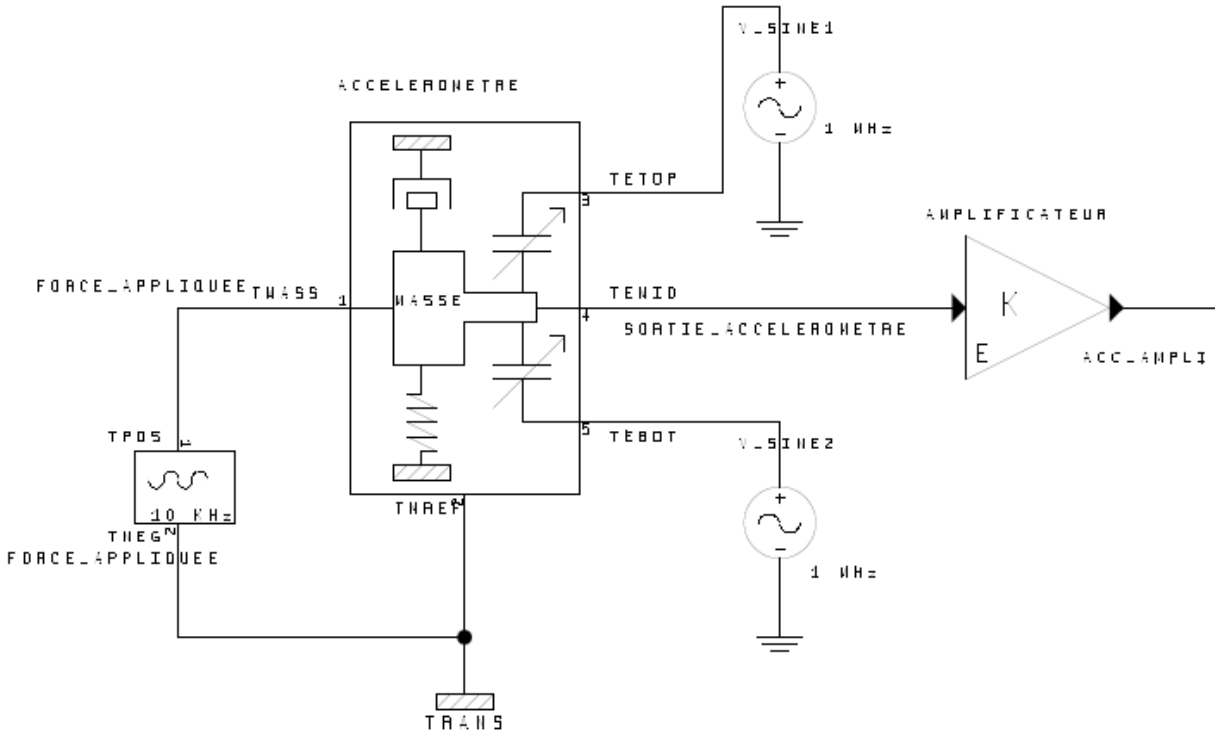


Figure 47 : Sous Système Accéléromètre

La structure de l'accéléromètre est représentée sur la Figure 45. L'accélération est modélisée sous la forme d'une force appliquée de type sinusoïdale à une fréquence de 10 kHz. L'accéléromètre fait l'interface entre le stimulus d'accélération et un comparateur électrique analogique. Le modèle VHDL-AMS de l'accéléromètre est décomposé en deux parties :

- Une première partie qui transforme l'accélération du missile en un déplacement mécanique par le biais d'une masse donné par l'équation différentielle du second degré (1) avec : *F* la force appliquée à la masse *M*, *D* le coefficient d'amortissement et *k* le coefficient de raideur.

$$F(t) = M\frac{d^2x}{dt^2} + D\frac{dx}{dt} + kx \qquad (1)$$

Une deuxième partie convertit le déplacement mécanique en un signal électrique. La masse est en effet associée à deux capacités, et son déplacement entraine une modification de leur valeur respective.

La modélisation met en jeu les lois de Kirchhoff sur la conservation de l'énergie, qui sont implicites dans le langage VHDL-AMS. L'écriture des équations physiques se fait en VHDL-AMS de manière directe par le biais d'instructions simultanées (cf. meta modèle VHDL-AMS 2.4.2).

3.1.3.2 Modélisation de la partie Commande

La partie commande gère les ordres des utilisateurs du missile, récupère les informations des capteurs de mouvement et envoie les commandes d'activation des micro-initiateurs pyrotechniques suivant un comportement bien défini. Ce comportement est régi par une machine à état représentée sur la Figure 46. Quatre états principaux ont été définis :

- P0 : Phase de veille sécurisée,
- P1 : Phase Initiateur Stérilisé,
- P2 : Phase Attente Déclenchement,
- P3 : Initiateur Actionné.

La transition entre chaque état se fait suivant des conditions bien précises conformément aux spécifications du système. Ainsi la transition de la Phase 0 à la Phase P2 s'effectuera lorsque deux conditions seront satisfaites à savoir qu'il y ait une requête d'armement du missile et qu'il y ait mouvement du missile.

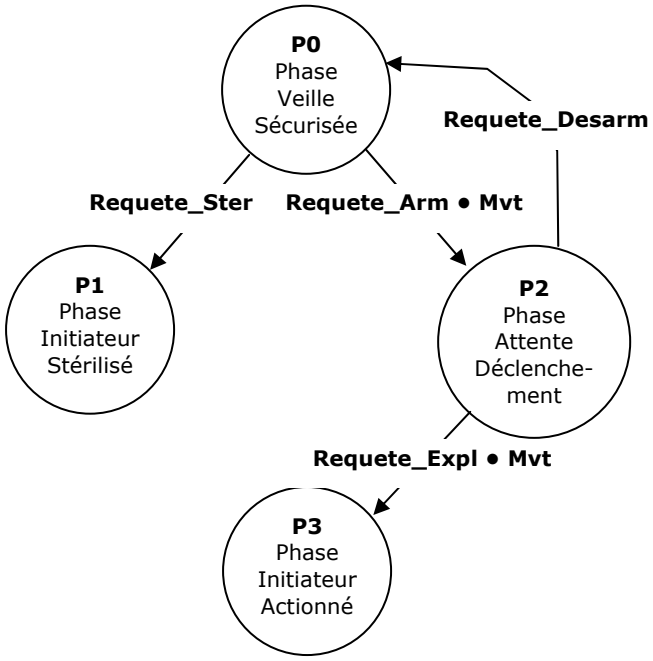


Figure 48 : Machine à Etats Finis de la partie Commande

Ce type de modélisation s'effectue très efficacement en VHDL-AMS par le biais d'instructions concurrentes comme l'instruction *process* et d'instructions séquentielles conditionnelles comme l'instruction *case* (cf. meta modèle VHDL-AMS 2.4.2) (cf. Source du modèle partie commande Annexe 3.2).

Cette machine à état s'intègre dans la partie commande contenant la partie comparateur, qui traite le signal électrique envoyé par la partie accéléromètre pour en obtenir un signal booléen permettant de savoir si le missile est en mouvement ou non. Cela se modélise en VHDL-AMS notamment par une détection de seuil sous la forme d'un attribut *above* qui transforme une quantité en un signal booléen (cf. Source du modèle partie commande Annexe 3.2). Le sous système commande, représenté sur la Figure 47, compte outre la partie commande, une partie de stimuli modélisant les requêtes de l'utilisateur du missile.

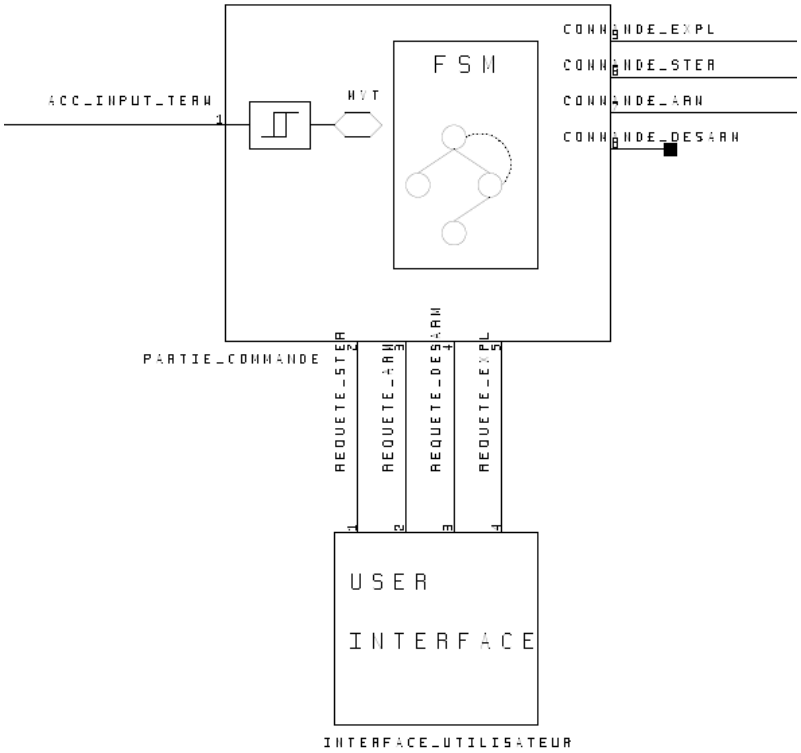


Figure 49 : Sous Système Commande

3.1.3.3 Modélisation de la partie alimentation

La partie alimentation du système est encore à l'étude et les spécifications la concernant ne sont pas encore bien définies. Néanmoins, nous savons que la batterie utilisée sera comprise entre 5 et 10v, et devra pouvoir générer des pulses de puissances suffisants à l'initiation des composants pyrotechniques (initiateurs et interrupteurs). D'après les spécifications données sur la Figure 44, il faut avoir une puissance de 500 mW pour déclencher les interrupteurs.

Aucune solution physique n'a encore été définie, nous avons donc opté pour une modélisation complètement comportementale de cette partie à un très haut niveau d'abstraction. Ce modèle est purement descriptif et a pour objet de valider le système complet. Pour se faire un modèle de pulse de puissance a été écrit en VHDL-AMS permettant d'envoyer un pulse d'une puissance défini par le concepteur lorsque la commande adéquate arrive. La structure de ce sous système alimentation est donné sur la Figure 48.

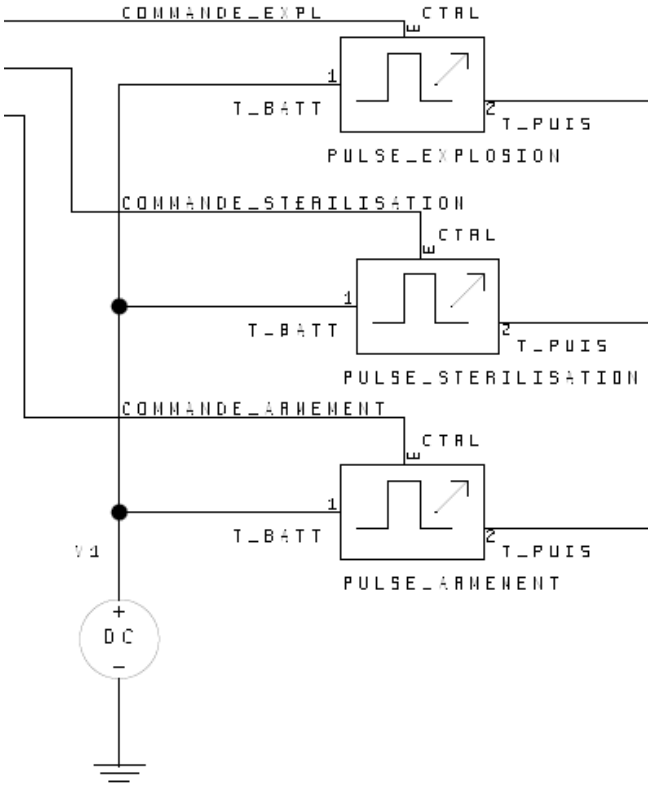


Figure 50 : Modélisation du sous système alimentation

Des futures améliorations de cette partie seront notamment :

- D'utiliser un modèle de batterie plus réaliste que notre source idéale de tension. L'outil SystemVision intègre par exemple la bibliothèque de modèles VDA (Verband Der Automobilindustrie) [VDA07] qui contient de nombreux modèles VHDL-AMS pour une application notamment dans le domaine automobile. On y trouve un modèle simple de batterie.

- D'améliorer le modèle de pulse en mettant en place un modèle physique suivant l'architecture choisie.

3.1.3.4 Modélisation de la partie pyrotechnique

Les spécifications des parties pyrotechniques ont été principalement définies dans le paragraphe 3.1.2. Pour des raisons de temps, l'interrupteur permettant le désarmement n'est pour le moment pas modélisé. Seuls les deux interrupteurs permettant la stérilisation et l'armement (P1 et P2 cf. Figure 43) ont été modélisés. Cette partie est représentée sur la schématique de la Figure 49. Ces composants ont un comportement identique (interrupteur one shot ON-OFF). L'initiateur de l'amorce reprend aussi la même base avec l'allumage par effet joule du matériau pyrotechnique, mais ici dans le but d'une amorce pour une explosion et non une rupture de membrane.

L'initiateur a été modélisé suivant la méthode A-FSM développée par Y.Hervé (cf. [Her03]). Cette méthode adapte le principe de la machine d'états finis pour décrire le comportement de système analogique. La décomposition du comportement d'un système en différents états auxquels on lie un ensemble d'équations physiques, permet de modéliser simplement des systèmes complexes. La méthode s'appuie sur la combinaison possible en VHDL-AMS de l'instruction conditionnelle *case is* au sein d'un *process* pour le management des états (transitions), et de l'instruction simultanée *case use* qui regroupe

par état l'ensemble des équations physiques régissant le comportement du système dans cet état (cf. Source du modèle Interrupteur ON-OFF Annexe 3.3).

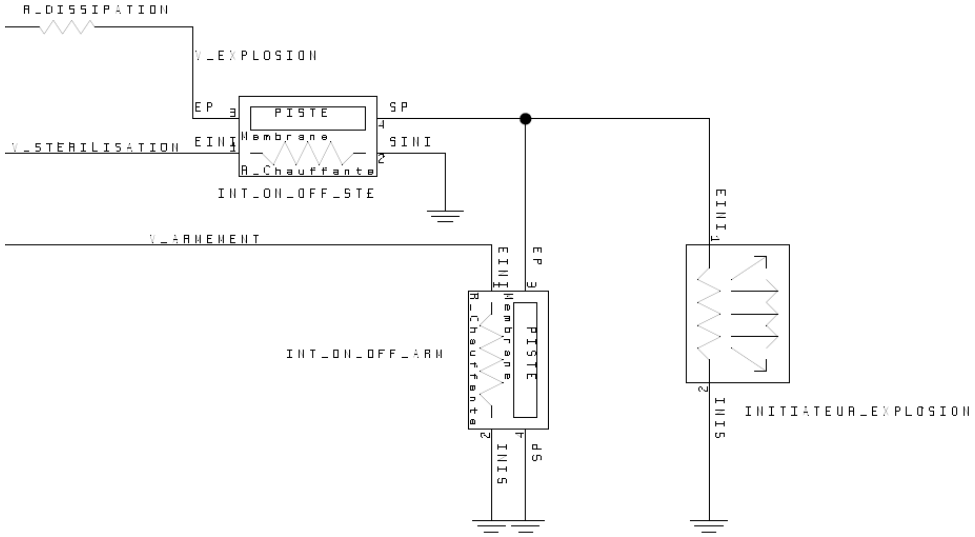


Figure 51 : Modèle de la partie sous système pyrotechnique

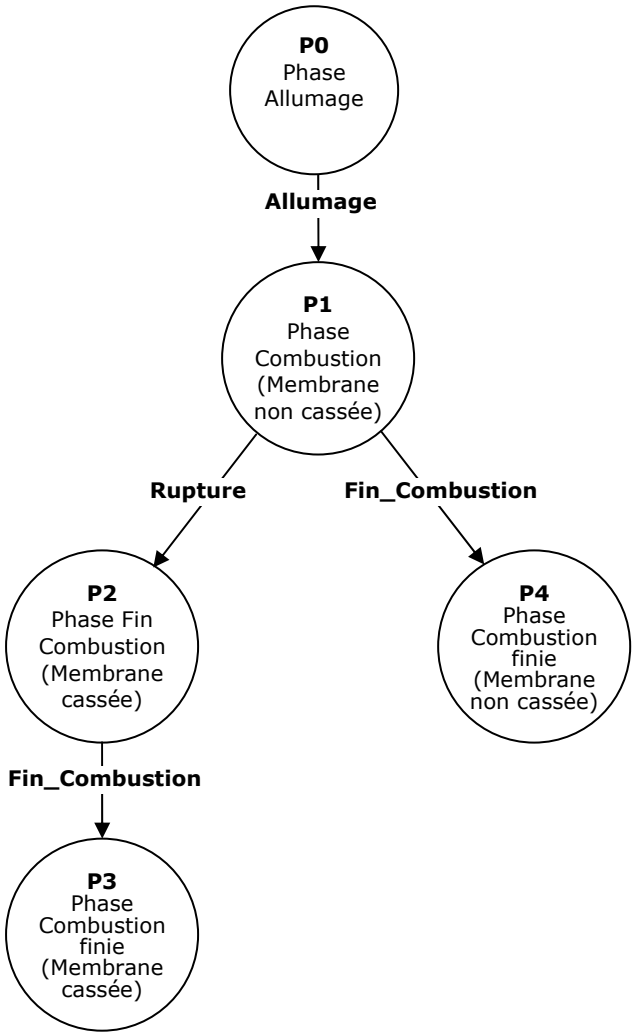


Figure 52 : Machine à Etats Finis Analogique de l'interrupteur ON-OFF pyrotechnique

L'A-FSM de l'initiateur est donné sur la Figure 50. Les équations physiques que nous avons définies afin de modéliser le comportement des initiateurs sont relatives aux solides et aux gaz générés notamment au cours de la combustion. Les principales équations physiques que l'on retrouve sont entre autres :

- Les équations régissant la température à la surface du matériau pyrotechnique,
- La pression sous la membrane,
- La masse consumée de matériau pyrotechnique.

3.1.3.5 Intégration et Résultats de Simulation

L'ensemble des modélisations a été réalisée à l'aide de l'outil SystemVision de la société Mentor Graphics [Sys50]. Le logiciel permet la saisie de modèles comportementaux avec un éditeur de modèle VHDL-AMS, mais aussi la saisie de schématique pour réaliser une architecture structurelle d'un système.

Pour effectuer une validation du modèle complet du système, les différents modèles VHDL-AMS réalisés ont d'abord été validés de manière indépendante et ensuite intégrés au sein d'une schématique reprenant la structure complète du système (cf. Figure 52).

La partie accéléromètre a été tout d'abord validée par le biais d'un banc d'essai visant à vérifier la bonne détection d'une accélération via l'application d'une force appliquée à une masse avec en résultante un signal électronique dont l'amplitude est proportionnelle à l'accélération subie. La Figure 51 présente les résultats de simulation, à savoir la courbe de déplacement de la masse (Tmass) et la courbe du signal électrique amplifié en sortie d'accéléromètre en fonction du temps. Vient ensuite l'analyse de ce signal électronique qui est gérée par un processus de la partie commande, avec une détection de seuil. Un nombre fini de pulse (ici 10) dans une demi période donnée, est recherché afin de s'assurer qu'il y ait bien mise en mouvement. Les résultats sont donnés sur la Figure 51.

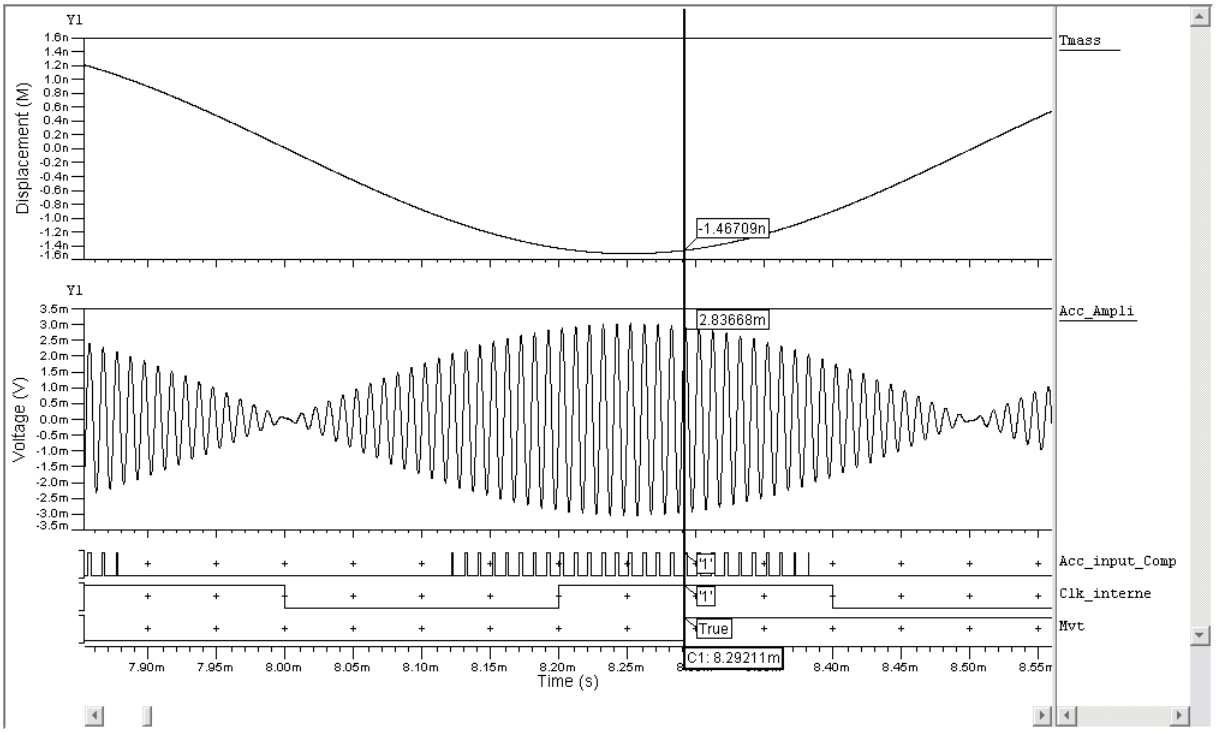
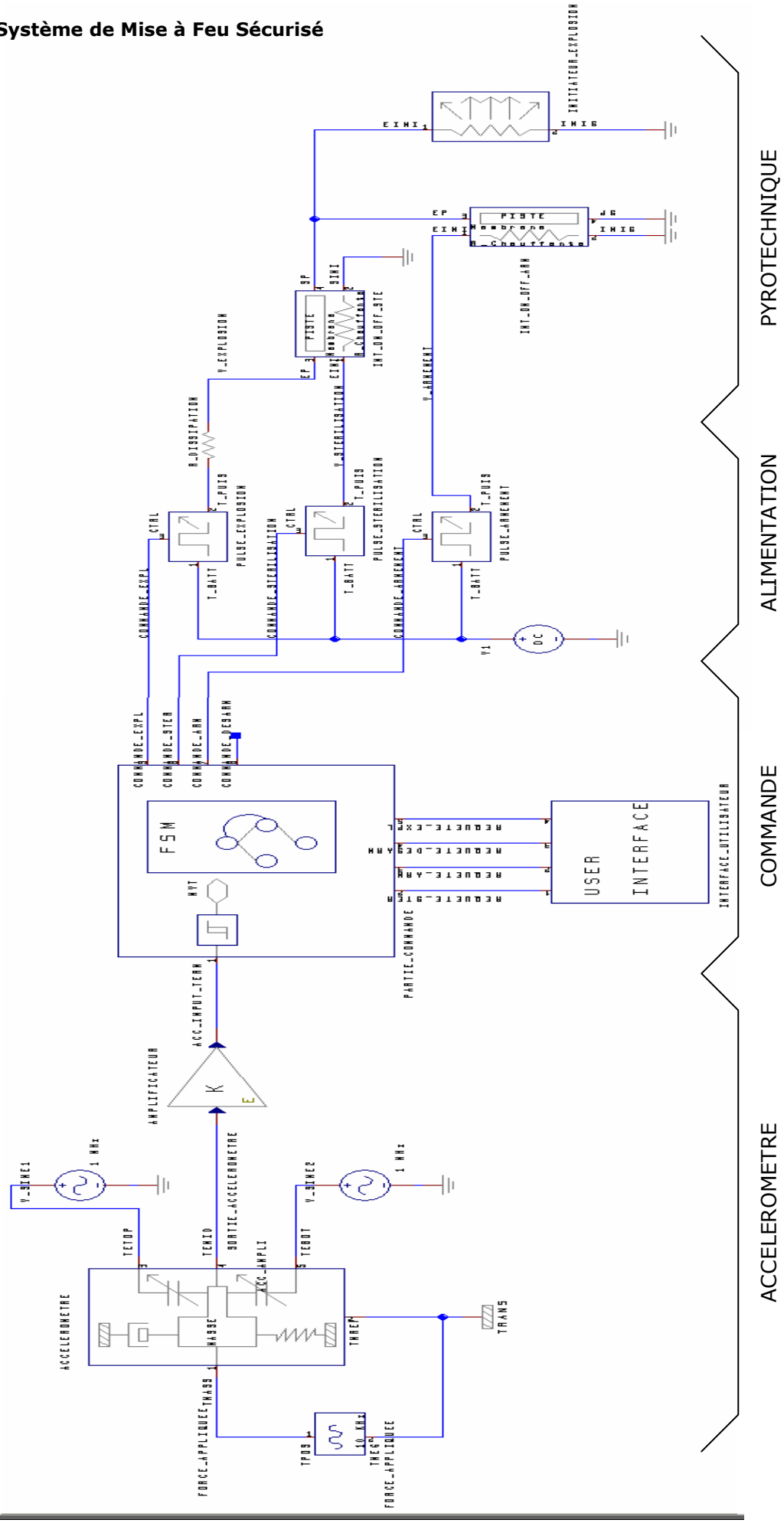


Figure 53 : Résultats du sous système Accéléromètre avec reconnaissance de mouvement (Waveform Viewer SystemVision)

Figure 54 : Système de Mise à Feu Sécurisé



La partie pyrotechnique a ensuite été validée. Nous avons commencé par la validation d'un seul initiateur permettant la réalisation de la fonctionnalité d'un interrupteur ON-OFF conforme aux spécifications données dans le paragraphe 3.1.2. Le banc d'essai de cet initiateur a consisté en un générateur de pulse de puissance commandé par un signal de commande. La principale caractéristique observée est le temps d'initiation qui sépare la commande logique et la rupture de la membrane. Pour cela on observe sur la Figure 53, les résultats de simulation de l'initiateur avec le signal de commande, la puissance émise, la température surfacique du matériau pyrotechnique, l'état de la AFSM (machine à états finis analogique)).

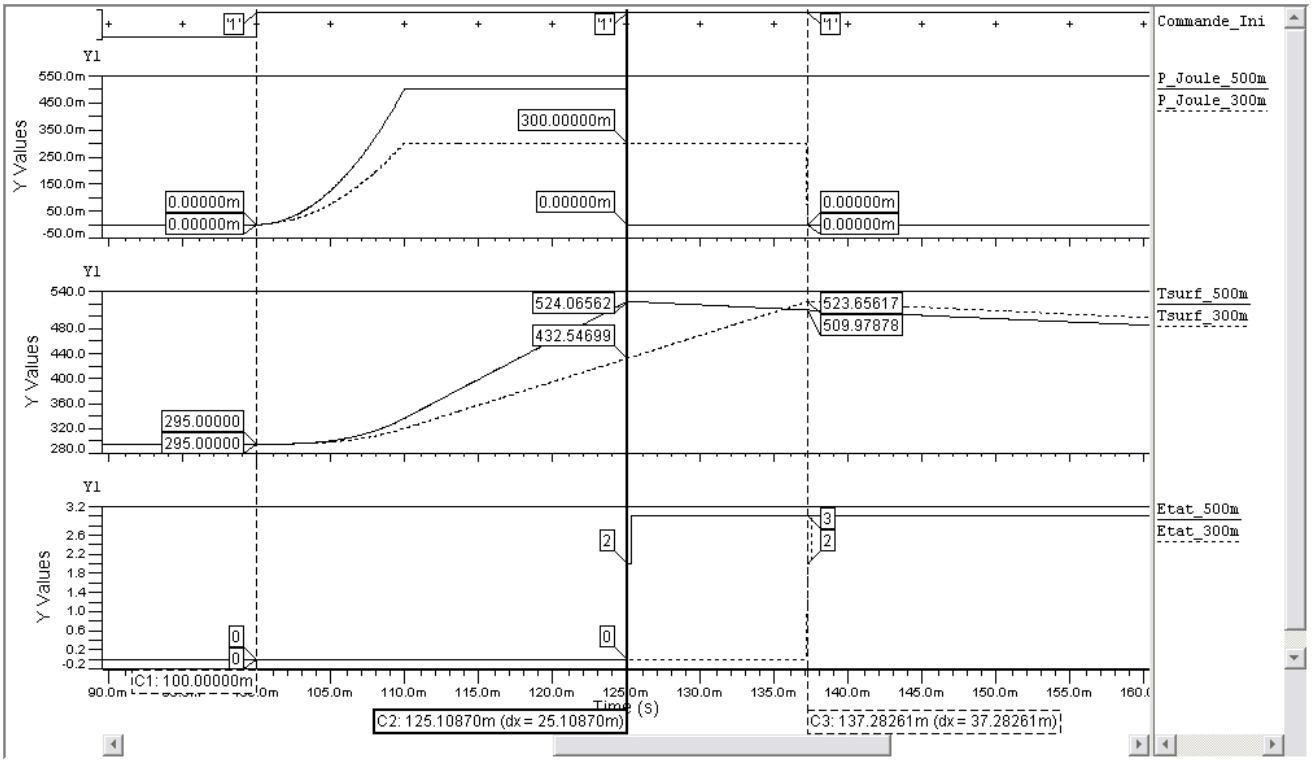


Figure 55 : Résultats de Simulation d'un initiateur pyrotechnique réalisant la fonctionnalité interrupteur ON-OFF (Waveform Viewer SystemVision)

L'analyse de ces courbes montre que le modèle de l'initiateur est en conformité avec le comportement attendu de l'initiateur. Il passe en état de combustion (Etat 2) lorsque la température surfacique attente un seuil (523°K). La phase de combustion entrainant la rupture de la membrane est très rapide nous le verrons sur la Figure 54. Le temps d'initiation donné par notre modèle est comparé avec les résultats obtenus par P. Pennarun (cf. [Pen06]) qui a modélisé en éléments finis, réalisé et caractérisé des prototypes réels de ces interrupteurs. Les résultats sont donnés dans le Tableau 6.

Puissance (en mW)	300	500
Prototype réel	**39**	**24**
Théorie (Éléments Finis)	**32**	**14**
Modèle (avec amorce P 10ms)	**37**	**25**

Tableau 5 : Temps d'Initiation (en ms)

Nous pouvons constater un ordre de grandeur identique des temps d'initiation. Cependant des différences `de l'ordre de 5% entre les prototypes réels et notre modèle) existent qui peuvent être reliées aux erreurs dues à l'utilisation d'un profil d'application de puissance légèrement différent. Dans le cas de notre modèle, le profil impose une

rampe de 10 ms qui influe sur le temps d'initiation. Une amélioration du modèle est aussi à envisager avec l'intégration d'effets physiques négligés au préalable, et une amélioration des équations existantes. Nous pensons notamment à l'équation de la température surfacique du matériau pyrotechnique, qui a due être simplifiée à cause de la non possibilité de modéliser les équations à dérivées partielles en VHDL-AMS. Cette contrainte aurait aussi pue être contournée avec l'intégration de données provenant d'un modèle à éléments finis, mais nous aurions perdu le caractère prédictif du modèle.

La phase de combustion entrainant la rupture de la membrane étant d'une durée très courte comparée à la phase d'allumage, nous avons représenté sur la Figure 54 un ZOOM sur ces changements d'états. Une courbe donnant la masse consumée de matériau pyrotechnique a été ajoutée. La combustion de l'ensemble du matériau pyrotechnique entraine un changement d'état.

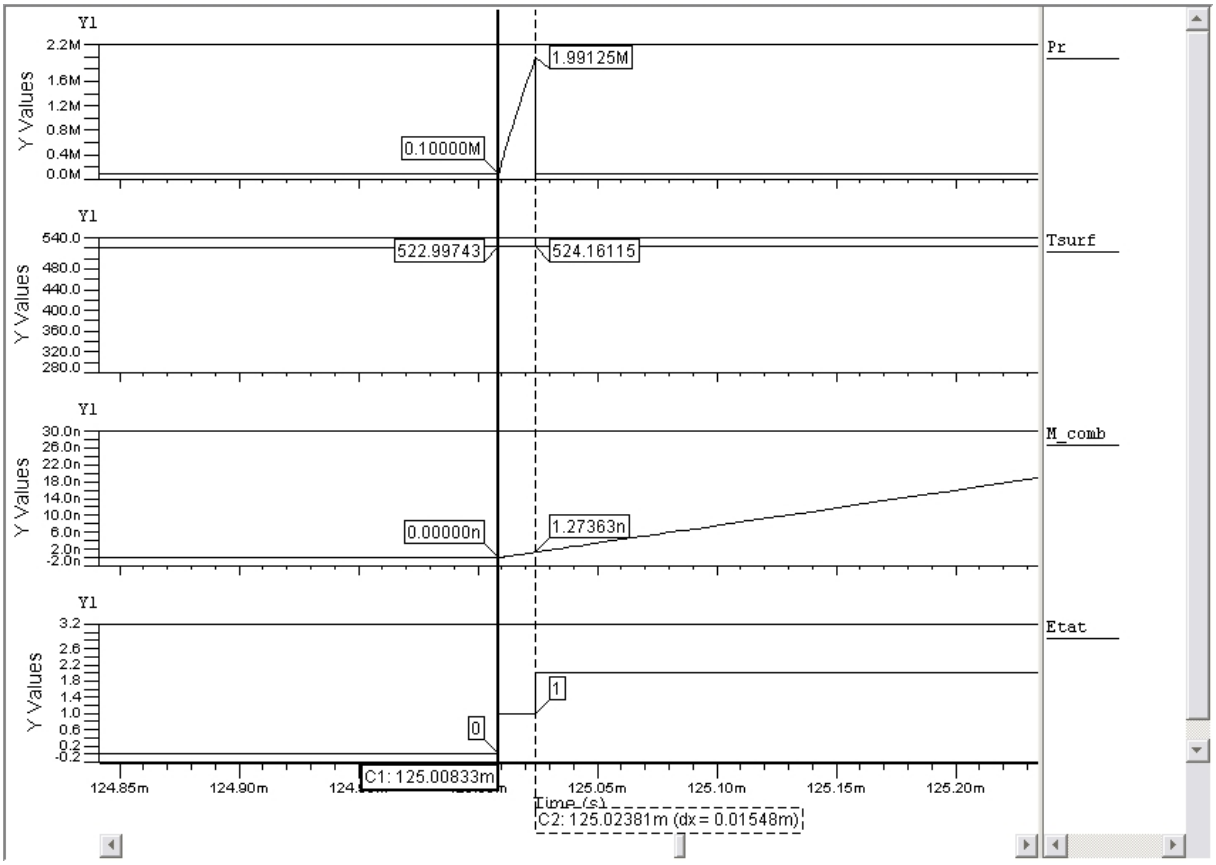


Figure 56 : Zoom des résultats précédents sur la phase de conbustion (Waveform Viewer SystemVision)

Le démarrage de la combustion à la fin de la phase d'allumage (Etat 0 vers Etat 1), entraine une augmentation de la pression sous la membrane. L'initiateur passe dans l'Etat 2, lorsque la pression dépasse le seuil de la pression de rupture de la membrane. Ces courbes nous permettent d'évaluer un temps de combustion du matériau pyrotechnique, permettant la rupture de la membrane, de 15 µs. L'Etat 2 se poursuit jusqu'à ce que tout le matériau pyrotechnique soit consumé (vers Etat 3 cf. Figure 53).

Une fois l'ensemble de ces sous systèmes validés, nous pouvons les intégrer dans le système complet (cf. Figure 52). Cela nous permettra aussi de valider la partie commande, et le comportement global du système. Nous avons choisi de représenter dans ce mémoire, l'activation classique du missile à savoir, l'armement du missile (si mouvement) et l'initiation du missile (si mouvement). Les résultats sont représentés sur la Figure 55.

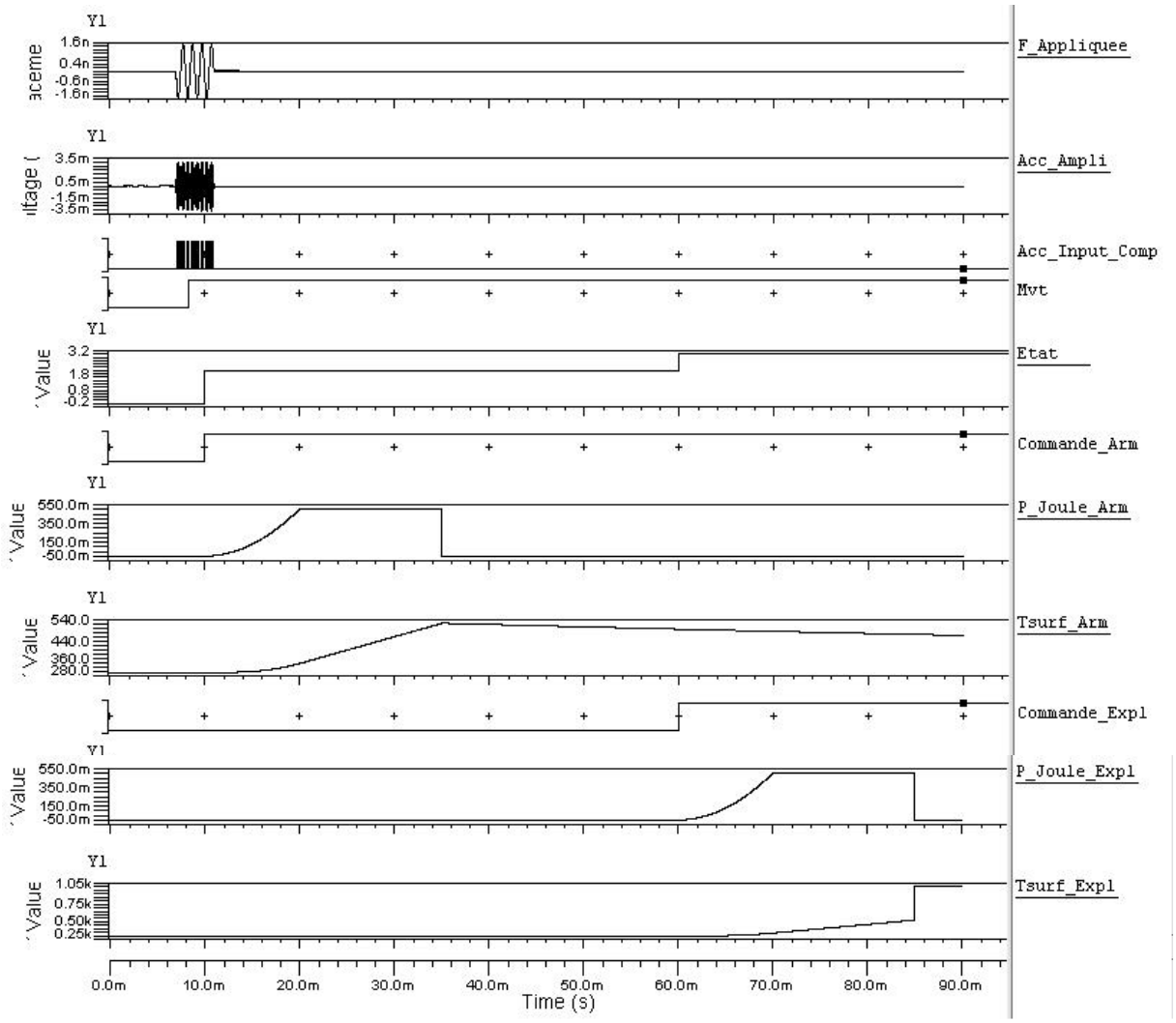


Figure 57 : Résultats Simulation Système complet (activation classique d'un missile) (Waveform Viewer SystemVision)

Les résultats montrent que le comportement de notre modèle est conforme aux spécifications définies au préalable. Ainsi le missile explose (ici à T=85ms), après qu'il ait été au préalable mis en mouvement (détection de l'accélération à T=8ms). L'armement est ensuite réalisé : initiation de l'interrupteur ON-OFF de sécurité : commande à T=10ms, rupture membrane à T=35ms. L'initiation de l'amorce est ensuite commandée à T=60ms. Le signal *Etat* nous permet de savoir où l'on se place dans la machine à états finis de la partie commande (cf. Figure 46).

D'autres scénarios ont été simulés, qui nous ont permis de valider l'ensemble de la machine à états de la partie commande (stérilisation, commande d'explosion avant armement,…), mais les résultats ne seront pas représentés ici.

3.1.4 Conclusions sur le VHDL-AMS comme support de la Solution Physique

La modélisation en langage de description de matériel normalisé VHDL-AMS d'un système complet de mise à feu sécurisé, nous a permis de mettre en avant les facultés de ce langage à modéliser des systèmes hétérogènes et ce à plusieurs niveaux différents d'abstraction. Nous avons ainsi pu modéliser au sein d'un même modèle des composantes mécaniques, électroniques analogiques, électroniques numériques, thermique,…

Au cours du paragraphe 3.1.3, nous avons explicité les modélisations des sous systèmes et les moyens qu'offre le VHDL-AMS pour y parvenir. La possibilité d'avoir dans des structures des modèles ayant des niveaux d'abstraction différents, permet de développer une partie du système avec un modèle plus fin (pouvant être prédictif), tout en ayant son environnement qui bien que moins développé (modèles souvent descriptifs) permet le bon fonctionnement du modèle.

Cette approche permet aussi de gagner en temps de calcul qui est un des paramètres les plus sensibles. On se retrouve bien souvent dans le besoin d'augmenter le niveau d'abstraction d'une partie des modèles, cette limite provient essentiellement des performances de l'ordinateur sur lequel tourne le programme. Mais il s'avère que d'un outil de simulation VHDL-AMS à un autre, des différences surviennent aussi sur un même ordinateur (cf. [Gui03]). Cependant, des solutions logicielles peuvent être mises en place comme le partage de tâches au sein d'un même logiciel, comme nous l'avons souligné dans le paragraphe 1.3.2.1 avec le logiciel SystemVision.

On peut aussi rencontrer certains problèmes lors de l'intégration au sein d'un même système, de modèles ayant des domaines d'application très différents notamment au niveau de la fréquence d'activité. Ainsi un modèle comme celui de l'accéléromètre ou un modèle numérique (processeur,…) va créer des points de simulation à une fréquence très élevée. Cela aura une influence certaine sur le temps de simulation s'ils sont utilisés au sein d'une simulation d'un système complet couvrant plusieurs secondes simulés.

Cela reste néanmoins des problèmes qui peuvent être atténués par l'évolution rapide des capacités de calcul des ordinateurs et ne remettent pas en cause les facultés du VHDL-AMS à modéliser des systèmes hétérogènes.

3.2 De la solution logique à la solution physique

Notre processus de conception système tel que nous l'avons défini dans notre groupe de travail au LAAS-CNRS, nous amène à établir une solution logique, pour ensuite en élaborer une solution physique. Le langage VHDL-AMS que nous souhaitons utiliser comme support de la solution physique va hériter de ce travail de modélisation amont. De nombreux résultats obtenus par des travaux récents ([Ham05], [Mau05], [Gui03], [Alb05]) ont montré qu'une passerelle entre la plateforme HiLeS, qui propose une solution logique dans un formalisme à base de réseaux de Petri, et le langage VHDL-AMS est réalisable.

L'objectif du langage HiLeS est d'aider à la conception amont des systèmes (cf. [Ham02], [HSc03], [Jim00] et [Ham05]). Ainsi, en partant des spécifications générales, nous obtenons une représentation par blocs fonctionnels interconnectés et hiérarchisés dont on peut évaluer la bonne conformité fonctionnelle et opératoire.

Précédemment au cours du paragraphe 2.3.4, nous avons explicité les différents objets et concepts présents dans le formalisme HiLeS. Nous avons alors déduit un meta modèle HiLeS (cf. Figure 36), qui va assurer la conformité des modèles HiLeS que l'on va créer.

Dans cette partie, nous allons nous intéresser à la manière dont on peut modéliser en VHDL-AMS les différents objets représentant les éléments du langage HiLeS, à savoir les modèles fonctionnels et structurels, le modèle de commande et les canaux. Cela nous permettra notamment de définir un certain nombre de règles de transformation. Nous verrons également que des travaux de transformations ont déjà été effectués, et seront explicités par la suite.

3.2.1 Traduction des Blocs HiLeS en VHDL-AMS

Dans une première étape, nous allons aborder la traduction des blocs structuraux et fonctionnels du formalisme HiLeS en VHDL-AMS. Ces deux types de blocs permettent de définir la structure d'un projet et doivent être cohérents avec une structure standard VHDL-AMS construite à base d'entités et d'architectures. Ces travaux de traduction ont été réalisés conjointement avec J.C. Hamon [Gui03][Ham05].

3.2.1.1 Schématique niveau 0 et concept de banc d'essai

La représentation de plus haut niveau du projet que veut créer l'utilisateur peut être modélisée comme une entité *banc d'essai* ne possédant aucune entrée/sortie, ou comme un module pouvant être réutilisé par la suite dans un autre projet (niveau 0). On utilise le concept de banc d'essai ici (comme une représentation de niveau 0) pour modéliser un système et son environnement afin d'en évaluer les performances. La Figure 56a illustre ces propos avec le système modélisé au sein de son environnement. Le réseau de Petri cadence l'exécution des actions dans l'environnement du système. Toute la partie des réseaux de Petri peut être regroupée en un seul bloc de commande, comme évoqué dans le paragraphe 3.2.2. Son architecture va être décrite de manière structurelle en instanciant et en reliant les différents blocs entre eux via les différents canaux. La création des signaux, quantités, terminaux locaux en VHDL-AMS se fait automatiquement lorsque l'utilisateur relie graphiquement les différents blocs entre eux par le biais de l'interface graphique de l'outil HiLeS Designer. La Figure 56b présente l'architecture du bloc « System » au Niveau -1.

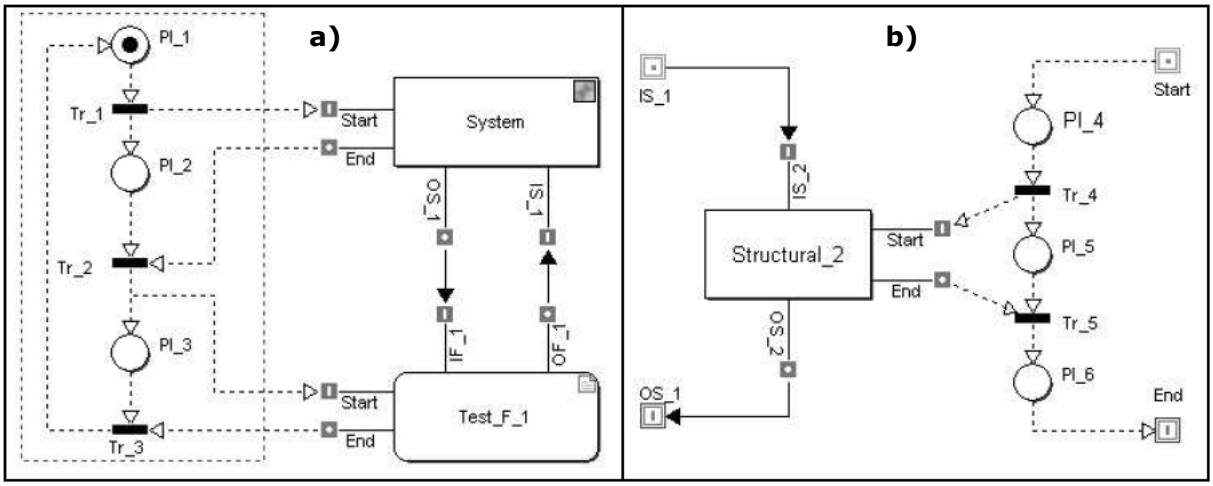


Figure 58 : a) Représentation du système au Niveau 0 HileS (Banc d'Essai)
b) Strucuture du Bloc « System » au Niveau -1 (Source [Ham05])

3.2.1.2 Interprétation des Blocs Structurels

Les blocs structurels sont modélisés par des entités. L'architecture de ces entités (qui se situe à un niveau –1 dans une nouvelle schématique) est décrite de manière structurelle en instanciant et en reliant les différents blocs contenus dans le bloc structurel entre eux via les différents types de canaux. L'entité possède des entrées/sorties qui interviennent dans l'architecture. Ainsi, au niveau 0, on définie pour l'entité :

- son nom (lorsque l'on crée le bloc),
- son *port map* (suivant que l'on relie l'entité à d'autres blocs)
- son *generic map* (en effet par souci de ré-utilisabilité, l'utilisateur doit avoir la possibilité de paramétrer une entité (exemple : modèle d'une résistance dont on peut modifier la valeur)).

Concernant les ports, l'utilisateur doit pouvoir en spécifier le nom, l'objet (signal, terminal, quantité), le type (std_ulogic, bit, etc…) ou la nature (electrical, thermal, etc…) (cf. Figure 57). L'utilisateur a la possibilité de choisir une ou plusieurs bibliothèques à utiliser lui donnant accès à des natures et des types. Il peut aussi appeler une bibliothèque non prédéfinie. Cependant le type et la nature du port vont être étroitement liés avec les caractéristiques des canaux auxquels sont reliés les ports (cf. 3.2.1.4). Ainsi suivant le sens des canaux, les sens des ports seront forcés en mode in ou en mode out, sauf dans le cas des terminaux qui sont par définition toujours en mode in/out. En VHDL-AMS tous les objets sont typés (strong typing) permettant une initialisation fiable et augmentant le pouvoir de vérification du compilateur. On ne peut donc pas connecter tout et n'importe quoi ensemble. Ainsi, en spécifiant le champ type d'un port, on force le type des canaux reliés à ce port et inversement.

L'utilisateur peut aller à l'intérieur du bloc (via la commande *Go Inside*) (niveau –1) afin d'y créer l'architecture de l'entité en connectant différents types de blocs. Les ports d'entrées\sorties appelés services apparaissent au niveau –1 de l'interface graphique et sont donc accessibles pour être connectés aux autres blocs (cf. Figure 56 b)).

Dans le cadre de l'utilisation d'un bloc déjà créé et disponible dans une bibliothèque (ex. IP), l'utilisateur peut choisir l'entité qu'il souhaite utiliser, son architecture s'il en existe plusieurs et préciser ses paramètres génériques. Cela équivaut en VHDL-AMS à instancier une entité d'ores et déjà compilée et stockée dans une bibliothèque qu'il faudra spécifier. On peut aussi créer une unité de conception *configuration* qui permet

d'associer une entité et une architecture (cela n'est pas encore implémenté dans HiLeS Designer pour le moment). Les caractéristiques *generic map* et l'unité de conception *configuration* sont les bases de la réutilisabilité en VHDL-AMS.

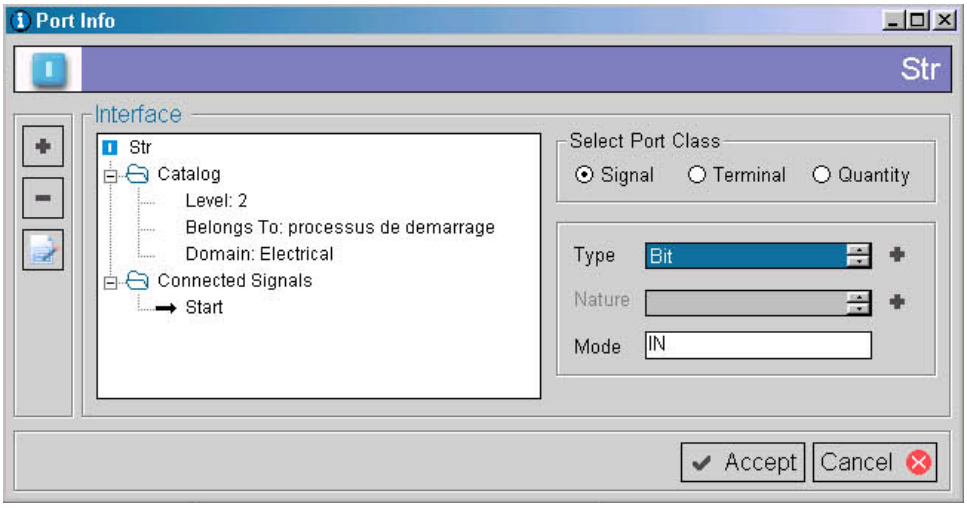


Figure 59 : Spécifications des Ports sous HiLeS Designer

3.2.1.3 Interprétation des blocs fonctionnels

Les blocs fonctionnels représentent également une entité. Au niveau N, l'utilisateur peut nommer l'entité, définir le generic map. Le port map est généré automatiquement en fonction des différentes connexions avec les autres blocs (les affirmations faites dans le paragraphe précédent sur les ports sont valables ici aussi, c'est à dire que les ports sont créés mais leurs caractéristiques restent à définir). Concernant l'architecture de l'entité, elle est accessible par l'utilisateur directement au niveau N par le biais d'une fenêtre. L'utilisateur peut écrire directement le fichier VHDL-AMS de l'architecture de l'entité. On ne peut donc pas descendre d'un niveau dans un bloc fonctionnel. Seule la partie architecture est à écrire par l'utilisateur. Le nom de l'architecture peut être spécifié.

L'utilisateur doit pouvoir s'il le souhaite créer plusieurs architectures pour une même entité. Ainsi il peut réutiliser son entité créée en utilisant l'architecture qu'il souhaite. Le choix de l'architecture doit se faire parmi une liste. La traduction de cette possibilité en VHDL-AMS consiste à créer une configuration ou une instanciation du type : *composant : entité(architecture)*.

Pour profiter pleinement de la norme VHDL-AMS, il est préférable de raisonner en terme d'unité de conception (code compilable seul). Ainsi il existe cinq types d'unité de conception : l'entité, l'architecture, la configuration, le paquetage (déclaration et corps). Néanmoins une architecture par exemple ne pourra être compilée que si l'entité à laquelle elle est associée a d'ores et déjà été compilée. Il faut mettre en place un système d'étiquettes afin qu'une unité de conception faisant appel à une autre ne soit pas compilée avant celle-ci.

3.2.1.4 Interprétation des canaux

Dans le langage HiLeS, les canaux servent à relier les blocs et leur réseau de contrôle entre eux. D'après [Jim00], ils modélisent le flot de matière, d'énergie et d'informations entre blocs structurels. Ils transportent deux modes d'information : des modes continus et des modes d'événement. L'ensemble des canaux représentent le flot

de données. Dans le paragraphe 3.2.2 nous parlerons des flots de commande modélisés par les arcs des réseaux de Petri.

Les canaux discrets transportent les informations concernant le monde logique (numérique). Les canaux continus transportent les informations concernant le monde analogique.

Par le biais de l'interface graphique d'HiLeS, l'utilisateur peut relier simplement le réseau de contrôle et les blocs entre eux. Cette opération en VHDL-AMS consiste à créer des objets locaux (signal, terminal, quantité) au sein d'une architecture pour relier les différents composants.

La traduction d'un canal discret en VHDL-AMS se fait grâce à l'objet SIGNAL. On doit lui associer un type (bit, booléen). Concernant le canal continu, nous avons le choix entre utiliser l'objet TERMINAL ou l'objet QUANTITY. Les terminaux se comportent comme des nœuds de connexion et ne portent pas de valeur. Ils possèdent une nature définie par deux types réels concernant le flux (THROUGH) à travers et l'effort (ACROSS) aux bornes du terminal. Les quantités, quant à elles, sont des fonctions continues du temps ou de la fréquence. Elles peuvent être associées ou non à un terminal (quantité de branche ou quantité free).

L'utilisateur peut nommer le canal, mais aussi lui donner un type ou une nature. Ces caractéristiques sont étroitement liées avec celles des ports auxquels est relié le canal. Ainsi si l'on relie le canal à un port possédant un type déjà défini, le canal possédera automatiquement ce type. La norme VHDL-AMS impose un typage fort qui permet de ne connecter ensemble que les éléments compatibles.

Les différents points évoqués ci-dessus montrent la faisabilité de traduction en VHDL-AMS, les modèles fonctionnels, structurels et les canaux du formalisme HiLeS. Nous allons maintenant étudier la partie commande de ce formalisme qui est basé sur les Réseaux de Petri.

3.2.2 Traduction des Réseaux de Petri

Le modèle de commande d'un système est basé dans l'outil HiLeS sur des réseaux de Petri. Ils réalisent l'exécution des commandes de contrôle et la synchronisation des blocs. Un réseau de Petri est composé de quatre éléments de base : un ensemble de places, un ensemble de transitions, une fonction d'entrée (des transitions vers les places) et une fonction de sortie (des places vers les transitions).

Dans le langage HiLeS, le flot de contrôle est représenté par le biais des arcs du réseau de Petri. La modélisation de ces arcs en VHDL-AMS dépend de la manière avec laquelle on décrit le réseau de Petri, de façon structurelle ou comportementale.

L'objectif ici est de décrire dans le langage VHDL-AMS le réseau de Petri. Pour ce faire on dispose de plusieurs possibilités. On peut tout d'abord transformer le réseau de Petri en une FSM (Finite State Machine : machine d'états finis) facilement modélisable en VHDL-AMS. Les FSM consistent en un ensemble d'états, d'entrées et de sorties, ainsi qu'en une fonction de transition permettant le calcul de l'état suivant à partir des entrées et de l'état courant. Cependant les FSM ne sont pas dédiées à l'expression de la concurrence ainsi lorsque la complexité de la spécification augmente, va apparaitre une croissance exponentielle du nombre d'états. Les FSM sont bien adaptées pour modéliser des processus séquentiels. Mais les réseaux de Petri représentent quant à eux des processus concurrents, ainsi un simple réseau de Petri peut devenir une FSM complexe.

Une méthodologie efficace, appliquée en industrie, est proposée par les auteurs de la référence [FAP97]. La méthode permet la génération automatique de code VHDL et

supporte les spécifications hiérarchiques. Pour garder un isomorphisme entre les spécifications initiales du réseau de Petri et les instructions VHDL, on utilise le langage intermédiaire CONPAR (cf. [FAP97]).

Dans le langage CONPAR, les réseaux de Petri interprétés sont traduits en spécifications « rule-based », qui sont composées de symboles d'état discret et de signaux d'entrée et de sortie. Les règles des transitions d'état discret décrivent un changement d'état local.

Les travaux menés à ce sujet ont été concluants (cf. [Gui03] et [Ham05]). Cependant le fait de passer par une représentation intermédiaire est contraignant, surtout si la complexité du réseau de Petri traduit augmente. Cela amene à regarder une approche de traduction plus immédiate, c'est-à-dire avec une correspondance directe entre places et transitions et leurs modèles équivalents.

Des travaux ont été faits dans ce sens démontrant la faisabilité de cette traduction. Ainsi dans les travaux de A. Nketsa [CEN04] ont été définis des modèles VHDL de place et de transition adaptés au problème de la traduction de la solution logique HiLeS en VHDL. Cela a notamment été adapté et mis en œuvre par J.C Hamon [Ham05] et J. Baylac [BHE04].

Il ressort de ces techniques de traduction des réseaux de Petri en VHDL une automatisation difficile à mettre en place. La transformation de modèle via les meta modèles est une alternative possible. Des travaux ont été menés dans ce sens par V. Albert [Alb05]. Le principe de sa méthode est basé sur le concept de meta modèle et de transformation au niveau meta modèle. Il a créé le meta modèle du formalisme des réseaux de Petri ainsi que celui du VHDL (cf. 2.4). Il a écrit un programme informatique en java qui permet de manipuler des modèles de réseaux de Petri pour en générer des modèles VHDL équivalents, ainsi toutes ses règles de transformations ont été écrites en Java. Ici les meta modèles ne servent que de référence pour le programmeur et le programme ne manipule que les modèles d'entrée et de sortie.

Il ressort de cette étude que le passage du formalisme HiLeS vers le formalisme VHDL-AMS se fait sans perte sémantique. On obtient par transformation une représentation en VHDL-AMS de la solution logique sans perte vis-à-vis de sa version sous HiLeS. Ce modèle de la solution logique doit ensuite être enrichi des contraintes et choix technologiques pour permettre d'obtenir un modèle de la solution physique.

3.3 Conception directe des Modèles en VHDL-AMS

3.3.1 Méthodologie mise en place

Nous venons de montrer la faisabilité du passage de la solution logique vers la solution physique ayant pour support le VHDL-AMS. Nous allons maintenant nous pencher sur d'autres types de convergence que l'on rencontre visant la solution physique. Pour nos premiers travaux au LAAS-CNRS, nous avons mis en place une méthodologie de conception de modèle afin d'améliorer la rapidité de traduction et la qualité du modèle généré. La Figure 58 schématise la démarche correspondante. Le point d'entrée peut être soit le code du modèle source s'il est disponible écrit dans un langage de description de matériel, soit uniquement les spécifications du système à modéliser.

Une des applications de cette démarche a été de générer un modèle VHDL-AMS à partir d'un modèle écrit dans un langage de description de matériel, le MAST, qui est un langage propriétaire de l'outil Saber [Sab]. Selon la nature de ce modèle (analogique, numérique ou mixte), des étapes différentes sont appliquées, utilisant ou non certains outils. Ainsi, le logiciel Paragon, étudié au cours du deuxième chapitre (cf. paragraphe 2.2.2) peut être utilisé lorsque l'on a un modèle analogique et/ou mixte afin

d'automatiser le processus. Cependant il reste toujours, pour le moment, des étapes de mise au point pour corriger les éventuelles erreurs et difficultés. Pour les modèles numériques, le logiciel Paragon n'est pas utilisable : la traduction se fait essentiellement manuellement. Seuls les modèles MAST analogiques peuvent être actuellement interprétés par Paragon. Mais, certaines interprétations peuvent être imprécises, c'est pourquoi des étapes de vérifications doivent être insérées dans la méthodologie.

L'étape de validation du modèle s'effectue par la création d'un banc d'essai (testbench) et d'un vecteur de test permettant de vérifier que les spécifications et les contraintes liées au modèle source sont respectées. Cette étape de validation du modèle s'appuie sur le logiciel SystemVision [Sys50] : Cet outil permet de modéliser des systèmes pouvant être analogiques, numériques ou mixtes. Il a été utilisé pour simuler les modèles VHDL-AMS générés. En cas de divergence, il est nécessaire de revenir à l'étape de conception du modèle. Si le modèle est validé, il est stocké dans une bibliothèque de modèles grâce à laquelle il pourra être réutilisé dans un autre système, par instanciation notamment.

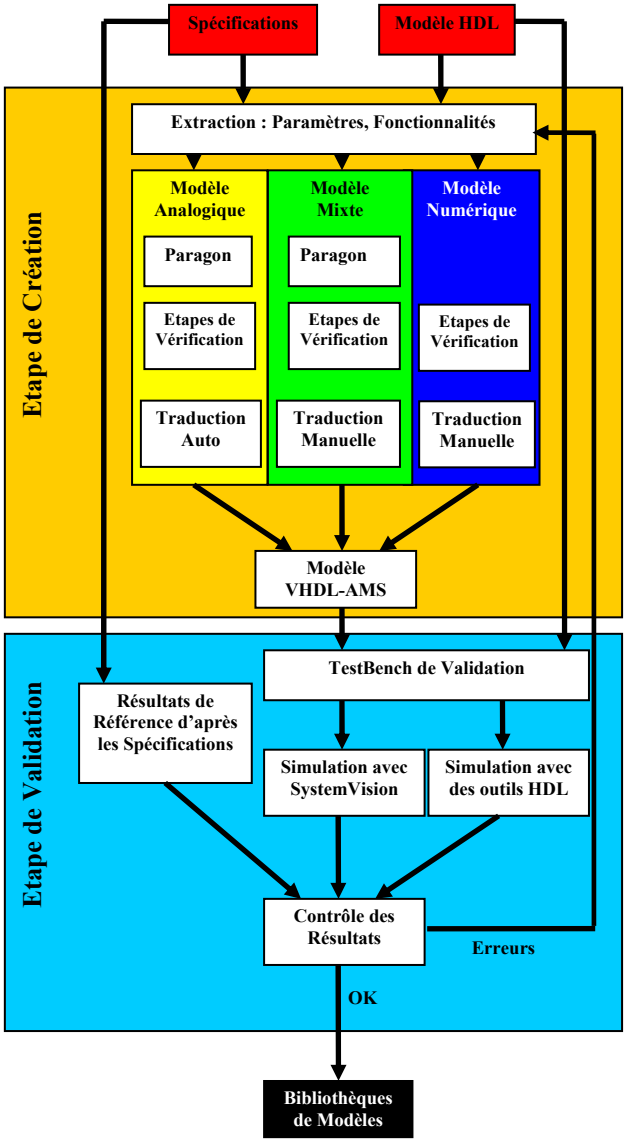


Figure 60 : Organigramme de création directe de modèles VHDL-AMS

3.3.2 Techniques et traduction manuelle

Actuellement, il est encore nécessaire d'insérer des étapes de traduction manuelle pour compléter certaines lacunes actuelles des outils. Dans la littérature, de nombreux travaux décrivant les langages VHDL-AMS et MAST nous ont permis d'élaborer des correspondances entre les différents concepts de modélisation des deux langages [Her02].

Dans le Tableau 6, sont cités quelques éléments importants traduits dans les deux langages. ON retrouve entre le VHDL-AMS et le MAST des concepts de modélisation équivalents (détection de seuil, affectation de signaux, équations physiques). Leurs mises en œuvre est néanmoins différentes. Le VHDL-AMS profite des avancés du VHDL en matière de modélisation de systèmes numériques, pour proposer des instructions dédiées à la manipulation des variables discrètes parfaitement adaptées (process, wait, affectation de signaux) ce qui n'est pas le cas pour le MAST.

En pratique, certaines traductions n'ont pu être mises en œuvre, essentiellement du fait de certaines limitations des outils utilisés, avec notamment une couverture de la norme VHDL-AMS encore incomplète. L'amélioration de ce point serait bénéfique pour notre solution physique en VHDL-AMS, non seulement pour les possibilités de modélisation mais aussi pour la simulation de notre modèle. Ainsi il y a de nombreux intérêts à intégrer dans les outils le support de certaines instructions, notamment de l'instruction procedural qui même si elle n'influe pas sur la précision de la simulation, s'avère être la traduction sémantiquement la plus proche de la section *value* d'un modèle MAST. Cette instruction permet de réduire la taille du système d'équations simultanées, ce qui allègera les calculs du simulateur et permettra un gain de temps de simulation. A noter que, sous la pression de besoins industriels, les outils VHDL-AMS s'améliorent rapidement. Ainsi certaines limitations actuelles vont être rapidement levées sous SystemVision notamment.

Concepts	MAST	VHDL-AMS
Détection Seuil Analogique	Fonction Seuil (variables beforestate et afterstate)	Attribut `above quantity
Manipulation séquentielle des valeurs à temps continu	Section *Value*	Procedural
Dérivation	d_by_dt (ordre 1)	`dot
Intégration	impossible	`integ
Quantité de branche Type through	branch … = i (… -> …)	Quantity … through … to … ;
Quantité de branche Type across	branch … = v (… , …)	Quantity … across … to … ;
Synchronisation Simulateurs numériques et analogiques	Fonction Schedule_next_time	Instruction Break
Détection Evènement numérique	Fonction Event_on	Liste Sensibilité Instruction Wait dans Process
Affectation de signaux numériques	Fonction Schedule_event	<=

Tableau 6 : Traduction en langage MAST et VHDL-AMS de plusieurs concepts de modélisation

Nous pouvons donc constater que cette approche implique d'une part des experts en conception dans les langages de modélisation source et cible, qui vont tirer d'un modèle source ses fonctionnalités pour les restituer dans le modèle cible (portage sémantique [Her06]). D'autre part, l'approche implique des spécialistes qui, à l'aide d'équivalences entre les deux langages (comme par exemple celles du Tableau précédent) vont créer le modèle cible à partir du modèle source (portage par identification [Her06]). L'intégration de ces types de portage dans notre méthodologie ne va cependant pas favoriser son automatisation.

Cette méthodologie a été utilisée dans nos premiers travaux de modélisation VHDL-AMS. Elle a été mise en place pour répondre à un besoin industriel d'obtenir des modèles VHDL-AMS dans un délai court, elle n'a donc pas pu être optimisée. Mais cela nous a permis d'en tirer des enseignements qui nous ont amenés à envisager d'autres concepts à mettre en jeu, notamment celui de la meta modélisation (cf. le paragraphe 2.3 sur le meta modélisation, et le paragraphe 3.4 sur la transformation de modèle au niveau meta modèle). Nous avons fait notamment deux publications mettant en jeu cette méthodologie dans des conférences internationales [GAE06a] [GAE06b].

3.4 D'un Langage de description de matériel quelconque vers le VHDL-AMS

Dans notre processus de conception de système, le passage entre la solution logique et la solution physique est une étape très importante qui a été traitée dans le paragraphe précédent. Outre ces questions sur l'héritage de la solution logique dans la solution physique qui ont impliquées une étape de transformation de modèle, il existe d'autres cas de figure qui nous amenent à converger vers des modèles en VHDL-AMS. Notamment pour des problématiques d'intégration de modèles d'ores et déjà créés, ou de migration de bibliothèques, il est à présent incontournable de réaliser des transformations de modèle horizontal de type exogène. Une des principales problématiques, qui nous a amenée à mener ce travail de thèse, est un besoin de différents industriels à migrer leurs bibliothèques de modèles écrits dans un langage de description de matériel quelconque vers la norme VHDL-AMS. Cela implique un certain nombre de question sur la méthode à employer pour réaliser cette migration, notamment au sujet de la méthodologie à employer, des experts à impliquer dans le processus, et sur l'évaluation du temps que prendra cette migration de bibliothèque.

Comme évoqué dans le paragraphe 3.3, la transformation de modèle entre deux langages de description de matériel à partir du code source peut impliquer plusieurs types de portage [Her06]. On a vu dans le premier chapitre les deux premiers types de portage, dont l'un impliquait la mise en jeu d'experts au cours d'un portage sémantique, et l'autre impliquait la mise en jeu de spécialistes au cours d'un portage par identification. Ces types de portage notamment sémantique, ont été mis en jeu au cours de nos travaux de thèse pour la réalisation de modèles VHDL-AMS présentés dans le dernier chapitre de ce mémoire.

Dans ce paragraphe, nous allons plus particulièrement nous attarder sur la mise en œuvre du troisième type de portage se basant sur l'abstraction et la meta modélisation. Ainsi nous allons mettre en place une transformation de modèle à base de meta modèle pour répondre à nos objectifs. Nous nous sommes attardés au cours du deuxième chapitre sur la notion de meta modèle et nous l'avons appliqué aux différents formalismes rencontrés au sein de notre processus de conception. Ainsi nous avons créé au cours de ce deuxième chapitre le meta modèle du formalisme HiLeS qui est le support de la solution logique, et nous avons aussi proposé un meta modèle du VHDL-AMS qui est notre support de la solution physique. Nous avons introduit le principe de ce type de transformation dans l'introduction général et dans le paragraphe 2.2.3. Il est notamment représenté sur la Figure 4, avec les meta modèles source et cible et des règles de transformation qui sont définis au niveau des meta modèles.

3.4.1 Meta Modèle VHDL-AMS en vue d'une transformation MAST vers VHDL-AMS

Nous allons appliquer cette démarche à un cas de transformation particulièrement motivé par des intérêts industriels forts à savoir la transformation de modèles écrits en MAST (Langage propriétaire SYNOPSYS lié à l'outil Saber [Sab]) vers la norme VHDL-AMS. La transformation se fait ici sans perte sémantique.

Notre meta modèle a été créé au cours du deuxième chapitre (cf. 2.4.2), nous avions cependant conclu sur le fait que en utilisant la démarche de création passant par la grammaire abstraite, nous obtenions un meta modèle lourd et difficile à utiliser. Ce meta modèle est surtout dédié à une manipulation par une plateforme pour vérifier la conformité des modèles créés. Dans ce paragraphe, nous souhaitons écrire des règles simples de transformation entre le meta modèle MAST et celui du VHDL-AMS, en profitant notamment du fait que ces deux langages de description de matériel sont proches et que leurs concepts de modélisation ont de nombreux points communs. On tend ainsi vers le concept d'un meta modèle commun aux langages de description de matériel comme on peut le retrouver dans l'écriture de la DTD utilisé dans le logiciel Paragon dont nous avons parlé au paragraphe 1.3.3.

La principale modification que l'on va apporter au meta modèle précédemment développé, est l'apport d'une classe objet, qui va généraliser les différents types d'objets utilisés dans les différentes instructions. Dans cette classe objet, on retrouve entre autres :

- Les Terminaux,
- Les Quantity,
- Les Signaux,
- Les Variables,
- Les Constantes,
- Les Fichiers.

Ces différents objets vont trouver leurs équivalents dans le langage MAST et nous pourrons en écrire les correspondances au sein des règles de notre modèle de transformation. Pour le reste, nous conservons les différents concepts de modélisation de notre premier meta modèle qui vont avoir leurs équivalents dans le langage MAST, avec notamment les différents types d'instructions que nous pouvons rencontrer dans le langage, à savoir :

- les instructions concurrentes,
- les instructions simultanées,
- les instructions séquentielles,
- les instructions de déclaration.

Le nouveau meta modèle ainsi créé, représenté sur la Figure 59, reprend les éléments caractéristiques du meta meta modèle MOF, à savoir les classes et les relations de composition et de généralisation. A cela sont ajoutés des contraintes via OCL, comme explicité dans le paragraphe de création du meta modèle du formalisme HiLeS (cf. 2.3.4). Cette décomposition implique l'ajout de certaines contraintes comme par exemple le fait que les objets « variables » ne peuvent être utilisés que dans des process ou des sous-programmes.

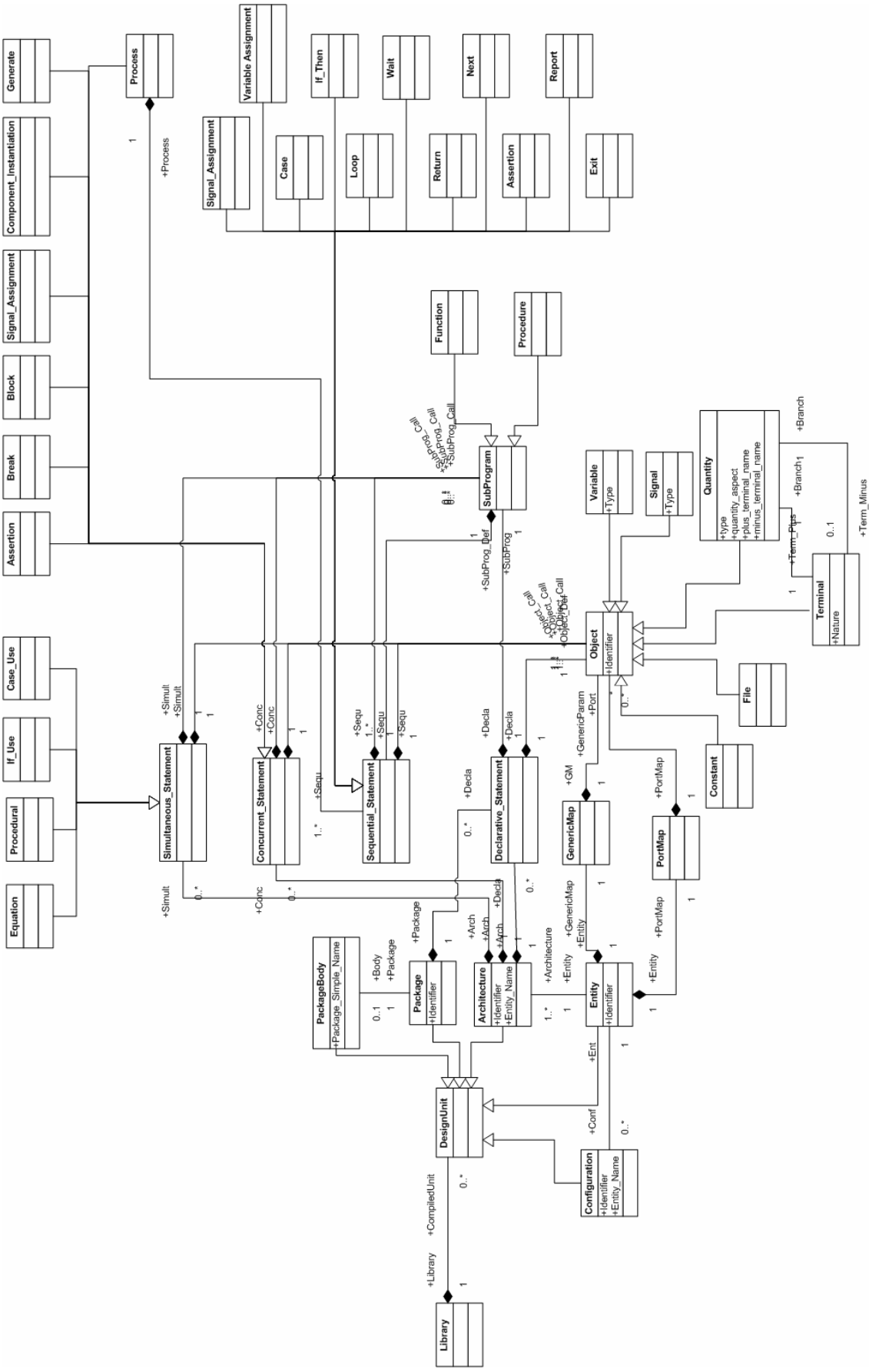


Figure 61 : Meta Modèle VHDL-AMS complet (premiers niveaux) et modifié pour transformation entre langages de description de matériel

3.4.2 Création du modèle de transformation

Afin d'exprimer les différentes règles de transformations, nous allons créer un modèle de transformation dont le langage sera ATL, dont nous avons présenté le principe au cours du deuxième chapitre. ATL est un langage permettant d'exprimer les règles de transformations, mais aussi un outil permettant l'exécution de ces règles au sein d'une plateforme (Eclipse) capable de manipuler les meta modèles. Le meta meta modèle utilisé sous Eclipse est l'Ecore, très proche du MOF. Il est d'ailleurs sous ATL possible d'en effectuer la transformation entre un meta modèle exprimé sous MOF vers un meta modèle exprimé sous Ecore. Les meta modèles sont sous la forme de fichiers XMI (XML Metadata Interchange), qui sont des fichiers XML dont la DTD définit les labels permettant la sauvegarde d'informations relatives aux meta modèles. Il faut noter qu'ATL a aussi besoin d'avoir le modèle à transformer en entrée sous cette forme XMI. Cela nécessite une étape préliminaire sur le modèle source permettant d'obtenir ce modèle sous la forme d'une instanciation du meta modèle sous forme XMI (Injecteur cf. Figure 60). Des travaux ont été réalisés dans ce sens par J. Delatour [SDC06], avec la création de meta modèle des langages VHDL et *Logic*, et la création de règles de transformation sous ATL. On retrouve ses différentes étapes dans la Figure 60, avec la représentation de cette méthodologie.

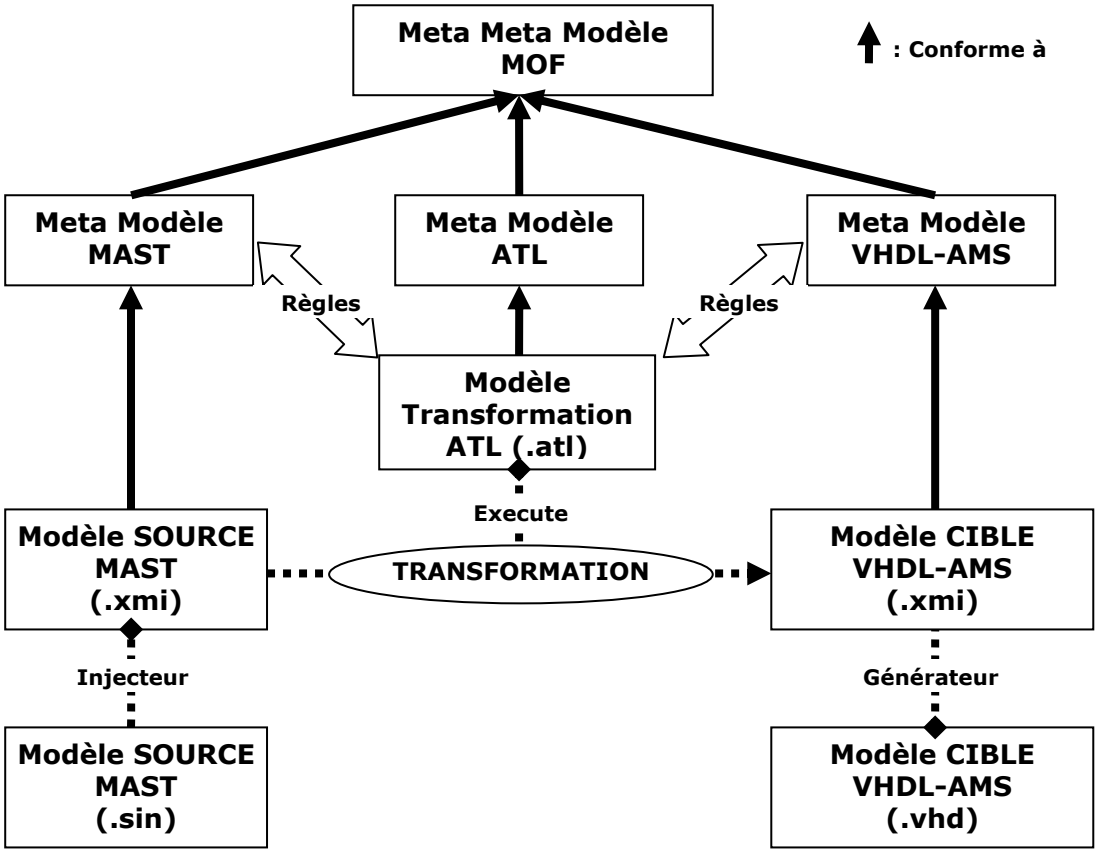


Figure 62 : Méthodologie Transformation de Modèle via Meta modèles et ATL

Pour illustrer ce travail, nous allons reprendre le modèle de la résistance, qui nous avait permis dans le deuxième chapitre d'illustrer la création du meta modèle du langage VHDL-AMS (cf. 2.4.2). Cet exemple simple analogique va nous permettre d'exprimer simplement les règles de transformation. Outre le modèle VHDL-AMS d'ores et déjà présenté sur la Figure 39, la Figure 61 présente le même modèle de résistance électrique exprimé dans le langage MAST.

```
template resistor p m = res

electrical p, m

number res

{
branch ir = i(p->m)
branch vr = v(p,m)

equations {
            ir = vr/res
      }
}
```

Figure 63 : Modèle MAST d'une résistance

Pour effectuer cette transformation, nous avons besoin en entrée du meta modèle du langage MAST. Un travail sur la grammaire du langage [Syn04] a permis d'en tirer un meta modèle simplifié et présenté sur la Figure 62, ne représentant que les concepts mis en jeu dans le cas du modèle de la résistance. De manière identique, nous avons repris le meta modèle du VHDL-AMS que nous avons défini dans le paragraphe précédent et nous l'avons simplifié aux seuls concepts de modélisation utilisés dans le modèle de la résistance (cf. Figure 63).

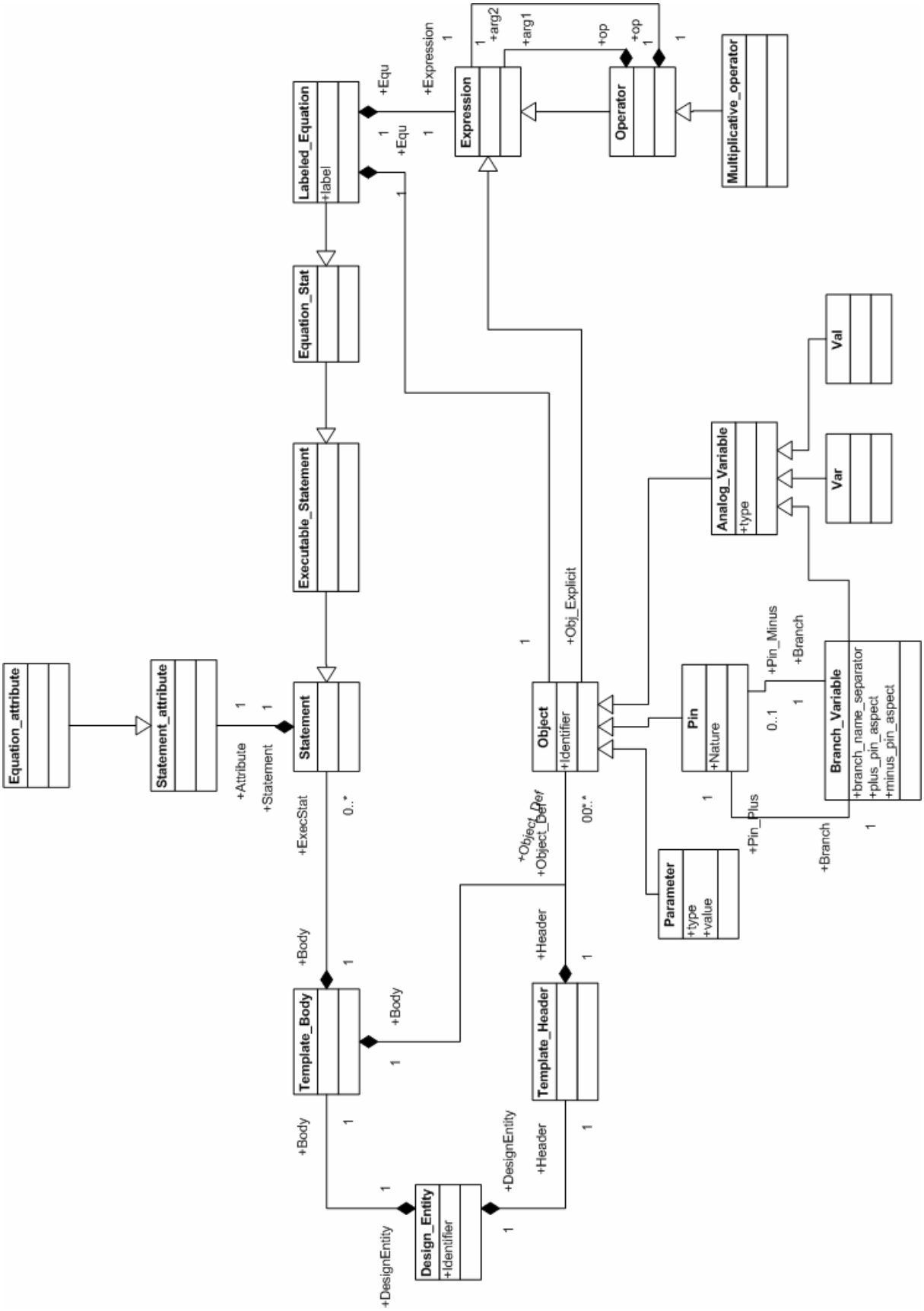


Figure 64 : Meta Modèle MAST simplifié pour modélisation d'une résistance

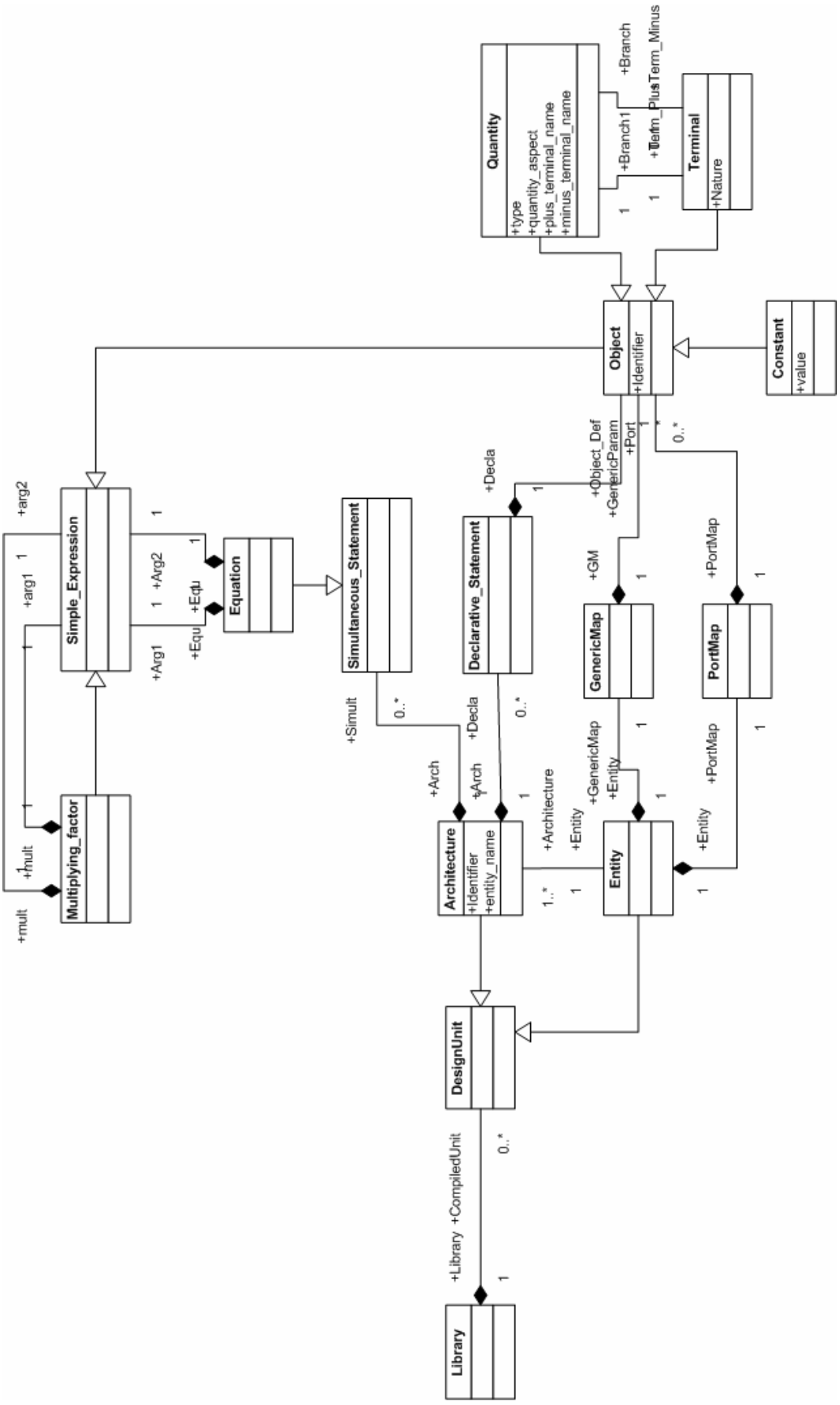


Figure 65 : Meta Modèle VHDL-AMS simplifié pour modélisation d'une résistance

Le modèle MAST de la résistance est exprimé sous forme d'une instanciation du meta modèle MAST, lui même sous la forme d'un fichier XMI, afin de pouvoir être manipulé par l'outil ATL. Le fichier XMI de ce modèle MAST de la résistance, conforme au meta modèle MAST donné sur la Figure 62, est donné en ANNEXE 1.

L'écriture des règles se fait simplement, en manipulant les éléments des meta modèles pour en définir un certain nombre de règles permettant de faire la correspondance entre les deux langages (cf. la méthodologie décrite sur la Figure 60).

Nous dressons de façon informelle les règles de transformations à appliquer sur ces deux meta modèles :

- **Règle 1** : **Header2Entity**
  - Pour chaque Template_Header MAST, une Entité VHDL-AMS doit être créée
  - Le nom doit être le même
  - Le portmap de l'entité VHDL-AMS doit contenir la déclaration des objets générés par la règle **Pin2Terminal**
  - Le genericmap de l'entité VHDL-AMS doit contenir la déclaration des objets générés par la règle **Param2Constant**

- **Règle 2** : **Body2Architecture**
  - Pour chaque Template_Body MAST, une Architecture VHDL-AMS doit être créée
  - La référence à l'entité doit être la même
  - La partie déclarative de l'architecture doit contenir la déclaration des objets générés par la règle **Branch2QuantityBranch**
  - La partie instructions simultanées doit contenir les instructions simultanées générées par **Lab_Equ2Equ**

- **Règle 3** : **Param2Constant**
  - Pour chaque Paramètre MAST, une constante VHDL-AMS doit être créée
  - Le nom doit être le même
  - Le type doit être le même
  - La valeur par défaut doit être la même

- **Règle 4** : **Pin2Terminal**
  - Pour chaque Pin MAST de type conservatif, un Terminal VHDL-AMS doit être créé
  - Le nom doit être le même
  - La nature doit être la même

- **Règle 5** : **Lab_Equ2Equ**
  - Pour chaque Labeled_Equation MAST, une Equation VHDL-AMS doit être créée
  - Le premier argument de l'équation doit contenir l'objet appelé explicitement
  - Le deuxième argument de l'équation doit contenir les expressions générées par les règles **Object2Object** et **MultiMAST2MultiVHDLAMS**

- **Règle 6 : Object2Object**
  - Pour chaque Object MAST, un Object VHDL-AMS doit être créé
  - Le nom doit être le même

- **Règle 7 : MultiMAST2MultiVHDLAMS**
  - Pour chaque opération de multiplication MAST, une opération de multiplication VHDL-AMS doit être créée
  - Les deux arguments doivent être les mêmes

- **Règle 8 : Branch2QuantityBranch**
  - Pour chaque Branch_Variable MAST, une Quantité VHDL-AMS doit être créée
  - Le nom, le type, la valeur par défaut doivent être les mêmes
  - Les noms des terminaux plus et moins doivent être les mêmes
  - L'aspect de la quantity VHDL-AMS reprend l'aspect de la quantity de branche MAST

Ces règles sont ensuite exprimées dans le langage ATL au sein d'un modèle de transformation. Le langage ATL est un langage hybride déclaratif et impératif, mais le plus souvent déclaratif comme pour l'écriture des règles appelées *'rule'.* Une règle déclarative spécifie :

- Une partie source qui doit être vérifiée dans le modèle source de la transformation pour que la règle se mette en œuvre. Cela permet de faire des filtres,
- Une partie cible qui doit créer dans le modèle cible à chaque correspondance durant l'exécution de la règle.

Il est aussi possible de créer des fonctions appelées *'helper'* qui permettent de donner un résultat après manipulation des meta données d'entrée.

Le code ATL des huit règles décrites ci-dessus est donné dans l'ANNEXE 2.

3.4.3 Conclusions sur la méthode

Au cours de ce paragraphe, nous avons mis en avant la faisabilité de transformer des modèles analogiques MAST vers des modèles VHDL-AMS en appliquant une méthodologie de transformation de modèles avec définition de règles de transformation au niveau meta modèle. En profitant que les deux langages sont très proches et qu'ils possèdent de nombreux concepts de modélisation en commun, **nous avons créés leurs meta modèles de manière à faire ressortir ces points communs, et avoir ainsi à définir des règles de transformation les plus simples possibles**. La mise en œuvre de ce type de transformation permet de rendre automatisable le processus en se conformant aux normes de meta modélisation et de transformation utilisées dans des outils ou des plateformes comme Eclipse [Ecl] ou Topcased [Top06].

Nous n'avons appliqué cette méthode que sur des modèles de type analogique. Dans le cadre de modèles numériques, des exemples comme les travaux de J. Delatour [SDC06], ont montré qu'une utilisation de la méthode de transformation de modèle basée sur les meta modèles était aussi réalisable. Dans notre cas de transformation MAST vers VHDL-AMS, les concepts de modélisation mis en jeu dans la partie numérique sont différents et il devient difficile d'écrire des règles simples de transposition permettant une automatisation complète du processus. Il est ainsi toujours préconisé d'avoir l'apport d'un expert dans les deux langages, qui va être capable de puiser les fonctionnalités comprises dans les nombreuses sections numériques MAST (*when section*), pour les retranscrire en VHDL-AMS en profitant des nombreuses instructions du langage adaptées à cela comme les instructions *process*, *wait*, affectation de signaux,…

La mise en place d'une telle démarche de transformation reste encore assez difficile et lourde, avec notamment la définition d'un très grand nombre de règles surtout si l'on utilise des meta modèles créés directement à partir des grammaires des langages. De plus, il faut aussi créer un analyseur de code en entrée (injecteur) pour générer une version XMI du modèle conformément à son meta modèle, mais aussi créer un générateur de code (extracteur) qui en fonction du modèle XMI obtenu en sorti et

conformément à son meta modèle, génèrera le code du modèle souhaité suivant la syntaxe concrète du VHDL-AMS.

Comme nous l'avons noté au cours du premier chapitre (paragraphe 1.3.3), de nombreux outils de modélisation et de simulation, utilisant des langages de description de matériel, utilisent d'ores et déjà des approches XML pour le stockage des données relatives aux modèles créés. Nous l'avons vu, la définition de la grammaire de ces fichiers (DTD pour Paragon ou XSD pour SystemVision) est directement liée au concept de meta modélisation prouvant que la tendance actuelle est bien tournée vers la meta modélisation. La transformation de modèle parait donc une approche attractive comme sa mise en place dans des plateformes telles qu'Eclipse ou Topcased le démontrent.

3.5 Conclusion

Les deux premiers chapitres de ce mémoire nous ont permis de présenter la problématique dans laquelle nous nous plaçons, en résumé la recherche d'une solution physique dans le cadre d'un processus de conception d'un système hétérogène. Notre étude des langages de modélisation dans le premier chapitre nous a amené à proposer l'utilisation du langage de description de matériel normalisé VHDL-AMS en tant que support de cette solution physique. Le deuxième chapitre a été l'occasion d'introduire les différentes méthodes de création et de transformation de modèles, que nous avons appliquées dans ce troisième chapitre, en ayant cette fois-ci comme objectif et cible une modélisation en VHDL-AMS. Nous nous sommes efforcés pour chaque méthodologie d'en évaluer la faisabilité et le degré d'automatisation possible.

Dans ce troisième chapitre, nous avons étudié une première méthodologie pour créer des modèles VHDL-AMS à partir des spécifications du système à modéliser ou d'un modèle d'ores et déjà créé dans un langage de modélisation différent. Cette méthodologie nécessite une grande part d'expertise qui, en contre partie, va diminuer le degré d'automatisation. Néanmoins, des outils comme Paragon peuvent contribuer à rendre plus efficace cette démarche. Nous avons exporté cette méthode dans un grand nombre de nos premiers travaux de cette thèse, car elle est rapide à mettre en place et permet une optimisation rapide du code cible notamment par l'intervention d'un expert dans les deux langages de modélisation source et cible.

Les limites de cette première approche nous a conduit vers de nouveaux concepts de transformations. Il s'agit de palier le manque d'automatisation de la première méthodologie, mais aussi d'intégrer, dans notre processus de conception, des concepts de transformations normalisés mis en jeu dans plusieurs outils et plateformes industriels comme Eclipse ou Topcased.

Ainsi, nous avons, au cours du deuxième chapitre, introduit le concept de meta modélisation et de transformation de modèles avec définition des règles de transformation au niveau meta modèle : Nous l'avons appliqué au cours de ce troisième chapitre à la problématique de convergence vers le VHDL-AMS. Nous avons utilisé cette méthode dans le cas particulier d'une transformation de modèle horizontale exogène entre le langage MAST et le langage VHDL-AMS. Nous avons créé un meta modèle du formalisme MAST et modifié le meta modèle du formalisme VHDL-AMS créé dans le deuxième chapitre dans l'optique de ce type de transformation. Cela dans le but de profiter du fait que les concepts de modélisation de ces deux langages sont proches permettant d'écrire des règles de transformation au niveau meta modèle les plus simples possibles. Nous montrons la faisabilité d'une telle approche en développant un modèle de transformation utilisant le langage de transformation ATL pour écrire les différentes règles nécessaires de la transformation d'un modèle MAST simple analogique vers son équivalent en VHDL-AMS.

Comme nous l'avons fait observer dans le paragraphe 3.4.3, cette méthodologie s'avère tout de même difficile et lourde à mettre en œuvre et nécessite la création d'injecteur et de générateur de code pour utiliser des outils comme ATL (besoin d'avoir en entrée des fichiers de type XMI). Elle semble tout de même être une perspective plausible étant donné les tendances actuelles qui voient de nombreux outils comme SystemVision ou Paragon utiliser des concepts (XSD, DTD,…) proches de celui de meta modélisation pour la sauvegarde des données relatives aux modèles VHDL-AMS notamment.

Ce troisième chapitre a aussi été l'occasion de rappeler les différents travaux menés à bien permettant le passage entre la solution logique vers la solution physique qui nous intéresse. Comme nous l'avons précisé dans les chapitres précédents, le support de notre solution logique dans notre processus de conception est le formalisme HiLeS. Nous avons montré, dans ce troisième chapitre, qu'une transformation d'une description HiLeS est possible sous la forme d'un modèle VHDL-AMS qui est notre choix de représentation pour la solution physique. Nous avons d'ailleurs argumenté ce choix, en modélisant, au cours de ce chapitre, un système hétérogène de mise à feu d'une charge pyrotechnique. Ce sera d'ailleurs un des objectifs du quatrième chapitre qui, sur la base de plusieurs exemples de systèmes réels, démontrera les aptitudes et performances de modélisation de ce formalisme normalisé.

CHAPITRE 4

4. TRANSFORMATIONS ET VALIDATIONS SUR DES EXEMPLES D'APPLICATION

Dans les chapitres précédents, nous avons :

- définit notre processus de conception
- précisé nos besoins d'une solution physique pour la modélisation d'un système hétérogène
- présenté les différents types de modèles que l'on pouvait rencontrer,
- fait le choix d'un langage de modélisation support de cette solution physique : le VHDL-AMS
- définir des méthodologies de création et de transformation de modèle dans un souci de convergence vers le langage cible VHDL-AMS.

Nous allons dans ce quatrième chapitre mettre en œuvre ces différents au travers d'exemples concrets de modélisation en VHDL-AMS, sur des systèmes ayant une provenance académique ou industrielle.

Créer un modèle pour l'intégrer dans un simulateur complexe est une tâche extrêmement délicate car, comme nous l'avons souligné dans le premier chapitre, à un modèle sont toujours associées des limites d'usage qui peuvent avoir, dans le comportement global du système, des conséquences graves. Un modèle doit toujours être accompagné d'un certain nombre d'informations le concernant, permettant notamment de définir son mode et ses limites d'utilisation. Nous en discuterons au cours de ce chapitre, lorsque nous aborderons la question de validation de nos modèles créés et de réutilisabilité de ces modèles dans d'autres contextes.

Où se trouvent les supports de modélisation ? C'est ce que nous avons traité lors des chapitres précédents, en faisant référence aux modèles physiques et aux modèles empiriques, ainsi qu'aux différents niveaux de modélisation qui peuvent coexister dans une simulation. Les types de modèles souhaités doivent être bien définis. Ainsi, en pratique dans l'industrie, les modèles dont dispose le concepteur proviennent de fournisseurs de composants qui, de plus en plus souvent, assurent les caractéristiques techniques de leurs fournitures et la qualité des modèles qui les accompagnent. Ces modèles sont souvent de type boite noire et ne sont que copie du comportement du système dans un cas d'utilisation particulier. Ces modèles n'ont pas de caractère prédictif permettant une optimisation du système : l'émergence d'une norme comme le VHDL-AMS permet d'avoir des modèles plus exportables.

Comme nous l'avons évoqué dans le troisième chapitre, le point de départ de la création d'un modèle peut être les spécifications du système à modéliser (cf. paragraphe 3.3 ou cf. paragraphe 3.1 avec la modélisation du système de mise à feu d'une charge pyrotechnique). Mais ce point de départ peut aussi être un modèle décrit dans un langage de modélisation différent. C'est le cas de certains exemples que nous allons rencontrés au cours de ce chapitre.

Pour introduire l'esprit de ce quatrième chapitre, nous allons donc considérer qu'il existe un modèle SOURCE référant et que nous cherchons un modèle CIBLE écrit en VHDL-AMS. Nous voulons :

- Que le modèle CIBLE soit équivalent au modèle source,
- Que le modèle VHDL-AMS soit conforme au meta modèle établi dans le deuxième chapitre.

Les exemples, qui seront traités dans ce chapitre, nous permettrons d'illustrer cela. Ils sont référencés dans le Tableau 7. D'autres projets de modélisation industriels ont aussi été réalisés durant la thèse, mais des clauses de confidentialité ne permettent pas de les faire figurer dans ce mémoire.

Nom du Modèle	§	Contexte	Origine	Point de départ Transformation
Transistor Bipolaire à Hétérojonction	4.1	Académique	LAAS-CNRS	Code MAST du modèle source
MOSFET de Puissance	4.2	Industriel	INFINEON	Code MAST du modèle source
Capteur Oxygène	4.3	Industriel/Académique	NEOSENS LAAS-CNRS	Spécifications et Prototypes Réels
Fil et Fusible	4.4	Industriel	RENAULT	Spécifications

Tableau 7 : Listes des modèles réalisés et présentés dans le quatrième chapitre

Dans le même temps, ce quatrième chapitre sera aussi l'occasion d'illustrer la capacité du VHDL-AMS à traiter de la pluridisciplinarité comme nous l'avons fait avec l'exemple dans le troisième chapitre d'un dispositif de mise à feu développé par C. Rossi et P. Pennarun. Dans ce même esprit, les exemples de modélisation que nous allons aborder dans ce quatrième chapitre, vont mettre en jeu plusieurs domaines physiques.

Au niveau des méthodologies de création et de transformations de modèle que nous avons explicitées dans le troisième chapitre, il s'avère que pour des contraintes industrielles impliquant un grand nombre de modèles à créer dans un délai très court, la méthodologie requérant une grande part d'expertise et l'utilisation d'outils d'aide comme Paragon (cf. paragraphe 3.3) a été principalement utilisée. La deuxième méthodologie avec notamment le concept de meta modèle a été abordée plus tardivement notamment dans l'optique d'une amélioration en termes d'automatisation du processus, mais aussi dans l'optique d'une conformité avec les normes mises en jeu par les nouvelles plateformes comme Eclipse et Topcased.

4.1 Modèle de Transistor Bipolaire à Hétérojonction

4.1.1 Présentation du Modèle

La méthodologie présentée dans le paragraphe 3.3 est appliquée à un modèle analogique de transistor de puissance développé au laboratoire LAAS-CNRS, dont le code MAST est disponible [And95]. Nous avons fait une publication concernant ce modèle dans la conférence internationale MIXDES [GAE06a].

Le modèle physique non linéaire du transistor bipolaire à hétérojonction (TBH) est schématisé sur la Figure 64.

Sa topologie est basée sur le modèle Gummel-Poon. Les effets d'auto échauffement sont implicitement décrits avec l'ajout d'une cellule calculant la température par le biais de la résistance thermique du composant.

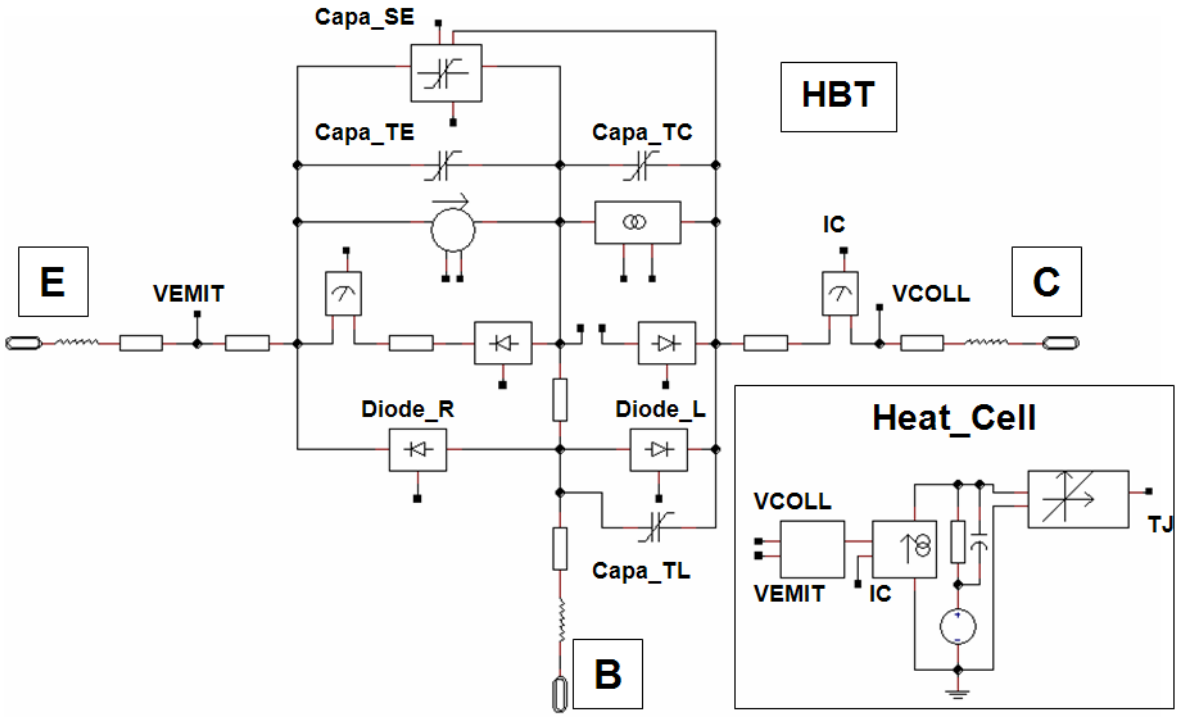


Figure 66 : Modèle Electrothermique du TBH

4.1.2 Méthodologie de création du modèle VHDL-AMS

Une grande partie des sous modèles sont analogiques. Ainsi le logiciel Paragon [PAR04] a pu être utilisé comme point de départ au processus de transformation, avec toutefois des étapes de mise au point manuelle. Le modèle étant hiérarchique, chaque sous modèle a subi le processus de transformation, et a été testé indépendamment dans un testbench adapté.

4.1.3 Résultats

Le modèle du TBH a été testé en régime statique (caractéristiques de sortie) et en régime dynamique petit signal (gain en puissance). Les résultats, tirés de l'interface de visualisation des courbes du logiciel SystemVision, sont représentés sur les Figure 65 (régime statique) et Figure 66 (dynamique petit signal). Les résultats provenant de la simulation du modèle MAST ont été importés de l'outil Saber [Sab].

Pour le domaine statique, on note sur la Figure 65 que les résultats de simulation des modèles MAST et VHDL-AMS sont similaires. Les effets d'auto échauffement sont bien reproduits avec une hausse en température du composant qui entraîne une chute du gain statique, phénomène bien connu dans ce type de transistor (cf. [And95]).

Dans une seconde étape, nous avons étudié les performances dynamiques du système en calculant notamment les paramètres de dispersion (paramètres S) du composant. D'après ces paramètres nous représentons l'évolution du gain en puissance du composant, représenté sur la Figure 66. Nous pouvons noter une bonne corrélation entre le modèle MAST et le modèle VHDL-AMS avec une erreur maximale de 7%.

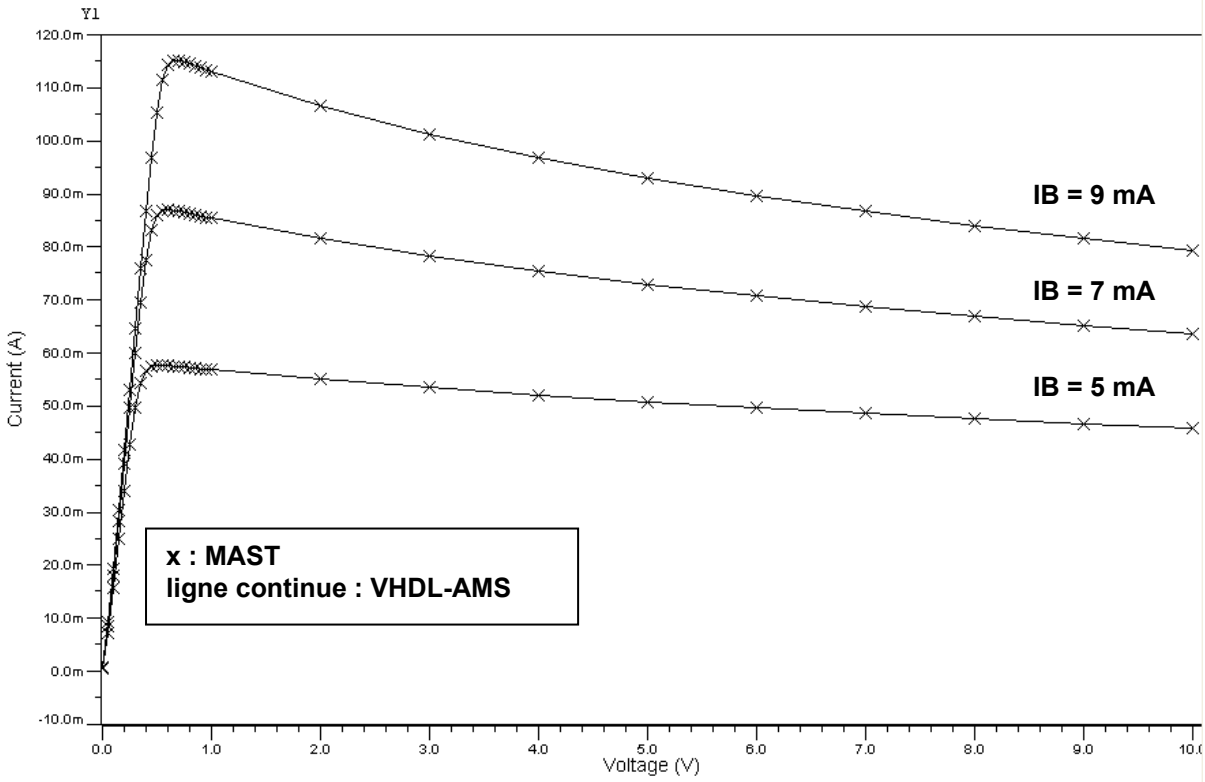


Figure 67 : Caractéristiques (Ic,Vce) avec résistance thermique =150°C/W

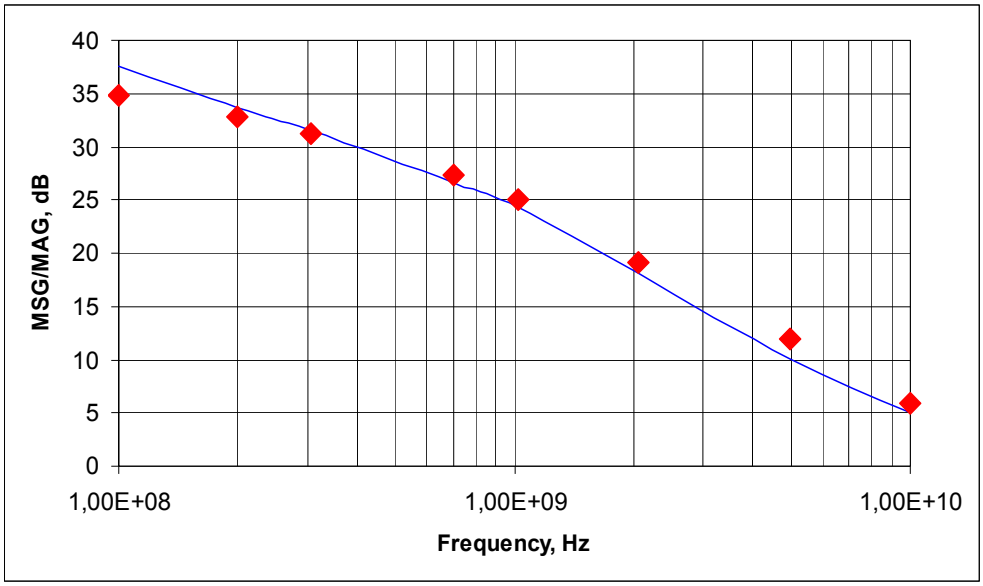


Figure 68 : Gain en puissance pour Rth=150°C/W

Certaines précautions sont cependant à prendre en compte quant à la comparaison des résultats. Ceux-ci provenant de simulateurs différents et les pas de temps étant non fixés, il devient difficile de les comparer finement notamment pour les modèles analogiques. Les réglages des simulateurs, en termes d'algorithme employé, méthode d'intégration, précision, ont aussi un rôle très important dans les résultats de simulation et peuvent expliquer les divergences que l'on obtient. De plus l'erreur faite entre le modèle source et le modèle cible est à mettre en rapport avec l'erreur du modèle source vis-à-vis de la réalité. De manière générale, le modèle source doit être accompagné par

le testbench qui a servi à le valider, mais aussi par un gabarit correspondant à l'erreur faite vis-à-vis de la réalité. Nous en parlerons dans le paragraphe 4.6 de ce chapitre.

4.2 Modèle de MOSFET de puissance

4.2.1 Présentation du Modèle

La méthodologie présentée dans le paragraphe 3.3 est appliquée à un modèle analogique de MOSFET de puissance canal N « OptiMOS2 » de la société INFINEON. Le code MAST associé est public et disponible sur le site internet de cette société [Inf06]. Nous avons fait une publication concernant ce modèle dans la conférence internationale FDL [GAE06b].

Le modèle du MOSFET reprend toutes les principales spécifications et caractéristiques du composant semi conducteur qu'il modélise. Nous avons choisi plus particulièrement le MOSFET de puissance canal N (BCS032N03S OptiMOS2) qui fait parti d'un catalogue de composant. Ce dispositif basse puissance (< 300V) est dédié à la conversion DC/DC haute fréquence (~300 kHz). Le modèle MAST d'INFINEON fournit un comportement dépendant de la température, basé sur la structure micro électronique du MOSFET et aussi sur le package (de type S08) via un réseau thermique, représenté sur la Figure 67.

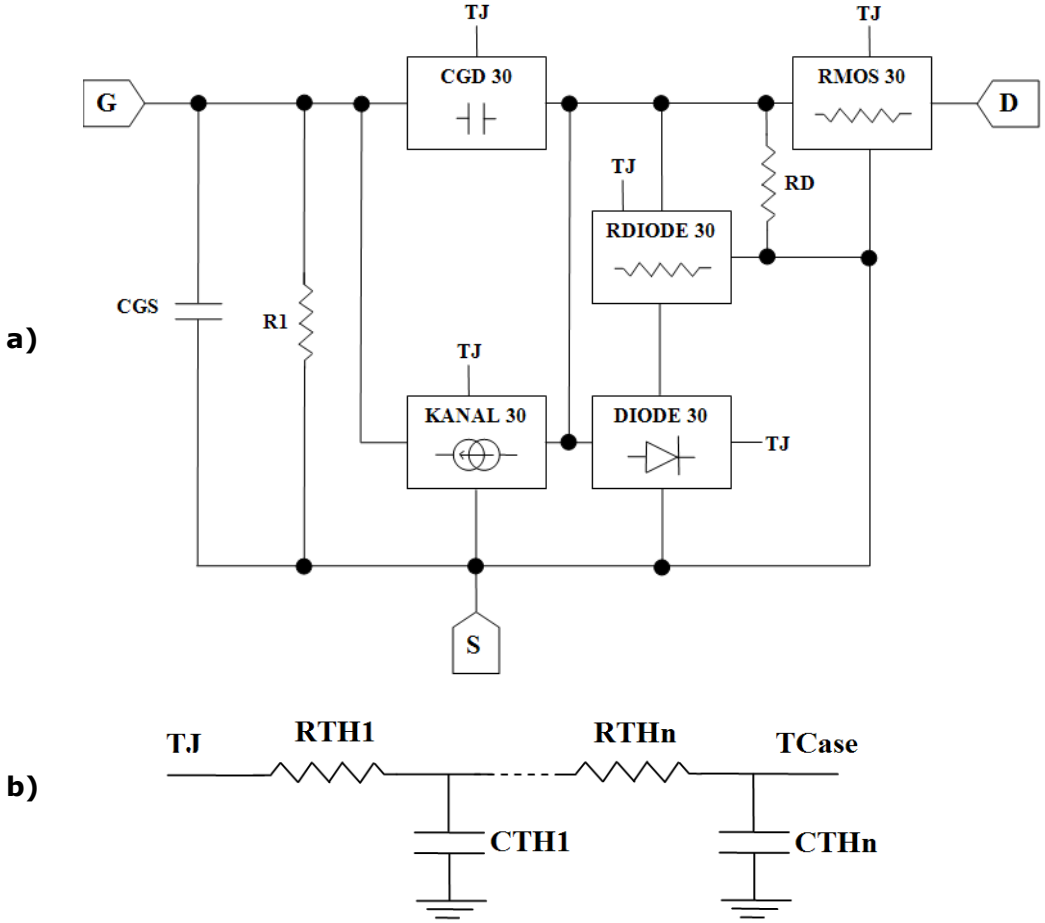


Figure 69 : a) Macro Modèle de l'OptiMOS BSC032N03S
b) Réseau thermique entre TJ (T° du cœur du composant) et TCase (T° du boitier)

Le nœud TJ permet au concepteur de visualiser la variable dépendante du temps de la température de jonction. Le port TCase doit être connecté à une source thermique afin de définir la température ambiante.

4.2.2 Méthodologie de création du modèle VHDL-AMS

De manière identique à l'exemple de transistor à hétérojonction développé dans la partie précédente, la majeure partie des sous modèles sont analogiques. Ainsi, le logiciel Paragon a pu être utilisé comme point de départ de notre processus, et des améliorations manuelles ont été ajoutées ensuite. Chaque sous modèle a été testé dans un banc d'essai qui lui était dédié, ce qui a permis de résoudre plus facilement les problèmes qui peuvent survenir au moment de l'intégration des sous modèles au sein du macro modèle. Chaque sous modèle présente la caractéristique d'être générique, ce qui permet sa réutilisation notamment dans la confection de l'ensemble du catalogue de composant INFINEON. La décomposition du modèle est représentée sur la Figure 68. On y retrouve les différents sous modèles utilisés ainsi que les macro modèles qui les instancient. Les codes VHDL-AMS de ces modèles sont disponibles dans l'Annexe 3.3 de ce mémoire.

Le macro modèle du MOSFET est testé en analyse transitoire, et en mode commutation via une source de tension qui envoie des pulses de 0 à 8 Volts, avec une période de 20µS. Le banc d'essai associé est représenté sur la Figure 69.

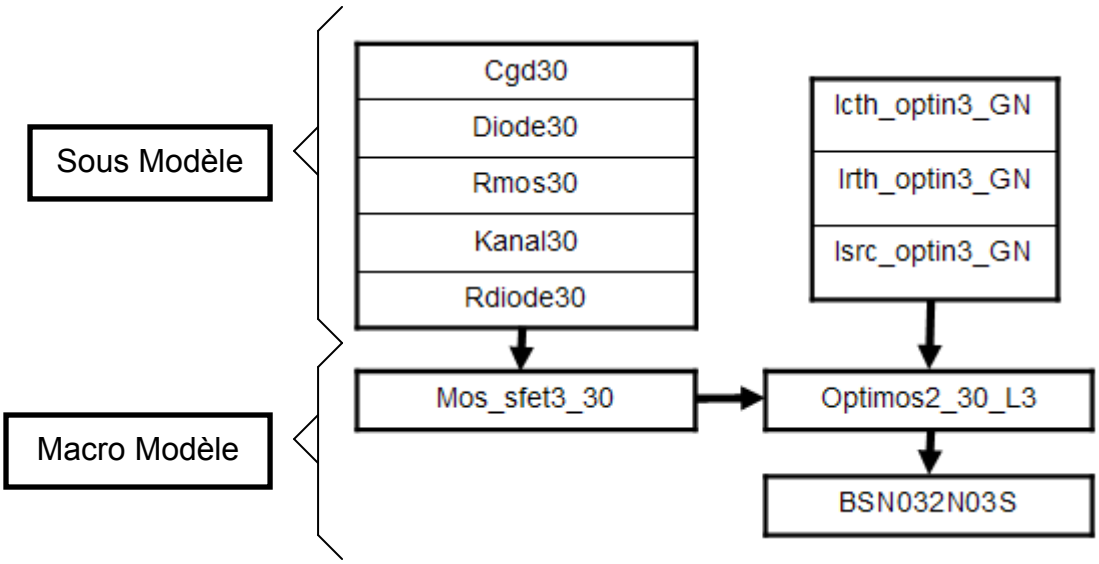


Figure 70 : Décomposition en sous modèles du modèle OptiMOS (BNS032N03S)

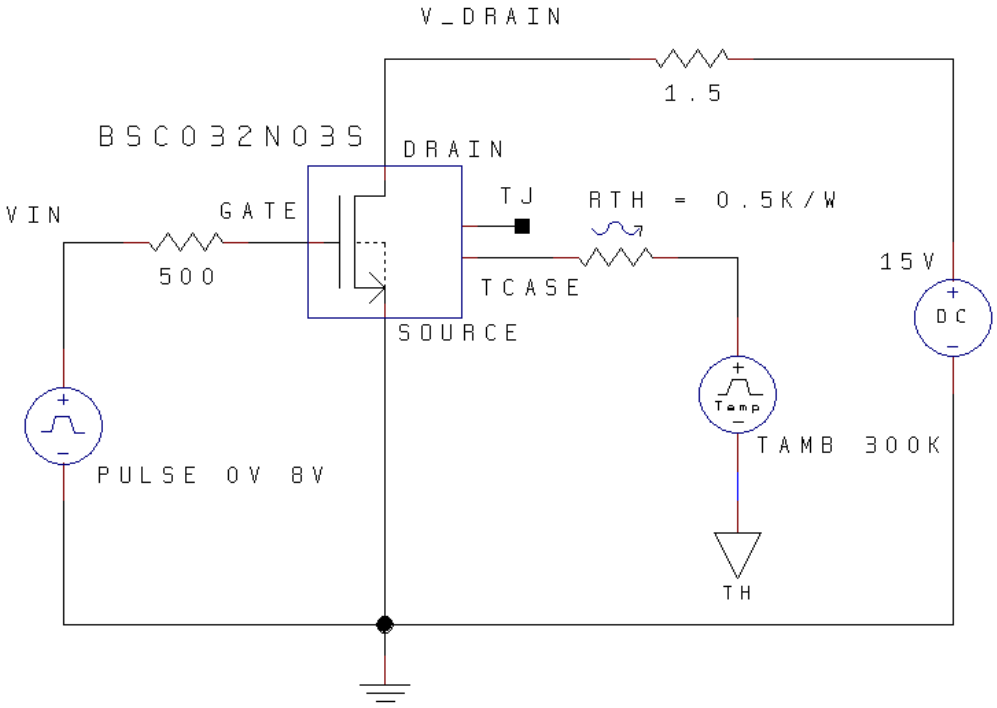
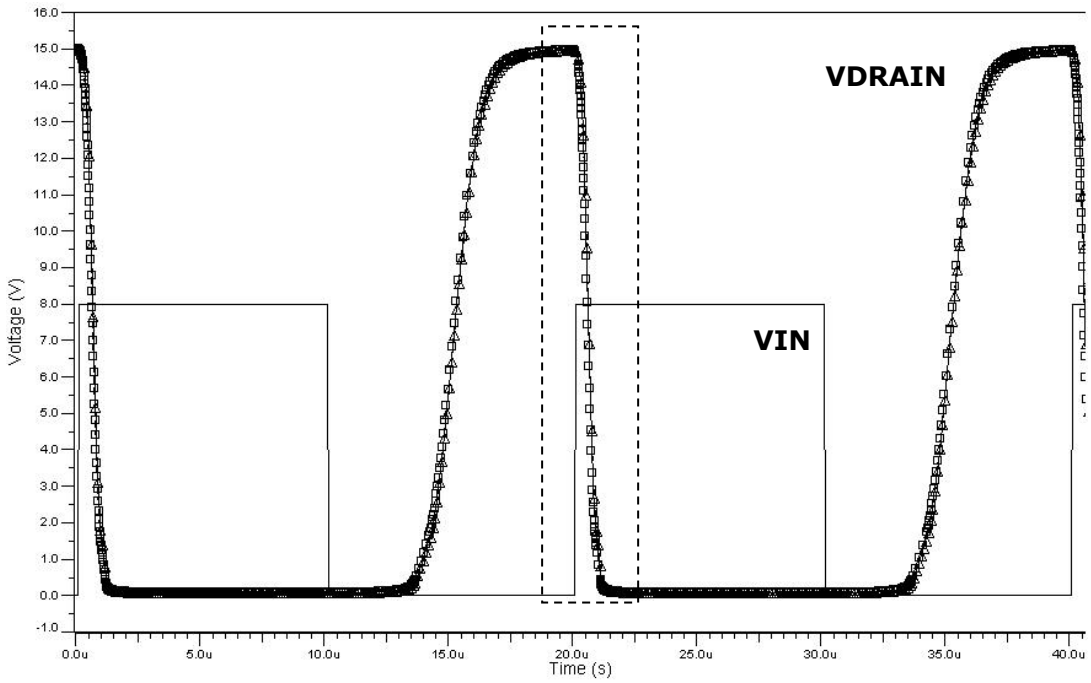


Figure 71 : Banc d'essai du BSC032NO3S en mode Commutation

4.2.3 Résultats

Les résultats pour le mode commutation sont représentés sur la Figure 70. Les courbes proviennent de l'afficheur de courbe de l'outil SystemVision [Sys50]. Les résultats de simulation des modèles MAST sous Saber [Sab] sont importés dans cet afficheur en récupérant les fichiers de points et en les rendant compatibles via un script spécifique que nous avons créé.

a)



b)

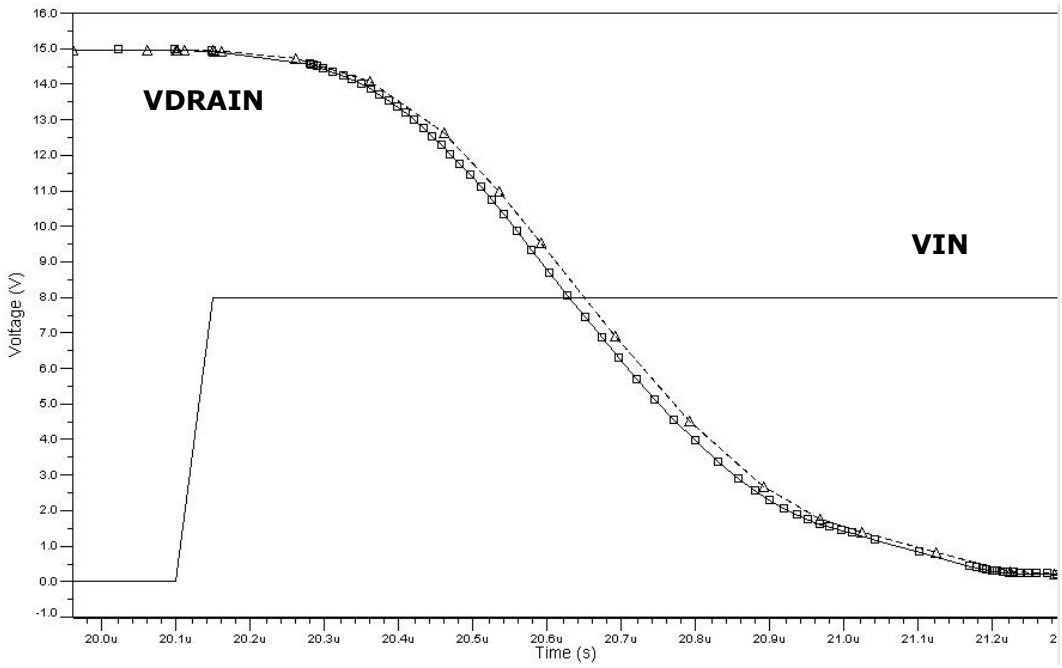


Figure 72 : L'OptiMOS a) en mode commutation
b) Zoom : Δ : Modèle MAST ; □ Modèle VHDL-AMS

Nous pouvons remarquer que les résultats entre les modèles MAST et VHDL-AMS sont similaires. Nous pouvons souligner une bonne corrélation entre le modèle source MAST et celui écrit en VHDL-AMS, avec une disparité maximale de 8%. Les réglages des simulateurs, en termes d'algorithme employé, méthode d'intégration, précision, ont un rôle très important dans les résultats de simulation et peuvent expliquer les divergences que l'on obtient.

4.3 Modèle de capteur oxygène

4.3.1 Présentation du modèle

Nous allons dans ce paragraphe étudier la création d'un modèle VHDL-AMS à partir de spécifications et de prototypes réels. Le besoin vient de l'industriel NEOSENS [Neo07] qui conçoit des capteurs d'oxygène.

Ces capteurs sont de type électrochimique et permettent la mesure de l'oxygène dissous dans l'eau. En effet, le développement de détecteurs d'espèces chimiques est un besoin important qui conditionne le bon fonctionnement des installations industrielles et la sécurité des installations en général. L'effort qui leur est consacré s'est fortement intensifié à partir de 1972 avec la découverte des IsFETs. La communauté a vu, dans cette innovation, la perspective de nombreux développements de capteurs chimiques « solides et bon marché ». Pourtant des difficultés persistent sur les questions de stabilité et de sélectivité comme tous les capteurs basés sur les propriétés d'absorption. Les capteurs électrochimiques sont des candidats intéressants car ils sont à priori sélectifs, la réaction électrolytique ne s'établissant qu'à des valeurs précises de potentiels électriques. Pour cela, ils restent des dispositifs difficiles à réaliser. La difficulté tient à la réalisation d'une membrane étanche à l'électrolyte et perméable à l'espèce à mesurer, en l'occurrence ici O2. M. Dilhan et L.Auret ont publiés une structure de capteur micro usiné sur silicium [ADE03], maintenant l'idée d'une électrolyte liquide entre deux plaques de silicium. Cette structure de base a été reprise par la start-up NEOSENS et développée industriellement.

L'objectif ici est de proposer une aide à la conception par une modélisation VHDL-AMS de ces capteurs afin d'en affiner le fonctionnement.

4.3.2 Méthodologie de création du modèle VHDL-AMS

Comme le montre la Figure 71, nous considérons le capteur constitué :

- D'une partie active I, qui va fixer la sensibilité du capteur par la surface s de l'électrode d'Au,
- D'une partie passive II, qui va fixer le temps de réponse lié à l'équilibrage des espèces au travers de la membrane.

Dans la partie active, nous pouvons écrire simplement le système d'équations sous la forme, incorporant notamment Dw, coefficient de diffusion dans la membrane et De, coefficient de diffusion dans l'électrolyte :

- Le flux de molécules O2 circulant dans la structure active au niveau de la membrane est donné par l'équation :

$$\left([O_2]_{ext} - [O_2]_{Memb}\right) \times \frac{Dw}{w} \times s = \frac{d[O_2]}{dt} \quad (2)$$

- Le flux de molécules d'O2 circulant dans l'électrolyte face à l'électrode d'Au, de surface s est donné par l'équation:

$$\left([O_2]_{Memb} - [O_2]_s\right) \times \frac{De}{e} \times s = \frac{d[O_2]}{dt} \quad (3)$$

- Le flux défini par la réactivité (0 -> 0-) de l'électrode caractérisée par la vitesse de réaction Vr est donné par l'équation :

$$[O_2]_s \times Vr \times s = \frac{d[O_2]}{dt} \quad (4)$$

Dans la partie passive, on considère que le flux d'oxygène traversant la membrane sur la surface S-s vient remplir la zone « morte » d'électrolyte. La concentration en O2 évolue dans le temps, sous la forme :

$$\left([O_2]_{ext} - [O_2]_{Memb}\right) \times (S - s) \times \frac{Dw}{w} = \frac{d[O_2]}{dt} \tag{5}$$

Pour illustrer cette première approche de modélisation, nous utilisons une représentation électrique (avec résistances et capacités) selon le schéma de la Figure 71.

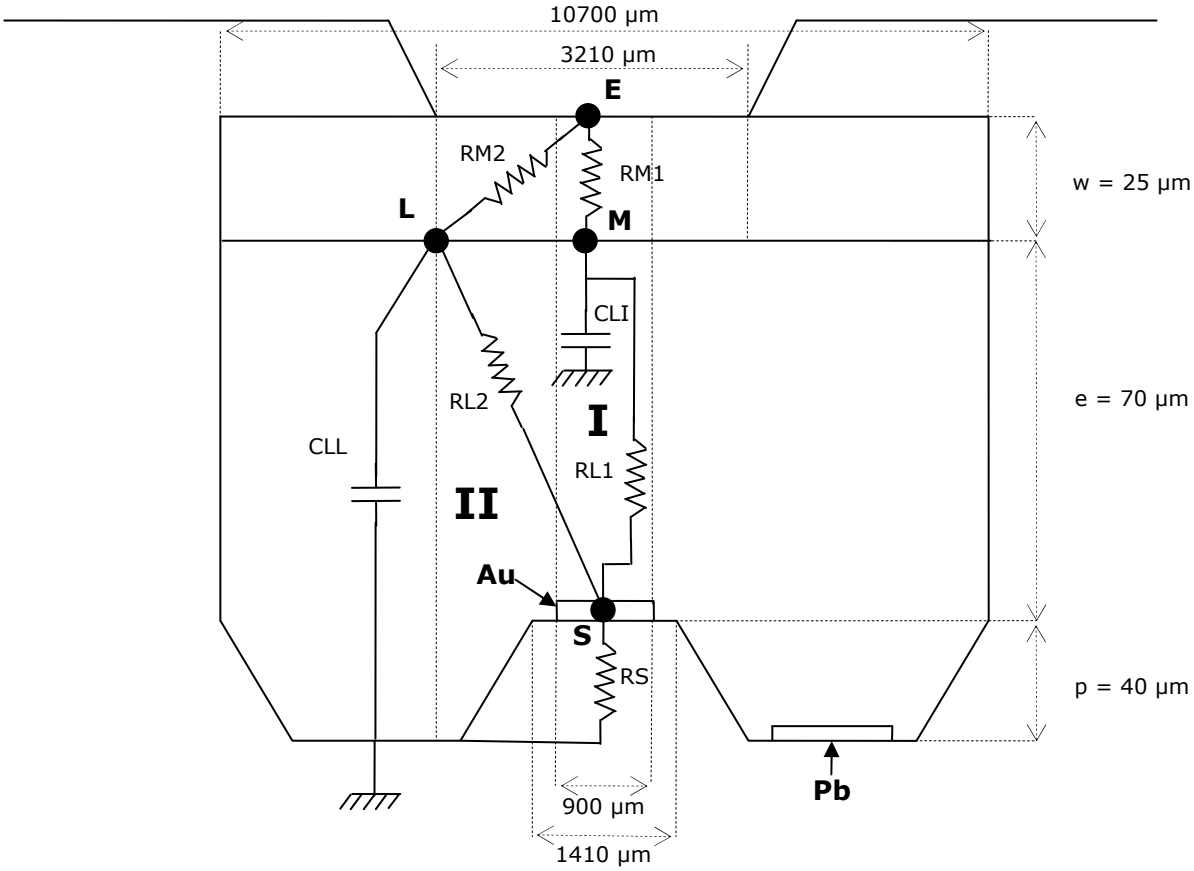


Figure 73 : Découpe avec paramètres du capteur + superposition du modèle équivalent électrique

Cette approche de premier niveau amène une analogie entre concentration et potentiel électrique. Le calcul de la valeur des différents composants électriques (résistances et capacités du modèle) s'obtient alors via les équations suivantes :

$$RM1 = \frac{1}{q \times s\_capteur} \times \frac{w}{Dw} \tag{6}$$

$$RM2 = \frac{1}{q \times (Surf\_Diff - s\_capteur)} \times \frac{w}{Dw} \tag{7}$$

$$RL1 = \frac{1}{q \times s\_capteur} \times \frac{e}{De} \tag{8}$$

$$RL2 = \frac{1}{q \times (Surf\_Diff - s\_capteur)} \times \frac{e}{De} \tag{9}$$

$$RS = \frac{1}{q \times alpha \times s\_capteur} \quad (10)$$

$$Cll = Volume\_Mort * q \quad (11)$$

$$Cli = s\_capteur \times e \times q \quad (12)$$

Cette modélisation en équivalence électrique est évidemment facilement modélisable en utilisant le langage VHDL-AMS et simulable en utilisant un outil comme SystemVision de Mentor Graphics. L'analogie électrique n'est d'ailleurs pas indispensable ici puisque le VHDL-AMS est capable de modéliser les équations physiques directement.

Les valeurs de concentrations manipulées étant trop importantes, nous avons décidé d'exprimer toutes les concentrations en ppb (part per billion : partie par milliard).

Nous connaissons un certain nombre de résultats expérimentaux réalisés sur les prototypes réels qui vont nous permettre de caractériser notre modèle. Ainsi lorsque le capteur est placé dans une eau saturée à 8240 ppb, la réponse en courant du capteur est de 500 nA. Un document Excel a été réalisé permettant de calculer le paramètre alpha en fonction notamment de la surface considérée de membrane qui diffuse de l'oxygène dans l'électrolyte (cf. Annexe 4). Une fois ces deux paramètres choisis, le modèle est capable de calculer les différentes valeurs des composants de la structure électrique données par les équations (6) à (12) (cf. Code VHDL-AMS Annexe 3.4).

4.3.3 Résultats

Nous allons simuler notre modèle VHDL-AMS de notre capteur dans le cadre d'une utilisation du capteur dans une eau saturée. Sur la Figure 72, nous représentons les résultats de mesure fait par le prototype réel du capteur. Le capteur utilisé est de type K90 ce qui signifie que l'électrode de mesure à une surface de 900µm x 900µm.

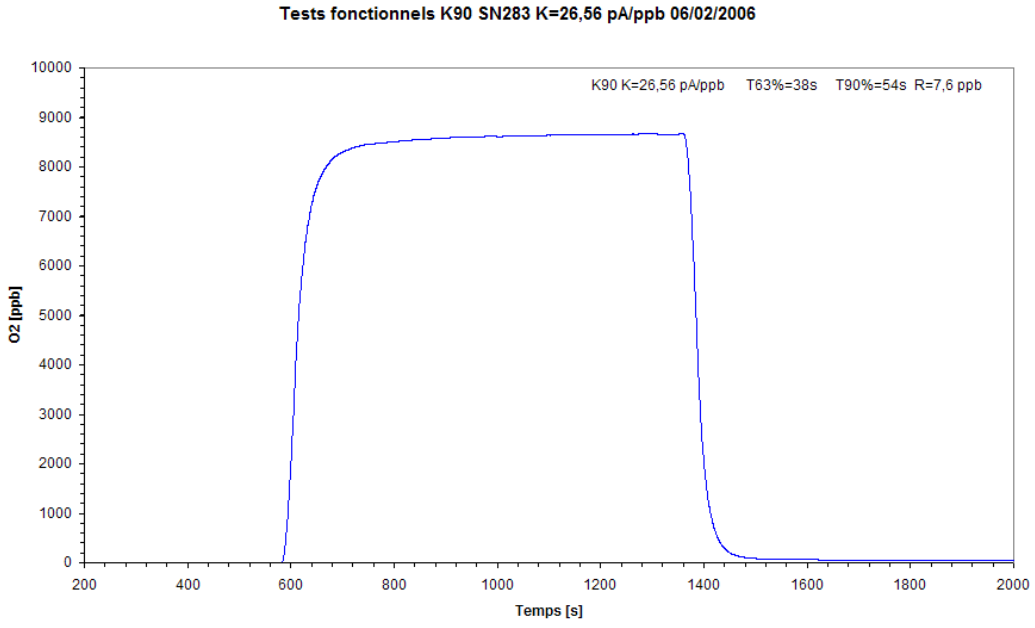


Figure 74 : Mesure de la concentration d'O2 par le capteur de type K90 en eau saturée

Comme énoncé dans le paragraphe 4.3.2, la réponse en courant du courant dans une eau saturée à 8240 ppb est de 500 nA. D'après les résultats expérimentaux donnés sur la Figure 72, nous pouvons observer un temps de montée de 54s. Nous avons mis sur la Figure 73, les résultats obtenus via le logiciel SystemVision de notre modèle VHDL-AMS. Trois variables différentes sont représentées, à savoir : la concentration externe de l'eau que l'on force à 8200 ppb, la concentration en oxygène au niveau de l'électrode (en ppb) et le courant mesuré dans la résistance Rs.

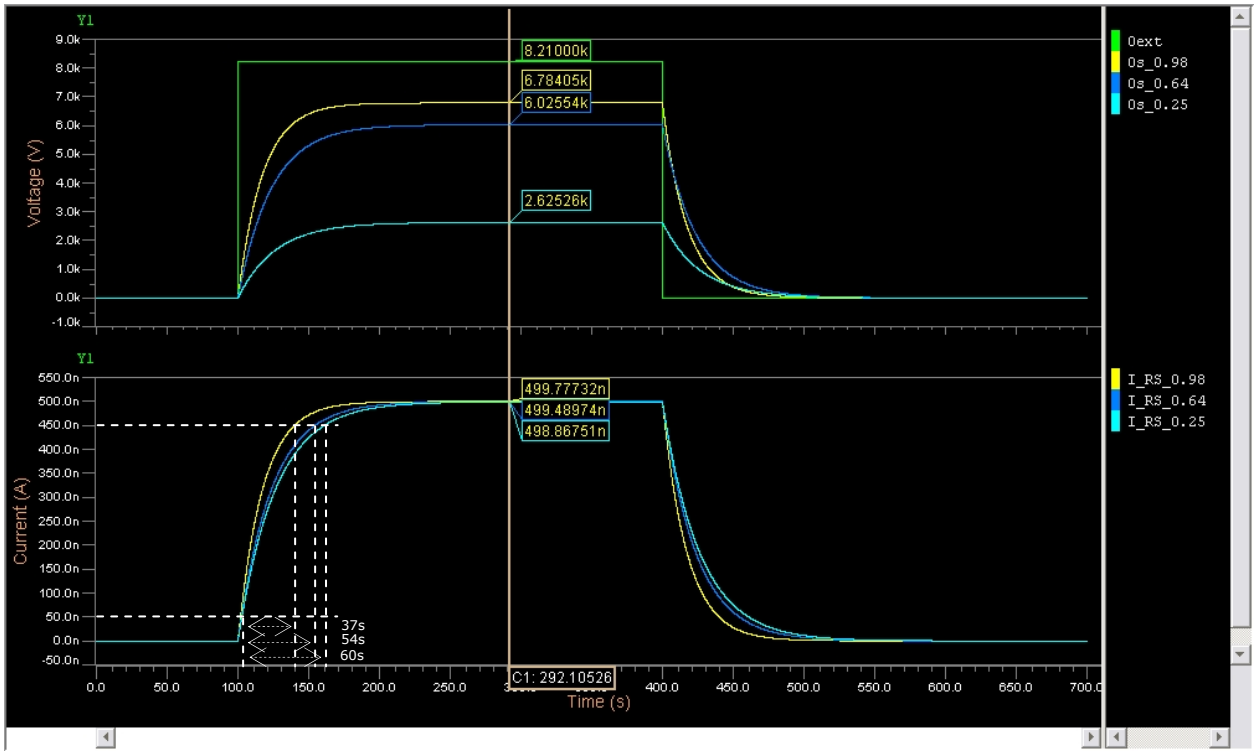


Figure 75 : Résultats de simulation du capteur oxygène : Réponse du courant à une concentration en oxygène (Oext), suivant la valeur de la surface de diffusion du capteur choisie

Certaines questions subsistent encore aujourd'hui concernant la valeur à mettre en œuvre pour la surface de la membrane qui diffuse de l'oxygène. Une première approche consisterait à ne prendre que la surface du puit (0.1 cm2) mais d'après nos calculs cela ne permettrait pas d'obtenir une diffusion d'oxygène suffisante. C'est pourquoi nous pensons que la diffusion au niveau de la membrane se fait sur une surface plus grande. Le procédé de réalisation du capteur est aussi à prendre en compte car il implique l'utilisation d'une colle entre la membrane et le support qui pourrait favoriser une augmentation de la surface de diffusion au niveau de la membrane. C'est pourquoi nous avons testé dans notre simulation des différentes valeurs pour la surface de diffusion de la membrane (cf. les différentes courbes sur la Figure 73, avec des valeurs de surface de diffusion fixées à 0.25, 0.64 et 0.98 cm2). Ces résultats montrent que la concentration au niveau de l'électrode va être de plus en plus faible si on diminue la surface de diffusion. Le temps de montée va aussi être influencé par la modification de ce paramètre. Nous observons d'ailleurs que pour être conforme au temps de montée obtenu sur le prototype réel (54s), la valeur de la surface de diffusion de la membrane devra se située entre 0.25 cm2 (temps de montée = 60s) et la surface totale de la membrane 0.98 cm2 (temps de montée = 37s). Par simulation, nous avons trouvé qu'il faut une valeur de surface de diffusion de 0.64 cm2 pour obtenir un temps de montée de 54s.

La version du modèle présentée dans ce mémoire n'est encore qu'une version préliminaire qui doit être encore améliorée avec l'intégration d'autres effets notamment au niveau de la membrane, mais aussi en affinant les différents paramètres suivant les résultats expérimentaux obtenus sur les prototypes réels. Cette première pose néanmoins les bases du modèle dont l'intérêt est de pouvoir le relier à une interprétation physique et de l'exploiter à des fins d'optimisation. Ce modèle est particulièrement adapté pour étudier la sensibilité et la réponse temporelle du capteur.

4.4 Modèle de Fil et de Fusible

4.4.1 Présentation du modèle

Dans le cadre d'un projet industriel avec Renault, des modèles VHDL-AMS de fil et de fusible ont été créés pour être ensuite utilisés dans des modélisations de systèmes plus complexes comme un modèle d'allume cigare. Ici le point de départ est les spécifications des modèles à réaliser. Une modélisation sous Saber avec des modèles MAST a été auparavant réalisée par les ingénieurs de chez Renault, mais nous n'y avons pas accès. Seuls des résultats de simulation sont donnés.

La structure du modèle de fusible est donnée sur la Figure 74. Les modèles de fil et de fusible doivent contenir un générateur thermique qui permet de fixer la valeur de la température ambiante. Un circuit RC thermique permet de fixer la réponse thermique du modèle. Lorsque le courant traverse la résistance électrique du modèle, la puissance dissipée est injectée dans le circuit thermique et permet la variation de la température. Pour le modèle de fil, la valeur de la température est relevée au point Temp_fil. Le fonctionnement intrinsèque du modèle de fusible est en tout point identique à celui du fil. La seule différence est l'introduction du switch qui permet d'ouvrir le circuit lorsque le fusible est détruit (l'ouverture du circuit est optionnelle ; par défaut, il reste fermé ce qui permet de détecter d'autres problèmes sur le circuit).

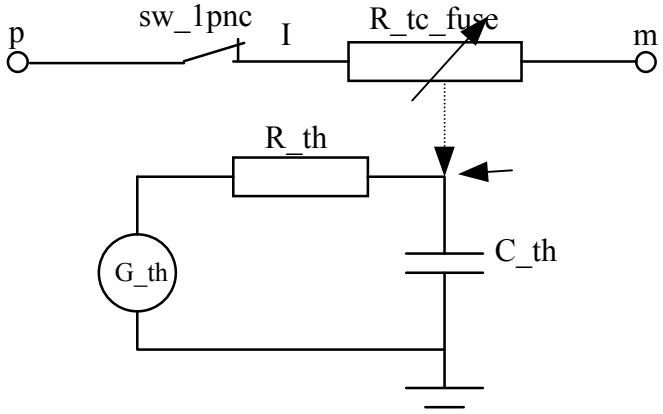


Figure 76 : Structure du modèle de Fusible

Le cahier des charges indique pour un fusible donné, son temps de fusion en fonction du courant qui le traverse. Sur le Tableau 8, nous avons mis ces données concernant un fusible particulier : fusible MINI 15 A CDC Renault.

Courant (A)	20.25	30	52.5
Temps de fusion(s)	1200	2	0.2

Tableau 8 : Spécifications Fusible Temps Fusion / Courant

A partir de ces valeurs du cahier des charges, les paramètres caractéristiques du modèle sont calculés à partir du logiciel MATLAB (cf. Figure 75).

```
category     fuse
supplier     renault
class        MINIFUSE
calibre15
dispersion   maximum
r0           4.5e-3
tmelt        419
vI           20.25 30 52.5
vT           1200 2 0.2
vTsimu 1200.0004      2.0000007            0.2
J      8.6179e-14
R1           72.7012
C1           0.0050205
R2           26.1588
C2           0.0056003
tau          484.5494
```

Figure 77 : Paramètres caractéristiques du fusible MINI 15 A CDC Renault (calculé par Matlab)

Ce travail préliminaire à la modélisation a été réalisé par les ingénieurs de Renault et a été stocké dans une base de données regroupant un grand nombre de fusible et fil différents. Cette base de donnée est accessible par les paramètres génériques de chaque modèle afin qu'il soit caractérisé.

4.4.2 Méthodologie de création du modèle VHDL-AMS

Le modèle VHDL-AMS est donc créé en fonction des spécifications. La structure des modèles de fil et de fusible est maintenue (cf. Figure 74). Certains modèles comme une résistance thermique, la source thermique pour la température ambiante, ou la capacité thermique existent dans les bibliothèques de modèles du logiciel SystemVision. Ils sont donc réutilisés. Le modèle d'une résistance ayant un comportement électrothermique existe aussi dans ces bibliothèques de modèles. Néanmoins il nécessite d'être modifié afin de correspondre exactement aux exigences des spécifications concernant le comportement du fusible notamment après que celui-ci ait fondu. En effet une des préoccupations de Renault est, outre de savoir à quel moment le fusible fond, quelles en sont alors les conséquences. Un travail approfondi, réalisé en collaboration avec les ingénieurs de chez Renault sur les spécifications, a permis de bien délimiter le comportement que devait avoir le modèle du fusible. Le modèle a donc été modifié de manière à ce que le fusible voit sa résistance figée à la valeur qu'elle avait au moment de la fusion, conformément aux spécifications de Renault. Le code du modèle de fusible est donné en Annexe 3.5 avec le modèle VHDL-AMS de la structure du fusible fuse_dim_V1.vhd en Annexe 3.5.1, et la résistance électrothermique du fusible en Annexe 3.5.2).

Une autre contrainte à respecter, est que le modèle générique doit être capable d'aller chercher dans une base de données les valeurs caractéristiques du fusible suivant ses références. Cela est réalisable en VHDL-AMS grâce à ses aptitudes à manipuler des fichiers via le package *TextIO*.

4.4.3 Résultats

Dans les spécifications, plusieurs bancs d'essais appelés périmètres sont définis. Nous nous sommes intéressés tout d'abord au périmètre 0 qui permet de valider dans un premier temps les modèles de fil et de fusible. Ce périmètre est représenté sur la

schématique de la Figure 76. Nous avons dans le circuit une source de tension qui applique une tension continue de 12 V. Un modèle de fil et de fusible sont utilisés. Leurs références sont définies dans les paramètres génériques des modèles afin que le modèle puisse aller chercher les caractéristiques physiques correspondantes dans la base de données. Une résistance de charge est utilisée afin de pouvoir faire varier l'intensité qui traverse le fil et le fusible.

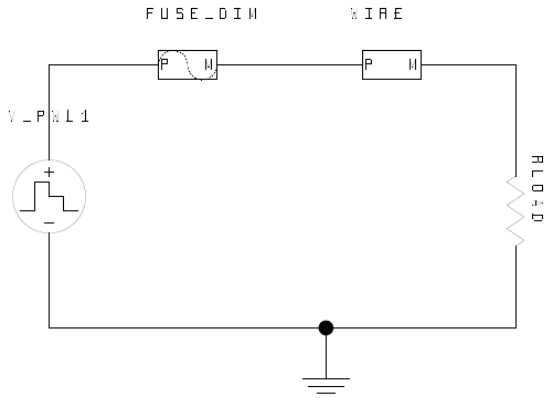


Figure 78 : Banc d'Essai : Périmètre 0

La validation de ces deux modèles se fait par comparaison de résultats au niveau des temps de fusion suivant le courant qui est appliqué. Aucune comparaison de courbes n'est effectuée ici. Les résultats de la simulation des modèle MAST sont donnés par le client. Ils sont aussi à comparer avec les données des spécifications données dans le Tableau 8. Les résultats de simulation du périmètre 0 pour les modèles MAST et VHDL-AMS sont donnés dans le Tableau 9.

Résistance (mOhms)	**Temps fusion(s)**		**I(fusion)(A)**	
	MAST	**VHDL-AMS**	MAST	**VHDL-AMS**
225	0.208	**0.205**	52.287	**52.28**
250	0.270	**0.268**	47.151	**47.15**
300	0.439	**0.453**	39.408	**39.4**
350	0.719	**0.805**	33.850	**33.85**
500	317.7	**318.2**	23.661	**23.66**
582	1196.5	**1203.2**	20.279	**20.28**

Tableau 9 : Résultats Simulation du Périmètre 0

Les écarts sont de l'ordre de 1%, ce qui nous permet d'affirmer que les modèles sont équivalents et conformes aux spécifications. Les modèles créés vont donc pouvoir être réutilisés dans un autre périmètre. Seul le périmètre 0 est explicité dans ce mémoire, mais le périmètre 1 (allume-cigare) a aussi été réalisé après validation des sous modèles de fusible et de fil. La schématique de ce périmètre 1 comprenant l'allume-cigare et les sous modèles de fil et de fusible est donnée en Annexe 5).

4.5 Problèmes Posés par la Validation

Dans les exemples de création de modèle VHDL-AMS que nous avons traités dans les paragraphes précédents, l'objectif est de réaliser des modèles VHDL-AMS en conformité avec les modèles et les spécifications de départ. Cette conformité s'applique sur la forme de ce modèle (interface, généricité,…), mais aussi sur son comportement dans son domaine de validité.

Concernant la traduction de modèles écrits dans un langage de description de matériel différent, nous nous sommes appliqués à reprendre fidèlement les choix faits lors de la création du modèle source concernant notamment la structure et le niveau d'abstraction du modèle. Le modèle source n'est pas amélioré vis-à-vis de ces performances mais retranscrit fidèlement suivant un portage que l'on peut qualifier d'identification (cf. [Her06]). L'hypothèse de départ est que le modèle source a été validé au point de vue de son comportement dans un certain domaine de validité donné. Notre modèle VHDL-AMS créé doit donc se conformer à ces mêmes contraintes.

Les moyens que l'on dispose pour valider notre modèle créé sont liés aux informations disponibles qui accompagnent le modèle source. Ainsi les données habituellement disponibles sont :

- Des spécifications détaillées, comprenant des contraintes à respecter en termes de temps, de seuil,… mais aussi des gabarits d'erreurs dans lesquels certaines variables doivent se conformer,

- Des bancs d'essais modélisant l'environnement d'un cas d'utilisation du modèle créé, afin d'en valider le bon fonctionnement.

Dans nos exemples, nous avons utilisé une validation des modèles créés se basant sur des résultats de simulation. Nous sommes partis des bancs d'essais qui avaient permis de valider le modèle source et nous l'avons ensuite retranscrit en VHDL-AMS. Les bancs d'essai ont été créés dans l'optique de valider une partie du comportement du modèle dans un cas d'utilisation. Il peut devenir très fastidieux de valider un à un l'ensemble de ces cas, mais l'on peut définir un plan d'expérience nous permettant de couvrir un domaine d'utilisation complet. Ce travail va être lié à celui de la création des vecteurs de test qui vont nous permettre de mettre notre modèle en situation pour en valider le comportement.

Nos résultats et nos interprétations explicités dans les paragraphes précédents de ce chapitre ne sont en fait qu'une validation que l'on peut qualifier de premier niveau [Her06]. Ce travail n'est pas assez rigoureux pour une application industrielle et nécessite l'application d'une méthodologie de validation plus fine. Lors de notre étape de validation de notre processus de conception de modèle, nous avons comparé les résultats de simulation au sein d'un même outil de visualisation de courbes et calculé l'erreur maximale. Pour se faire, nous avons importé les données de simulation du logiciel simulant le modèle source, dans le logiciel simulant le modèle cible après les avoir traitées pour les rendre manipulables par l'outil de visualisation de courbes de ce logiciel (script de modifications de syntaxe de la base de donnée).

Cette technique de validation implique néanmoins de prendre un certain nombre de précautions quant à sa mise en place et à l'interprétation de ces résultats dans le cadre de modèles analogiques. Les résultats de simulation ne sont en fait que des couples (temps, valeurs) associables à des points sur un graphique. Leur nombre et l'écart temporel les séparant sont fortement conditionnés par les paramètres de la simulation et plus particulièrement par le pas de temps. Celui-ci est bien souvent de type variable laissant au simulateur le choix d'adapter le pas de temps suivant la précision que l'on souhaite avoir. Cette option rend donc les comparaisons directes des bases de données

de simulation non réalisables de manière rigoureuse. Il convient en fait de ré échantillonner les courbes pour pouvoir en faire la comparaison à des temps précis. Cela implique donc un certain travail préliminaire (d'interpolation par exemple). Cela est essentiellement valable pour les validations de modèles analogiques. Cela n'est pas nécessaire dans le cadre de modèles numériques manipulant des données discrètes dont la simulation va générer des évènements à un temps t donné.

L'étape de comparaison des résultats va pouvoir ensuite impliquer plusieurs types de tests concernant le calcul de l'erreur faite par les résultats de la simulation du modèle cible vis-à-vis du modèle source (erreur moyenne, maximale, quadratique). L'erreur acceptable ou gabarit devra être définie dans les spécifications. Il convient d'ailleurs de prendre aussi en compte l'erreur faite par le modèle source vis-à-vis du système réel modélisé. Cette erreur doit être mise en rapport avec l'erreur faite par le modèle cible. Bien souvent cette erreur initiale n'est malheureusement pas toujours référencée. L'ensemble des recommandations précédentes nous amènent à conclure que les questions de validation doivent être abordées très tôt dans le processus de la transformation, afin que les exigences envers le modèle cible y soient bien définies et que les attentes vis-à-vis du modèle cible soient atteintes.

Nous avons traité dans ce paragraphe de la validation du modèle créé. Il convient de souligner la différence qu'il faut faire entre la validation et ce que l'on appelle la vérification. La validation va nous permettre de s'assurer que le modèle que l'on a créé est celui que l'on voulait et qu'il répond aux contraintes de comportement et de performances définies dans les spécifications. La vérification va plutôt s'assurer que le modèle que l'on a créé est bien fait, c'est-à-dire qu'il est conforme à son meta modèle et que les étapes de création ont été bien suivies. Dans le cadre d'un portage mêlant une grande part d'expertise, le processus est peu adapté à la vérification. La méthodologie de transformation de modèle basée sur les meta modèles est quant à elle beaucoup plus adaptée à cela, avec notamment une définition de règles de transformations qui assure une bonne traduction du modèle source vers un modèle cible conforme à son meta modèle.

4.6 Conclusion

Dans ce quatrième chapitre, nous avons illustré les aptitudes du langage normalisé VHDL-AMS à modéliser des systèmes hétérogènes très divers. Nous avons exploré plusieurs modèles mettant en jeu des domaines physiques différents, plusieurs niveaux d'abstraction, différents caractères de généricité et de réutilisabilité, un mode d'utilisation descriptif ou prédictif. Cela a aussi été l'opportunité de préciser les informations fondamentales qui doivent accompagner un modèle au delà de son code. Nous insistons sur l'idée de domaine de validité : description du banc d'essai de validation et définition des paramètres de la compilation et de la simulation.

Pour nos exemples, la démarche de validation a été basée sur une comparaison des résultats de simulation avec un gabarit d'erreur, ce que l'on peut qualifier de validation de premier niveau. Cependant ce type de validation n'est, en l'état, pas assez rigoureux. Un travail d'approfondissement doit être fait ici, notamment sur la méthodologie de création d'un banc d'essai permettant de valider le comportement du modèle dans un domaine d'utilisation bien défini, mais aussi sur des alternatives possibles à une validation des modèles par comparaison des résultats de simulation.

Au cours de ce quatrième chapitre, pour créer les différents modèles, nous avons principalement appliqué la méthode requérant une grande part d'expertise et d'outils comme Paragon, pour pouvoir répondre à des contraintes principalement industrielles, avec un besoin de plusieurs modèles à créer dans un délai court au départ de la thèse. Cela nous a permis néanmoins, avec les nombreux exemples de modélisation VHDL-AMS explicité dans ce quatrième chapitre (transistor bipolaire à hétérojonction, MOSFET de puissance, carte d'alimentation), de voir les limites d'une telle méthodologie de transformation en termes de temps de migration et d'automatisation. L'intégration de nouveaux outils et concepts pouvant améliorer l'automatisation du processus et la rapidité de transformation va devenir nécessaire pour répondre aux contraintes industrielles (temps, nombre de personnes impliquées,…) pour la migration d'une bibliothèque complète de modèles par exemple.

Sur ces exemples, il n'a pas été appliqué la méthodologie liée aux concepts de meta modélisation et de transformation de modèles avec définition des règles au niveau meta modèle. Cela aurait demandé un trop long travail préliminaire, notamment pour l'écriture de toutes les règles de transformations. Néanmoins, nous pouvons considérer que les nouvelles approches utilisées dans l'industrie concernant le stockage de données relatives aux modèles, sous la forme de fichier XML avec définition des grammaires sous forme de DTD (ou XSD), représente déjà un pas vers les concepts de meta modélisation. Cela peut être un bon point de départ pour le développement d'une méthodologie de transformation liée à ces concepts.

CONCLUSION GENERALE

Notre travail se fixait pour objectif de contribuer à la mise en place de méthodes et d'outils pour la conception des systèmes hétérogènes, basée sur un prototypage virtuel du système, c'est-à-dire d'un modèle simulable du système : c'est une option qui s'impose progressivement comme un préalable au prototypage réel, qui permet notamment la réduction de coût par la découverte d'erreurs suffisamment tôt dans le processus de conception.

Cette étape du prototypage virtuel s'inscrit dans un processus complexe de conception, qui part du cahier des charges du système pour aboutir aux étapes finales du prototypage et de la fabrication. Ce processus met en œuvre de nombreuses autres étapes qu'il convient de coordonner très précisément. Cela nous a conduits à promouvoir la norme EIA-632 de l'ingénierie système pour définir notre processus générique de conception : il est intéressant de constater que les travaux parallèles de S. Rochet montrent qu'une formalisation de l'ensemble des processus présentés dans l'EIA-632 est possible en langage SPEM [Roc06]. Nous montrons que le langage VHDL-AMS s'inscrit dans ce processus.

Ce processus générique de conception, partant du cahier des charges, comporte trois étapes principales : la formalisation des exigences, le modèle logique et le prototypage virtuel, qui nous intéresse plus spécialement. Pour les deux premières étapes, nous nous référons aux travaux de J.C. Hamon [Ham05] qui a proposé une plateforme HiLeS Designer, permettant la représentation d'un système via l'utilisation du formalisme HiLeS mettant en jeu notamment un ensemble de blocs structurels et fonctionnels ainsi que des Réseaux de Petri, pour représenter le modèle logique du système.

Notre idée principale, dans cette thèse, est de proposer le langage de modélisation normalisé VHDL-AMS (IEEE 1076-1999) comme langage cible du prototypage virtuel. La mise en œuvre de ce concept suppose d'identifier les chemins pour :

- Ecrire directement des modèles en VHDL-AMS,
- Transformer la représentation logique HiLeS en langage VHDL-AMS,
- Faire migrer des modèles issus de différentes disciplines et écrits en langages spécifiques vers VHDL-AMS, ou bien de les faire travailler ensemble par des pratiques de cosimulation.

En l'état actuel des connaissances, notre démarche générale conduit aux notions d'interopérabilité et de transformation de modèles : le deuxième chapitre a donc été largement consacré à présenter l'émergence de ce concept nouveau d'une ingénierie guidée par les modèles : une de nos contributions est d'avoir analysé ce concept, de l'avoir expérimenté et d'en tirer des leçons pour l'avenir.

Dans **le premier chapitre** de ce mémoire, nous avons replacé notre objectif de création de modèle de systèmes hétérogènes au sein d'une méthode générale de conception que nous avons relié à une ingénierie guidée par les modèles définie par l'OMG (Object Management Group) [OMG], sous la forme du MDA (Model Driven Achitecture) [MDA]. L'ingénierie système qui supporte notre processus général a ses principes issus de la norme EIA-632 [Eia98]. Nous avons, au cours de ce premier chapitre, explicité cette norme. Elle décrit deux niveux de conception : un niveau logique et un niveau physique. Nous situons notre travail au niveau de la création de la solution physique que nous identifions au concept de prototypage virtuel.

L'analyse des différents types de modèles que l'on peut rencontrer nous a permis de mettre en avant les qualités nécessaires à un langage de modélisation pour être le support de la solution physique que l'on cible. Nous nous sommes attardés sur l'importance :

- D'avoir une norme pour permettre la réutilisation des modèles créés,
- De permettre une modélisation du caractère hétérogène et multi-abstraction des systèmes,
- D'être facilement interfaçable avec la représentation logique et de pouvoir être un langage cible de transformation de modèles d'origines diverses. Dans l'optique du choix d'un langage de modélisation pour notre solution physique, nous avons présenté dans ce chapitre une comparaison de plusieurs langages. Il en est ressorti que la norme IEEE 1076.1-1999 VHDL-AMS était un bon candidat [Her02].

Ce choix de langage de modélisation nous a amené à étudier dans le **deuxième chapitre**, les différentes techniques de transformation de modèle possibles pour converger vers ce langage. Cela nous a permis de définir les qualités essentielles que devait posséder la démarche de transformation, qui permettait de mieux qualifier notre solution physique, telle que celle de permettre la traçabilité des exigences. Il ressort, tout de même, que, suivant les cas, les types de transformation seront différents. Ainsi, pour le passage de la solution logique vers la solution physique, la transformation sera de type vertical exogène, alors que pour la transformation d'un modèle écrit dans un langage de description de matériel quelconque, nous aurons une transformation de type horizontal exogène n'impliquant pas d'enrichissement [Top06].

Il ressort de notre étude, deux techniques différentes de transformation qui possèdent les qualités pour répondre à nos attentes.

La première technique se base sur un langage pivot. Nous avons alors développé l'exemple du logiciel Paragon [PAR04] qui utilise une base de donnée XML (eXtensible Markup Language) pour stocker les données relatives au modèle.

Une deuxième technique de transformation a aussi été évoquée et se base, quant à elle, sur le concept de meta modélisation et de définition des règles de transformations au niveau meta modèle. Ces concepts ont été rappelés dans ce chapitre et développé : en particulier, nous avons développé des meta modèles que nous avons appliqués aux formalismes supports de nos solutions logiques et physiques, à savoir respectivement HiLeS et VHDL-AMS. La méthodologie de création de meta modèle des langages de programmation a été explicitée dans ce chapitre, d'après les travaux de [Seb06][Mal02] : Les modèles créés par la suite doivent être conforme à ces meta modèles.

Le **troisième chapitre** nous a permis d'appliquer ces techniques de transformation pour la création de modèles VHDL-AMS. Ainsi, nous avons tout d'abord mis en place une première méthodologie qui met en jeu une grande part d'expertise de la part du concepteur dans les langages source et cible. Cette méthodologie s'appuie notamment sur certains outils comme Paragon qui utilise un langage pivot. Le défaut d'automatisation de la méthode nous a amené à explorer d'autres concepts de transformation, en particulier ceux basés sur la meta modélisation. Nous avons appliqué ce concept à la transformation de modèles écrits dans le langage de description de matériel MAST [Sab] vers des modèles écrits VHDL-AMS. Un meta modèle du formalisme MAST a été créé. Le meta modèle du formalisme VHDL-AMS créé dans le deuxième chapitre, a été modifié afin de permettre l'écriture de règles de transformations le plus simplement possible. Un exemple a été réalisé en langage ATL (ATLAS Transformation Language [ATL]) sur un modèle analogique simple.

Le passage de la solution logique HiLeS vers la solution physique VHDL-AMS a été traité et démontré faisable par différents travaux réalisés au LAAS-CNRS [Gui03][Alb05][Mau05][Ham05]. Une illustration du choix de VHDL-AMS comme support de la solution physique d'un système hétérogène a été réalisée par la modélisation d'un système de mise à feu sécurisé d'une charge pyrotechnique d'après des travaux antérieurs de P. Pennarun [Pen06] et C. Rossi [Ros97].

Dans cette optique de démonstration, le **quatrième chapitre** a été l'occasion d'appliquer les méthodologies évoquées dans les chapitres précédents pour la création de modèle VHDL-AMS, présentant comme point de départ des spécifications ou des modèles sources déjà créés. Cinq exemples concrets de modélisation sont explicités : Transistor Bipolaire à Hétérojonction, MOSFET de puissance, Capteur Oxygène, Fusible, et Carte d'Alimentation. L'intérêt de ces illustrations est de poser les questions liées à la validation. La technique de validation explorée est basée sur des comparaisons de résultats après définition d'un banc d'essai et d'une simulation du modèle dans un domaine d'utilisation bien défini. Cette méthodologie fait ressortir de nombreuses précautions à prendre. Il demeure néanmoins qu'un travail plus approfondi devra être réalisé pour améliorer cette étape.

En perspective de ce travail, l'évidence est que la construction d'un prototype virtuel d'un système hétérogène va rester un enjeu important et difficile. Deux stratégies doivent s'associer :

- Celle de faire entrer toutes les descriptions en conformité avec une seule « meta meta modélisation » et de fonder, sur cette cohérence, l'interopérabilité des outils multiples dont on peut avoir besoin. Dans le cas d'une plateforme comme ECLIPSE [Ecl], un meta meta modèle proche du MOF défini par l'OMG est proposé (Ecore), et permet l'intégration d'éditeurs de modèle basés sur le meta modèle de chaque formalisme. Ainsi les meta modèles des formalismes HiLeS et VHDL-AMS, comme ceux proposés dans le deuxième chapitre de ce mémoire, peuvent être utilisés dans cette optique. Outre la conformité des modèles créés, on peut mettre en place des règles de transformation entre meta modèles afin de converger vers la solution que l'on recherche.

- Celle de laisser chaque description dans son modèle disciplinaire et développer des grands canaux de communication au niveau des données au cours des simulations : c'est le principe de la cosimulation que l'on retrouve dans plusieurs outils comme COSIMATE [Cos06], qui permettent de faire communiquer des outils de simulation mettant en jeu des modèles d'origines différentes. Le temps et les ressources liés à la mise en place d'une telle possibilité doivent être comparés au cas par cas au temps qu'il serait nécessaire à la transformation des différents modèles dans un formalisme commun. De nombreux outils capables de simuler des modèles VHDL-AMS proposent ce genre d'alternative. Néanmoins, des systèmes modélisés et simulés de cette façon sont optimisés pour un certain cas d'utilisation et de combinaison de modèles et d'outils. Ils sont alors difficilement ré-utilisables dans un autre contexte.

De manière globale, nous pensons qu'il faut expérimenter encore davantage avec l'existant sur des exemples de plus en plus complexes pour mieux définir les problématiques de la simulation des grands systèmes complexes. Les développements de modèles VHDL-AMS sont de plus en plus nombreux, et l'on voit de plus en plus d'industriels lancer des projets concrets, conscients des bénéfices qu'apporte la conception préalable d'un prototype virtuel. Les contraintes matérielles en terme de temps de simulation et de ressource CPU limitent encore à l'heure actuelle la finesse globale de la modélisation d'un système complexe complet que l'on peut simuler. La démarche la plus riche se situe sûrement dans la simulation des systèmes à partir de modèles de différents niveaux, seule voie pour :

- Avoir des simulations globales
- Réaliser des « zooms », dans le contexte de fonctionnement et d'utilisation.

GLOSSAIRE

A-FSM	Analog Finite State Machine
AM3	ATLAS MegaModel Management Tool
AMMA	ATLAS Model Management Architecture
ASIC	Application Specific Integrated Circuit
ATL	ATLAS Transformation Language
ATP	ATLAS Technical Projectors
BNF	Backus-Naur Form
CIM	Computation Independent Model
CONPAR	Langage pivot pour transformation : Réseau de Petri vers VHDL
DSL	Domain Specific Language
DTD	Document Type Definition
EIA-632	Electronic Industry Alliance 632
FDL	Forum on specification and Design Language
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GMT	Generative Modelling Technologies
HDL	Hardware Description Language
HiLeS	High Level Specification
IEEE	Institute of Electrical and Electronics Engineer
IP	Intellectual Property
ISFET	Ion Sensitive Field Effect Transistor
KM3	Kernel Meta Meta Model
LAAS-CNRS	Laboratoire d'Analyse et d'Architecture des Systèmes – Centre National de la Recherche Scientifique
M2M	Model to Model
MDA	Model Driven Architecture
MDE	Model Driven Engineering

MIS	Microsystèmes et Intégration des Systèmes
MIXDES	MIXed DESign of Integrated Circuits and Systems
MOF	Meta-Object Facility
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
OCL	Object Constraint Language
OMG	Object Management Group
PDM	Platform Dependent Model
PIM	Platform Independent Model
PN	Petri Net
PSM	Plateform Specific Model
PWM	Pulse-Width Modulator
QVT	Query/View/Transformation
SPEM	Software Process Engineering Meta model
STANAG	STANdarzation AGreement
SVX	Technologie de cosimulation (MENTOR GRAPHICS)
SysML	Systems Modelling Language
TBH	Transistor Bipolaire à Hétérojonction
UML	Unified Modelling Language
UMTS	Universal Mobile Telecommunications System
VHDL-AMS	VHSIC Hardware Description Language – Analog Mixed Signal
VHSIC	Very High Speed Integrated Circuit
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSD	XML Schema Definition

BIBLIOGRAPHIE

[Acc98] Accellera, Accellera Verilog Analog Mixed-Signal Group, "Verilog-AMS Home", http://www.eda.org/verilog-ams/ 1998

[ADE03] L. Auret, M. Dilhan, D.Esteve, A.M. Gué, "Procédé de realisation d'életrodes, circuit intégré à électrode(s) protégée(s) et sonde électrochimique », Rapport LAAS No03675, Brevet NEOSENS, WO 2005/029063, FR2859821, 2003, 24p

[Adv] Mentor Graphics Corporation, « AdvanceMS Datasheet » Mentor Graphics Corporation.2001 http://www.mentor.com/products/ic_nanometer_design/ms_circuit_simulation/advance_ms/index.cfm

[Alb05] V. Albert, "Traduction d'un modèle de système hybride basé sur réseau de Petri en VHDL-AMS", Mémoire de stage Master, Université Paul Sabatier Toulouse III, LAAS-CNRS, 2005.

[And95] L. Andrieux, « Caractérisation du transistor bipolaire à hétérojonction GaAlAs/GaAs : utilisation en amplification hyperfréquence de puissance », 1995 Thèse LAAS CNRS, Rapport LAAS n°953

[Ans] ANSOFT Corporation, « System Modeling » Simplorer http://www.ansoft.com/products/em/simplorer/

[Arg] ArgoUML http://argouml.tigris.org/

[ATL] Atlas langage de transformation http://www.sciences.univ-nantes.fr/lina/atl/

[ATL06] ATL langage de transformation http://www.sciences.univ-nantes.fr/lina/atl/

[Béz06] J. Bézivin, « Introduction to Model Engineering », 2006, ATLAS Group (INRIA et LINA), Nantes http://www.modelware-ist.org/index.php?option=com_remository&Itemid=74&func=fileinfo&id=72

[BHE04] J. Baylac, J.C. Hamon, D. Esteve. « Conception système, traduction des réseaux de Petri en langage VHDL-AMS ». Rapport de stage LAAS-CNRS. Toulouse, France. 2004.

[BJK06] J. Bézivin, F. Jouault, I. Kurtev, P. Valduriez, *"Model-Based DSL Framework"* Companion to the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, OR, USA. ACM, pages 602—616. 2006

[Bro03] D. Brown « A beginners guide to UML ». University of Kansas, Department of Electrical Engineering and Computer Science Dustan Thomas Consulting 2003 http://consulting.dthomas.co.uk

[CEM02] C. Clauss, H. Elmqvist, S. E. Mattson, *"Mixed Domain Modeling in Modelica"*, FDL'02, Sept. 23-27, 2002.

[CEN04] M. COMBACAU, P. ESTEBAN, A. NKETSA. « Modélisation, Analyse et Mise en œuvre de commandes basées sur les réseaux de Petri ». Accepté et à paraître dans la revue Techniques de l'ingénieur - Informatique Industrielle. 2004.

[Cha02] V. Chaudhary, "Automatic Generation of Behavioral Models in VHDL-AMS and VTB", December 2002, University of Arkansas.

[Coo01] R. Scott Cooper. *« The Designer's Guide To Analog And Mixed-Signal Modeling »* Avant! Press, 2001. ISBN 0-9705-953-0-1.

[Cor] CORBA (Common Object Request Broker Architecture) OMG
http://www.omg.org/corba/

[Cos06] Cosimulation hétérogènes dans des environnements distribués : Cosimate Editeur : CHIASTEK
http://www.chistek.com/products/cosimate.html

[DMG97] G. De Michelli, R. Gupta. « Hardware / Software Co-Design ». Proceedings of the IEEE, Vol 85, No 3, March 1997.

[DSK04] P. Desgreys, M. Karray, S. Snaidero, Y. Hervé, J. Oudinot *"SoC modelling for virtual prototyping with VHDL-AMS"* Forum on Specification and Design Languages FDL 2004 Lille

[Dym07] Dymola Editeur : Dynasim
http://www.dynasim.se/

[Ecl] Eclipse Plateforme d'Outils
http://www.eclipse.org/m2m/atl/

[Ecl] Environnement Eclipse UML plugin :
http://www.eclipse.org/modeling/mdt/?project=uml2-uml

[Eia98] EIA-632
http://www.geia.org

[FAP97] M. Fernandes, M. Adamski, J. Proenca, Universidade do Minho Braga .« VHDL Generation from Hierarchical Petri Net Specifications of Parallel Controllers » Portugal. IEE Proceedings: Computers and Digital Techniques, 144(2):127--37, Mar.1997.

[Fav04] J.M. Favre, "Towards a Basic Theory to Model Driven Engineering", 3rd Workshop in Software Model Engineering, WiSME 2004
http://www-adele.imag.fr/~jmfavre

[FCM04] M. Francis, V. Chaudhary, H. A. Mantooth, *« Compact Semi-Conductor Device Modeling Using Higher Level Methods »* Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on Volume 5, 23-26 May 2004 Page(s):V-109 - V-112 Vol.5.

[GAE06a] *VHDL-AMS Model Creation* : Guihal, D.; Andrieux, L.; Esteve, D.; Cazarre, A.; Mixed Design of Integrated Circuits and System MIXDES 2006 Proceedings of the International Conference 22-24 June 2006 Page(s): 549–554

[GAE06b] *VHDL-AMS model generation from other HDL Language : Application to a MOSFET Switching Power Device* : Guihal, D.; Andrieux, L.; Esteve, D Forum on Specification and Design Languages FDL 2006 Proceedings of the International Conference 19-22 September 2006 Page(s): 77 – 82

[GK83] D. D. Gajski, R. H. Kuhn, *"New VLSI Tools"*, IEEE Computer, Vol. 16, no. 12 (December 1983), pp. 11-14.

[GO04] S. Guessab, J. Oudinot, *"How can MATLAB/Simulink co-exist with VHDL-AMS?"*, Euro DesignCon 2004.

[Gui03] D. Guihal, « Mémoire de Stage de Fin d'étude. Codesign HW-SW : Etude d'une interface entre outil de conception système et VHDL-AMS ». ENSPS Strasbourg, Airbus France, Avionics and simulation products 2003.

[Ham02] J.C. Hamon « Co-Conception Hardware/Software Conception Système et Prototypage virtuel », Déc. 2002 rapport LAAS n°02578.

[Ham05] J-C Hamon, « Méthodes et outils de la conception amont pour les systèmes et les microsystèmes » Rapport LAAS No05064 Doctorat, Institut National Polytechnique, Toulouse, 1er Février 2005, 178p.

[Her00] Grammaire VHDL-AMS BNF html par Y. Hervé : http://www-ensps.u-strasbg.fr/coursen/OLD%20Option3A/ams_bnf.html

[Her02] Hervé Yannick. *« VHDL-AMS Applications et enjeux industriels »*. Dunod, Paris, 2002. ISBN 2-10-005888-6.

[Her03] Y. Hervé, "Simple models for complex systems : A-FSM template", Forum on specification and Design Language, September 23-26 2003

[Her06] Y. Hervé, « La dynamique VHDL-AMS : Enjeux et Difficultés de la transformation de modèles » Séminaire LAAS-TOOLSYS, 7 février 2006

[HSc03] J.C.Hamon, P. Schmitt « Mise en place d'outils de conception de microsystèmes : Application à la conception d'un accéléromètre » article LAAS 2003.

[Inf06] Infineon Website (OptiMOS2 model) : http://www.infineon.com/cgi-bin/ifx/portal/ep/channelView.do?channelId=-64585&channelPage=%2Fep%2Fchannel%2FproductOverview.jsp&pageTypeId=17099

[JBK06] F. Jouault, J. Bézivin, "KM3: a DSL for Metamodel Specification" Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, LNCS 4037, Bologna, Italy, pages 171—185. 2006.

[Jim00] F. Jimenez, « Spécification et Conception de Micro systèmes Basés sur des Circuits Asynchrones », thèse doctorat, Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS. Toulouse, France 2000.

[Küh05] T. Kühne, *"What is Model?"*, Dagstuhl Seminar Proceedings 04101, http://drops.dagstuhl.de/volltexte/2005/23

[Mal02] J. Malenfant, « Modélisation de la sémantique formelle des langages de programmation en UML et OCL », Rapport INRIA, Juillet 2002, n°4499, ISSN 0249-6399

[Mat] MATLAB Editeur : The MathWorks http://www.mathworks.fr/products/matlab/

[Mata] MatLab/Simulink Editeur : the MathWorks http://www.mathworks.com/products/simulink/?BB=1

[Matb] Real Time Workshop pour MatLab/Simulink Editeur : the MathWorks http://www.mathworks.com/products/rtw/

[Matc] Simulink HDL coder pour MatLab/Simulink Editeur : the MathWorks http://www.mathworks.com/products/slhdlcoder/

[Mau05] R. Maurice, « Contribution à la méthodologie de conception système : application à la réalisation d'un microsystème multicapteurs communicant pour le génie civil » Rapport LAAS No05646 Doctorat, Institut National Polytechnique, Toulouse, 15 Décembre 2005.

[Mcl99] M. P. McLaughlin, *"A Tutorial on Mathematical Modeling"*, 1999, http://www.causascientia.org/math_stat/tutorial.pdf

[MCV05] T. Mens, K. Czarnecki, P. Van Gorp, *"A Taxonomy of Model Transformations"* Dagstuhl Seminar Proceedings http://drops.dagstuhl.de/opus/volltext/2005/11

[MDA] Model Driven Architecture http://www.omg.org/mda/

[MFC03] P. Mallick, M. Francis, V. Chandrasekhar, A. Austin, H. A. Mantooth, *"Achieving Language Independence with Paragon"*, Behavioral Modeling And Simulation Workshop (BMAS 2003), October 8, 2003.

[Mmo07] MathModelica System Designer Editeur : MathCore http://www.mathcore.com/products/mathmodelica/

[Mod] Outil pour la vérification numérique : MODELSIM Editeur : MENTOR GRAPHICS http://www.mentor.com/products/fv/digital_verification/modelsim_se/index.cfm

[Mod07] Modelica http://www.modelica.org

[MOF06] Meta Object Facility Norme OMG Version 1.4 http://www.omg.org/mof/

[Neo07] NEOSENS Labège http://www.neo-sens.com/home/index.php

[NVH00] F. Nacabal, C. Valderrama, F. Hessel, "Co-simulation C-VHDL pour la validation fonctionnelle de logiciel embarqué", Technique et science informatiques, Vol. 19 n° 8 pp. 1097- 1126, Hermes Science Publications, Oct. 2000.

[OMG] Object Management Group
http://www.omg.org/

[Omo] Plugin Eclipse Omondo
http://www.omondo.com/

[Opm07] OpenModelica Project
http://www.ida.liu.se/~pelab/modelica/OpenModelica.html

[Orc06] Pspice module de Orcad Editeur : Cadence
http://www.cadence.com/orcad/

[PAR04]] PARAGON Version 1.1.3 MSCAD (Mixed Signal Computer Aided Design Lab) University Arkansas
http://mixedsignal.eleg.uark.edu/paragon.html

[Pen06] P. Pennarun, « Conception et intégration d'un micro-initiateur sécurisé à base de micro-interrupteurs pyrotechniques sur silicium » Rapport LAAS No06717 Doctorat, Institut des Sciences Appliqués, Toulouse, 23 Octobre 2006.

[PLV05] F. Pêcheux, C. Lallement, A. Vachoux, *« VHDL-AMS and Verilog-AMS as Alternative Hardware Description Languages for Efficient Modeling of Multidiscipline Systems »*, IEEE Transactions On Computer-Aided design Of Integrated Circuits And Systems, Vol. 24, No. 2, February 2005.

[Pyt07] Langage de Programmation Python
http://www.python.org/

[Roc06] S. Rochet, « Méthodologie de modélisation de projet dans l'ingénierie système : Application à l'EIA-632 », INSA Toulouse, 21 Nov. 2006

[Ros97] C. Rossi, « Conception et réalisation d'un système de réhydratation pour patch transdermique à partir de micro actionneurs pyrotechniques » Rapport LAAS No97520 Doctorat, Institut des Sciences Appliqués, Toulouse, 18 Décembre 1997.

[Row94] J. Rowson, "Hardware-Software Co-Simulation", Proceedings of DAC, p439-440, 1994.

[Sab] Logiciel de simulation de système à technologie mixte (langage MAST) : SABER Editeur : SYNOPSYS
http://www.synopsys.com/products/mixedsignal/saber/saber.html

[Sab] Outil de simulation de système mixte SABER Editeur : SYSNOPSYS
http://www.synopsys.com/products/mixedsignal/saber/saber.html

[SDC06] G. Savaton, J. Delatour, K. Courtel, "Roll your own Hardware Description Language", ESEO 2006

[Sea] Covérification Hardware/Software : Seamless Editeur: MENTOR GRAPHICS
http://www.mentor.com/products/fv/hwsw_coverification/seamless/index.cfm

[Seb06] A. Sebatware, « Contribution à l'étude des langages d'un domaine : Application au domaine des procédures opérationnelles des systèmes embarqués spatiaux », Mémoire Ingénieur CNAM, 4 Décembre 2006

[Sma06] Dolphin Integration. "Dolphin Medal. New Features in SMASH™
http://www.dolphin.fr/medal/smash/smash_overview.html

[STA] STANAG (Standardization agreement) 4187 : Fuzing systems-safety design requirement

[Sta73] H. Stachowiak, *"Allgemeine Modelltheorie"*, Springer-Verlag, Wien and New York, 1973, ISBN 3-211-81106-0.

[Syn04] Synopsys®. « OpenMAST Overview ».
http://www.openmast.org/overview/overview.html Synopsys® Etats Unis d'Amérique 2004.

[Sys03] SysML (System Modeling Language) OMG
http://www.omgsysml.org/

[Sys50] Logiciel de simulation de système analogique, numérique et mixte (langage VHDL-AMS) : SystemVision (Version 5.0) Editeur : MENTOR GRAPHICS
http://www.mentor.com/products/sm/systemvision/index.cfm

[Tai02] Y. Hervé, *"VHDL-AMS : un atout pour la conception système"*, TAISA'2002, 12-13 Septembre 2002.

[TAP03] D. Teegarden, P. Ashenden, G. Peterson, *"The System Designer's Guide to VHDL-AMS"*, 2003 Morgan Kaufmann Publishers, ISBN 1-55860-749-8

[Top06] Projet TOPCASED Plateforme Open Source pour l'ingénierie dont : Groupe de Travail WP5 Transformations de modèles ; Groupe de Travail WP1 Spécifications
http://www.topcased.org/

[VDA07] Development of VHDL-AMS Libraries for Automotive Applications VDA
http://fat-ak30.eas.iis.fraunhofer.de/vdalibs/doc/

[Vhd00] Site Internet Allemand sur le VHDL-AMS : www.VHDL-AMS.de
http://www.vhdl-ams.org/Language/language.html

[Vhd99] 1076.1-1999 *IEEE Standard VHDL Analog and Mixed-Signal Extensions Language Reference Manual*, IEEE Press, ISBN 0-7381-1640-8.

[VIP06] Y. Hervé Systems'VIP Workshop Transformations de Modèle LAAS-CNRS 7 Février 2006

[Vis] Visio Editeur : Microsoft
http://office.microsoft.com/fr-fr/visio/default.aspx

[Zoo07] Projet AM3 Plateforme Eclipse Bibliothèques de Meta Modèles
http://www.eclipse.org/gmt/am3/zoos/

ANNEXE 1

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="MAST" name="resistance">
  <Design_Entity Identifier="resistor">
    <Header xsi:type="Temple_Header">
      <Object_Def xsi:type="Pin" Identifier="p" Nature="electrical"/>
      <Object_Def xsi:type="Pin" Identifier="m" Nature="electrical"/>
      <Object_Def xsi:type="Parameter" Identifier="res" type="real"/>
    </Header>
    <Body xsi:type="Temple_Body>
      <Object_Def xsi:type="Branch_Variable" Identifier="ir" branch_name_separator="->" plus_pin_aspect="p" minus_pin_aspect="m" type="real"/>
      <Object_Def xsi:type="Branch_Variable" Identifier="vr" branch_name_separator=", " plus_pin_aspect="p" minus_pin_aspect="m" type="real"/>
      <ExecStat xsi:type="Labeled_Equation" label=" ">
        <Obj_Explicit xsi:type="Branch_Variable" Identifier="ir" branch_name_separator="->" plus_pin_aspect="p" minus_pin_aspect="m" type="real"/>
        <Expression xsi:type="Multiplicative_operator">
          <arg1 xsi:type="Branch_Variable" Identifier="vr" branch_name_separator=", " plus_pin_aspect="p" minus_pin_aspect="m" type="real"/>
          <arg2 xsi:type="Parameter" Identifier="res" type="real"/>
        </Expression>
      </ExecStat>
    </Body>
  </Design_Entity>
</xmi:XMI>
```

ANNEXE 2

Définitions formelles des règles et code du modèle de transformation MAST vers VHDL-AMS restreint aux concepts de modélisation utilisés pour le modèle de la résistance.

Règles :

Nous dressons de façon informelle les règles de transformations à appliquer sur ces deux meta modèles :

- **Règle 1** : **Header2Entity**
  - Pour chaque Template_Header MAST, une Entité VHDL-AMS doit être créée
  - Le nom doit être le même
  - Le portmap de l'entité VHDL-AMS doit contenir la déclaration des objets générés par la règle **Pin2Terminal**
  - Le genericmap de l'entité VHDL-AMS doit contenir la déclaration des objets générés par la règle **Param2Constant**

- **Règle 2** : **Body2Architecture**
  - Pour chaque Template_Body MAST, une Architecture VHDL-AMS doit être créée
  - La référence à l'entité doit être la même
  - La partie déclarative de l'architecture doit contenir la déclaration des objets générés par la règle **Branch2QuantityBranch**
  - La partie instructions simultanées doit contenir les instructions simultanées générées par **Lab_Equ2Equ**

- **Règle 3** : **Param2Constant**
  - Pour chaque Paramètre MAST, une constante VHDL-AMS doit être créée
  - Le nom doit être le même
  - Le type doit être le même
  - La valeur par défaut doit être la même

- **Règle 4** : **Pin2Terminal**
  - Pour chaque Pin MAST de type conservatif, un Terminal VHDL-AMS doit être créé
  - Le nom doit être le même
  - La nature doit être la même

- **Règle 5** : **Lab_Equ2Equ**
  - Pour chaque Labeled_Equation MAST, une Equation VHDL-AMS doit être créée
  - Le premier argument de l'équation doit contenir l'objet appelé explicitement
  - Le deuxième argument de l'équation doit contenir les expressions générées par les règles **Object2Object** et **MultiMAST2MultiVHDLAMS**

- **Règle 6 : Object2Object**
  - Pour chaque Object MAST, un Object VHDL-AMS doit être créé
  - Le nom doit être le même

- **Règle 7 : MultiMAST2MultiVHDLAMS**
  - Pour chaque opération de multiplication MAST, une opération de multiplication VHDL-AMS doit être créée
  - Les deux arguments doivent être les mêmes

- **Règle 8 : Branch2QuantityBranch**
  - Pour chaque Branch_Variable MAST, une Quantité VHDL-AMS doit être créée
  - Le nom, le type, la valeur par défaut doivent être les mêmes
  - Les noms des terminaux plus et moins doivent être les mêmes
  - L'aspect de la quantity VHDL-AMS reprend l'aspect de la quantity de branche MAST

Code du Modèle :

```
module MAST2VHDLAMS
create OUT : VHDLAMS from IN : MAST

rule Header2Entity {
 from
       e : MAST!Template_Header
 to
       out : VHDLAMS!Entity (
                   identifier <- e.identifier,
                   genericmap <- g,
                   portmap <- p
                   )
       g : VHDL!genericmap (
                   generic <- e.object_def
                   )
       p : VHDL!portmap (
                   port <- e.object_def
                   )
 }


rule Body2Architecture {
 from
       e : MAST!Template_Body
 to
       out : VHDLAMS!Architecture (
                   identifier <- 'behavior',
                   entity_name <- e.identifier,
                   Simult <- e.execstat,
                   Decla <- e.object_def
                         )
 }
```

```
rule Param2Constant {
 from
       e : MAST!Parameter
 to
       out : VHDLAMS!Constant (
                   identifier <- e.identifier,
                   type <- e.type,
                   value <- e.value
                         )
 }

rule Pin2Terminal {
 from
       e : MAST!Pin(e.conservatif)
 to
       out : VHDLAMS!Terminal (
                   identifier <- e.identifier,
                   nature <- e.nature
                         )
 }

rule Lab_Equ2Equ {
 from
       e : MAST!Labeled_Equation
 to
       out : VHDLAMS!Equation (
                   arg1 <- exp1,
                   arg2 <- e.expression
                         )
       exp1 : VHDLAMS!Object (
                   identifier <- e.Obj_Explicit
                               )
 }

rule Object2Object {
 from
       e : MAST!Object
 to
       out : VHDLAMS!Object (
                   identifier <- e.identifier
                         )
 }

rule MultiMAST2MultiVHDLAMS {
 from
       e : MAST!Multiplicative_operator
 to
       out : VHDLAMS!Multiplying_factor (
                   arg1 <- e.arg1,
                   arg2 <- e.arg2
                               )
 }
```

```
     rule Branch2QuantityBranch {
      from
            e : MAST!Branch_Variable
      to
            out : VHDLAMS!QUantity (
                        identifier <- e.identifier,
                        type <- e.type,
                        plus_terminal_name <- e.plus_aspect,
                        minus_terminal_name <- e.minus_aspect,
                        quantity_aspect <- e.BranchQUantityaspect()
                              )
      }

     helper context MAST!Branch_Variable def : BranchQUantityaspect() :
string =
      if self.branch_name_separator->'->' then
            'through'
      else
            'across'
      endif;
```

ANNEXE 3 : CODES MODELES VHDL-AMS

1. Modèle Partie Commande Système Mise à Feu

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.electrical_systems.all;

ENTITY Partie_Commande_Pyro IS

     generic (    constant Comparateur_Acc : real;
                  constant clk_interne_periode : time := 20 us;
                  constant NPULSE : integer := 10
      );

     port ( Terminal Acc_Input_Term : electrical;

            signal Requete_Ster : in std_logic;
            signal Requete_Arm : in std_logic;
            signal Requete_Desarm : in std_logic;
            signal Requete_Expl : in std_logic;

            signal Commande_Ster : out std_logic := '0';
            signal Commande_Arm : out std_logic := '0';
            signal Commande_Desarm : out std_logic := '0';
            signal Commande_Expl : out std_logic := '0'
      );

end Partie_Commande_Pyro;

ARCHITECTURE beh of Partie_Commande_Pyro IS

     signal Phase : integer := 0;
     signal clk_interne : std_logic := '0';
     signal Acc_Input_Comp : std_logic := '0';
     signal Mvt : boolean := false;

     quantity Acc_Input across Acc_Input_Term;

BEGIN

     --management des Etats
     process
      begin
            wait on Phase,Requete_Ster, Requete_Arm, Requete_Desarm,
     Requete_Expl;
            case Phase is
                  --Phase 0 : Veille Sécurisée
                  when 0 =>
                        if Requete_Ster'event and Requete_Ster = '1' then
                              Phase <= 1;
                        end if;
                        if Requete_Arm'event and Requete_Arm = '1' and Mvt
     then
                              Phase <= 2;
```

```
                        end if;
                    --Phase 1 : Initiateur Stérilisé
                    when 1 =>
                        Commande_Ster <= '1';
                    --Phase 2 : Initiateur Armé Attente Déclenchement
                    when 2 =>
                        Commande_Arm <= '1';
                        if Requete_Desarm'event and Requete_Desarm = '1'
    then
                            Commande_Desarm <= '1';
                            Phase <= 0;
                        end if;
                        if Requete_Expl'event and Requete_Expl = '1' and
    Mvt then
                            Phase <= 3;
                        end if;
                    --Phase 3 : Initiateur Actionné
                    when 3 =>
                        Commande_Expl <= '1';
                    when others =>
            end case;
    end process;


Comparateur_Acc_Proc : process
  begin
       wait on Acc_Input'above(Comparateur_Acc);
      if Acc_Input'above(Comparateur_Acc) then
            Acc_Input_Comp <= '1';
      else
            Acc_Input_Comp <= '0';
      end if;
  end process Comparateur_Acc_Proc;

  clk_interne <= not clk_interne after clk_interne_periode/2.0;

 Generateur_Signal_Mvt : process
    variable compteur : integer := 0;
      begin
              wait on Acc_Input_Comp,clk_interne;
           if Acc_Input_Comp'event and Acc_Input_Comp='1' and
    clk_interne='1' then
            compteur := compteur + 1;
            if compteur = NPULSE then
                  Mvt <= true;
            end if;
           end if;
           if clk_interne'event and clk_interne='0' then
            compteur :=0;
           end if;
  end process Generateur_Signal_Mvt;


end beh;
```

2. Interrupteur on-off Système Mise à Feu

```
library ieee;
use ieee.electrical_systems.all;
use ieee.fundamental_constants.all;
use ieee.math_real.all;
use work.all;

ENTITY initiateur_modified IS

     generic (      --caractéristiques environnement
             constant Patm: real:=1.0e+05;    --pression atmosphérique (en Pa)
             constant T0 : real := 295.0; --Température initale environnement (°K)
                    -- caractéristiques résistance
             constant a_chauff : real := 0.540;-- côté du carré formé par la
             résistance (mm)
             constant section_resist: real:= 30.0*1.0e-06; --section de la
     résistance (mm²)
             constant Resist : real := 500.0; -- Valeur de la résistance (ohm)
             constant J: real:= 0.1; -- Densité de courant maxi passant dans la
     résistance (A/m²)
                    --caractéristique matériau pyrotechnique
             constant k : real := 0.15;-- Conductivité thermique du mat pyro
     (W/m.K)
             constant ro : real := 1500.0; -- Densité du mat pyro (Kg/m**3)
             constant Cp_pyro : real := 1850.0; -- Chaleur spécifique du mat pyro
     (J/kg.K)
             constant Cv_pyro: real := 1450.0; --Chaleur spécifique à volume
     constant (J/kg.K)
             constant gamma_pyro: real:=1.3;  --coef adiabatique mat pyro
             constant r_pyro : real := 435.0; -- constante réduite gaz parfait pour
     mat pyro ( J/kg.K)
             constant Tal : real := 523.0; -- Température d'allumage (initiation)
     (°K)
             constant Vcomb_max : real := 30.0; --vitesse de combustion du matériau
     pyrotechnique (mm/s)
             constant alpha : real := 0.055;  --pourcentage de propergol en contact
     avec avec le Si
             constant Hcomb : real := 3654.0*1.0e+3;-- enthalpie de combustion
     (J/kg)
                    --géométrie du matériau pyrotechnique
             constant h_pyro : real := 0.200; -- épaisseur du matériau pyrotechnique
     dans le réservoir (mm)
                    --caracteristiques reservoir
             constant a_res : real := 1.5; --côté du carré formé par la membrane
     (mm)
             constant h_res : real := 0.340;  -- hauteur du réservoir (mm)
             constant Prupture : real := 20.0*1e+5;  --pression de rupture de la
     membrane (Pa)
             constant h : real := 100.0       -- coef d'échange convectif avec paroi
     réservoir (W/m².K)
             );

     port (  terminal Eini, Sini : electrical; --Définition du noeud d'entrée et
     sortie Res therm associant le couple V,I
```

```
            terminal Ep, Sp : electrical  -- Définition noeuds entrée/sortie piste
            );

end initiateur_modified;

ARCHITECTURE arch_thermo of initiateur_modified IS


     --caractéristiques électriques:
     constant Imax : real := J*section_resist*1.0e-06;           -- valeur
     d'intensité max pouvant

            --traverser la résistance (A)
     quantity Vp across Ip through Ep to Sp;                              -- valeur
     d'entrée (V,I)
     quantity Vini across Iini through Eini to Sini;

     Constant Ron : real := 1.0e-12;
     Constant Roff : real := 1.0e12;

     -- caractéristique enceinte:
     constant Surf_memb: real := a_res**2*1.0e-6;                --Surface de la
     membrane qui est aussi celle du mat pyro (m²)
     constant Vol_total : real := h_res*(a_res)**2*1.0e-9; -- Volume total de
     l'enceinte (m**3)

     constant Vol_gaz_initial:real := (h_res - h_pyro)*(a_res)**2*1.0e-9;-- Volume
     de gaz initial (m**3)
     quantity Vol_gaz : real:=Vol_gaz_initial;                            -- Volume
     de gaz aucours du temps (m**3)

     quantity Surf_Si : real:=4.0*a_res*(h_res-h_pyro)*1.0e-6; --Surface de
     silicium du reservoir (défini la surface d'échange en m²avec le gaz)

     quantity Pr : real := Patm;                                                --
     Pression dans l'enceinte (Pa)

     quantity Transition_Vol : real :=0.0;                                -- loi de
     passage d'un volume limité à un volume "infini", loi optimisée pour rédire les
     discontinuités

     -- caractéristiques de l'air
     constant Cp_air:real:=1004.5;                                              --
     Chaleur spécifique de l'air pression constante (J/kg.K)
     constant Cv_air : real :=717.5;                                      -- Chaleur
     spécifique de l'air volume constant (J/Kg.K)
     constant Gamma_air : real := 1.4;                                    -- coef
     adiabatique
     constant r_air : real := 287.0;                                      --constante
     réduite des gaz parfaits (J/kg.K)

     --caractéristique du mélange
     quantity Cp : real:= Cp_air;                                               --
     chaleur spécifique du mélange air-pyro pression constante (J/kg.K)
     quantity Cv :real:= Cv_air;                                                --
     chaleur spécifique du mélange air-pyro volume constant (J/kg.K)
     quantity Gamma : real:= Gamma_air;                                         --
     coef adiabatique du mélange air-pyro
```

```
    quantity r : real:= r_air;                                                --
    constante réduite des gaz parfaits pour le mélange(J/kg.K)

    -- masse de gaz (2éléments : air et produit de combustion)
    quantity T_gaz : real:= 300.0;                                            --
    température au sein du gaz
    constant ro_air : real := 1.2;                                            --
    masse volumique de l'air (kg/m**3)
    quantity m_air : real;
     -- masse d'air dans le mélange gazeux(kg)
                              -- Initialement c'est la masse de gaz contenu dans
    le volume de gaz initial
                              -- Une fois la membrane cassée elle augmente du
    fait de l'ajout de l'air extérieur
    quantity m_gaz : real ;                                                   --
    masse totale de gaz dans l'enceinte (kg)
    quantity m_comb      : real := 0.0;
     -- masse issu de la combusiton du mat pyro(kg)
    quantity E_gaz : real:=0.0; --Energie du système (J)
    constant h_air : real := 15.0;

    -- materiau pyrotechnique
    constant m_pyro : real := ro*h_pyro*1.0e-3*Surf_memb; -- masse de combustible
    à l'état solide (kg)
    constant k_Si : real := 100.0;                                            --
    conductivité thermique du Si du réservoir (W/m².K)
    constant a_critique: real:=4.0*alpha*k_Si/(ro*Cp_pyro*Vcomb_max*1.0e-03);--
    dimension critique du réservoir

            --en dessous de laquelle plus d'initiation (m).
    constant Vcomb : real := Vcomb_max*1.0e-03*(1.0-(a_critique/(a_res*1.0e-
    03)));-- Vitesse de combusiton réelle, compte tenu des pertes

            --dues aux parois de l'enceinte  (m/s)
    constant Surf_chauff :real := a_chauff**2.0;
    quantity TsurfRes : real := 300.0;
    quantity Puissance_Joule : real;


    --environnement:
    constant Vol_infini : real := 1.0e+4*Vol_total;             --Vol infini
    virtuel quand la membrane casse
    constant Surf_infini : real := 10000.0*4.0*a_res*h_res*1.0e-6;--surface
    d'échange virtuelle du gaz avec l'air

    -- Signaux de management des différentes phases:
    signal phase: integer :=0;
    signal rupture, allumage, fin_comb, Vol_inf_atteint: boolean;
    signal t_phase : real :=0.0;

BEGIN
     --synchronisation
    break on phase;
    break on rupture;
    break on allumage;
    break on fin_comb;

    --transitions
```

```
rupture <= Pr'above(Prupture);                  --on dépasse la pression de
rupture
allumage <= TsurfRes'above(Tal);                --on dépasse la température
d'initiation à la surface
                                                                -- du matériau
pyrotechnique.
fin_comb <= m_comb'above(m_pyro);
--management des phases
process
 begin
        wait on phase,rupture, allumage, fin_comb;
        case phase is
              --phase d'allumage
              when 0 =>
                    if allumage then
                    phase <= 1;
                    end if;
              -- phase de combustion (membrane non cassée)
              when 1 =>
                    if rupture then
                    phase <= 2;
                    elsif fin_comb then
                    phase <= 4;
                    end if;
              when 2 =>
                    if fin_comb then
                    phase <= 3;
                    end if;
              when others =>
        end case;
end process;

 --équations associées à chaque phase
  case phase use

-----------------------------------------------------------------
  --A) ALLUMAGE
        when 0 =>
-----------------------------------------------------------------

--équations du solide:

Vini == Resist*Iini;--loi d'ohm sur resistance en poly

Puissance_Joule == Vini*Iini;

Puissance_Joule'integ == h_air*Surf_memb*(TsurfRes - T0) + Cp_pyro * ro *
1.0e-9 * Surf_chauff * h_pyro * (TsurfRes - T0);

Vp == Ron * Ip;

--équations du gaz:

E_gaz'dot==0.0;
E_gaz==Pr*Vol_gaz/ (Gamma-1.0);
T_gaz*(m_gaz*r)== Pr*Vol_gaz ;
Vol_gaz == Vol_gaz_initial;
m_gaz==m_air+m_comb;
```

```
m_air == ro_air*Vol_gaz_initial ;
m_comb== 0.0;
Surf_Si'dot == 0.0;
Transition_Vol==0.0;
  --équations constantes des gaz parfaits:
  Cp-Cv== r;
  gamma== Cp/Cv;
  Cp== Cp_air;
  Cv== Cv_air;

------------------------------------------------------------
-- B) Combustion
  when 1 =>
------------------------------------------------------------

  --équations du solide:

Vini == Resist*Iini;--loi d'ohm sur resistance en poly

Puissance_Joule == Vini*Iini;

Puissance_Joule == h_air*Surf_memb*(TsurfRes - T0) + Cp_pyro * ro * 1.0e-9 *
Surf_chauff * h_pyro * TsurfRes'dot;

Vp == Ron * Ip;

--équations du gaz:
E_gaz'dot==m_comb'dot*Hcomb-Pr*Vol_gaz'dot-h*(T_gaz-T0)*Surf_Si;
E_gaz == Pr*Vol_gaz/ (Gamma-1.0);
T_gaz*(m_gaz*r)== Pr*Vol_gaz ;

Vol_gaz'dot== Vcomb*(a_res)**2*1.0e-06;
Surf_Si'dot == 4.0*Vcomb*a_res*1.0e-3;
m_gaz==m_air+m_comb;
m_air'dot==0.0;       --la membrane n'a pas encore cassée donc le volume d'air
                            --extérieur qui se mélange au volume de combustion
est nul
                            -- La masse d'air est donc constante
m_comb'dot== ro*Vcomb*Surf_memb;

Transition_Vol==0.0;

  --équations constantes des gaz parfaits:
        Cp-Cv== r;
        gamma== Cp/Cv;
        Cp== Cp_air*(m_air/m_gaz)+ Cp_pyro*(m_comb/m_gaz);
        Cv== Cv_air*(m_air/m_gaz)+ Cv_pyro*(m_comb/m_gaz);


------------------------------------------------------------
-- C) Fin COMBUSTION (membrane cassée)
        when 2 =>
------------------------------------------------------------


Iini == 0.0;--loi d'ohm sur resistance en poly

Puissance_Joule == Vini*Iini;
```

```
Puissance_Joule == h_air*Surf_memb*(TsurfRes - T0) + Cp_pyro * ro * 1.0e-9 *
Surf_chauff * h_pyro * TsurfRes'dot;

Vp == Roff * Ip;

--équations du gaz:
E_gaz'dot==m_comb'dot*Hcomb-Pr*Vol_gaz'dot;
Pr==Patm;
T_gaz*(m_gaz*r)== Pr*Vol_gaz ;

Vol_gaz'dot== 0.0;
Surf_Si'dot == 0.0;
m_gaz==m_air+m_comb;
m_air'dot==0.0;

m_comb'dot== ro*Vcomb*Surf_memb;

Transition_Vol==0.0;

  --équations constantes des gaz parfaits:
        Cp-Cv== r;
        gamma== Cp/Cv;
        Cp== Cp_air*(m_air/m_gaz)+ Cp_pyro*(m_comb/m_gaz);
        Cv== Cv_air*(m_air/m_gaz)+ Cv_pyro*(m_comb/m_gaz);

------------------------------------------------------------
-- D) COMBUSTION Finie (membrane cassée)
        when 3 =>
------------------------------------------------------------


Iini == 0.0;--loi d'ohm sur resistance en poly

Puissance_Joule == Vini*Iini;

Puissance_Joule == h_air*Surf_memb*(TsurfRes - T0) + Cp_pyro * ro * 1.0e-9 *
Surf_chauff * h_pyro * TsurfRes'dot;

Vp == Roff * Ip;

--équations du gaz:
E_gaz'dot==0.0;
Pr==Patm;
T_gaz*(m_gaz*r)== Pr*Vol_gaz ;

Vol_gaz'dot== 0.0;
Surf_Si'dot == 0.0;
m_gaz==m_air+m_comb;
m_air'dot==0.0;

m_comb'dot== 0.0;

Transition_Vol==0.0;

  --équations constantes des gaz parfaits:
        Cp-Cv== r;
        gamma== Cp/Cv;
```

```
            Cp== Cp_air*(m_air/m_gaz)+ Cp_pyro*(m_comb/m_gaz);
            Cv== Cv_air*(m_air/m_gaz)+ Cv_pyro*(m_comb/m_gaz);


    ------------------------------------------------------------
    -- D) COMBUSTION Finie (membrane non cassée)
            when others =>
    ------------------------------------------------------------

    --équations du solide:
    Vini == Resist*Iini;--loi d'ohm sur resistance en poly

    Puissance_Joule == Vini*Iini;

    Puissance_Joule == h_air*Surf_memb*(TsurfRes - T0) + Cp_pyro * ro * 1.0e-9 *
    Surf_chauff * h_pyro * TsurfRes'dot;

    Vp == Ron * Ip;

    --équations du gaz:
    E_gaz'dot==0.0;
    E_gaz==Pr*Vol_gaz/ (Gamma-1.0);
    T_gaz*(m_gaz*r)== Pr*Vol_gaz ;
    Vol_gaz == Vol_gaz_initial;
    m_gaz==m_air+m_comb;
    m_air == ro_air*Vol_gaz_initial ;
    m_comb== 0.0;
    Surf_Si'dot == 0.0;
    Transition_Vol==0.0;
      --équations constantes des gaz parfaits:
      Cp-Cv== r;
      gamma== Cp/Cv;
      Cp== Cp_air;
      Cv== Cv_air;

    end case;


end arch_thermo;
```

3. OptiMOS

3.1 BSC032N03S.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--
-- This model is a component of the Mentor Graphics VHDL-AMS educational open
-- source model library, and is covered by this license agreement. This model,
-- including any updates, modifications, revisions, copies, and documentation
-- are copyrighted works of Mentor Graphics. USE OF THIS MODEL INDICATES YOUR
-- COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH
-- IN THIS LICENSE AGREEMENT.  Mentor Graphics grants you a non-exclusive
-- license to use, reproduce, modify and distribute this model, provided that:
-- (a) no fee or other consideration is charged for any distribution except
-- compilations distributed in accordance with Section (d) of this license
-- agreement; (b) the comment text embedded in this model is included verbatim
-- in each copy of this model made or distributed by you, whether or not such
-- version is modified; (c) any modified version must include a conspicuous
-- notice that this model has been modified and the date of modification; and
-- (d) any compilations sold by you that include this model must include a
-- conspicuous notice that this model is available from Mentor Graphics in its
-- original form at no charge.
--
-- THIS MODEL IS LICENSED TO YOU "AS IS" AND WITH NO WARRANTIES, EXPRESS OR
-- IMPLIED.  MENTOR GRAPHICS SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF
-- MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  MENTOR GRAPHICS SHALL
-- HAVE NO RESPONSIBILITY FOR ANY DAMAGES WHATSOEVER.
--------------------------------------------------------------------------------
-- File       : BSC032N03S.vhd
-- Author     : Mentor Graphics
-- Created    : 2005-07-20
-- Last update: 2005-07-20
--------------------------------------------------------------------------------
-- Function :
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version     Author              Description
-- 2005-07-20  0.1         David GUIHAL        Created
--------------------------------------------------------------------------------

-- genhdl\schem_bsc032n03s/schem_bsc032n03s.vhd
-- Generated by SystemVision netlister 1.0 build 2005.25.1_SV
-- File created Wed Jul 20 19:26:01 2005

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.electrical_systems.all;
USE ieee.thermal_systems.all;
USE ieee.math_real.all;
LIBRARY edulib;
USE work.all;

entity BSC032N03S is
       generic (    ic_Tj:real:=0.0;
```

```
                        dVth:real:=0.0;
                        dR:real:=0.0;
                        dgfs:real:=0.0;
                        dC:real:=0.0;
                        dZth:real:=0.0;
                        Ls:real:=0.0;
                        Lg:real:=0.0;
                        Ld:real:=0.0);
      port (terminal Tj: thermal; terminal Tcase: thermal; terminal Drain:
electrical; terminal Gate: electrical; terminal Source: electrical);
end entity BSC032N03S;

architecture beh of BSC032N03S is

   terminal TCOUP: THERMAL;
   quantity P_PACKAGE_SOURCE_LOSSES: REAL;

begin

   OPTIMOS2 : entity WORK.OPTIMOS2_30_L3(BEH)
      generic map ( DEV_DATA_DC => DC,
                    DEV_DATA_DGFS => DGFS,
                    DEV_DATA_DR => DR,
                    DEV_DATA_DVTH => DVTH,
                    DEV_DATA_DZTH => DZTH,
                    DEV_DATA_IC_TJ => IC_TJ,
                    MODEL_DATA_ACT => 6.755,
                    MODEL_DATA_CTH1 => 47.022E-6,
                    MODEL_DATA_CTH2 => 270.296E-6,
                    MODEL_DATA_CTH3 => 1.831E-3,
                    MODEL_DATA_CTH4 => 1.19E-3,
                    MODEL_DATA_CTH5 => 30.869E-3,
                    MODEL_DATA_CTH6 => 35.0E-3,
                    MODEL_DATA_CTHB => 1.0E-12,
                    MODEL_DATA_DRTH1 => 1.44E-3,
                    MODEL_DATA_DRTH2 => 19.15E-3,
                    MODEL_DATA_DRTH3 => 64.94E-3,
                    MODEL_DATA_DRTH4 => 300.59E-3,
                    MODEL_DATA_DRTH5 => 445.99E-3,
                    MODEL_DATA_G2 => 1000.0E-3,
                    MODEL_DATA_GMIN => 57.0,
                    MODEL_DATA_INN => 50.0,
                    MODEL_DATA_LD => LD,
                    MODEL_DATA_LG => LG,
                    MODEL_DATA_LS => LS,
                    MODEL_DATA_RD => 50.0E-6,
                    MODEL_DATA_RG => 0.6,
                    MODEL_DATA_RM => 318.0E-6,
                    MODEL_DATA_RMAX => 4.9E-3,
                    MODEL_DATA_RRBOND => 345.0E-9,
                    MODEL_DATA_RRF => 500.0E-3,
                    MODEL_DATA_RS => 400.0E-6,
                    MODEL_DATA_RTB => 28.9E-3,
                    MODEL_DATA_RTH1 => 3.9E-3,
                    MODEL_DATA_RTH2 => 51.76E-3,
                    MODEL_DATA_RTH3 => 186.67E-3,
                    MODEL_DATA_RTH4 => 211.6E-3,
                    MODEL_DATA_RTH5 => 313.96E-3,
```

```
                    MODEL_DATA_UNN => 4.5 )
      port map ( TJ => TJ,
                 TCASE => TCASE,
                 TCOUP => TCOUP,
                 DRAIN => DRAIN,
                 GATE => GATE,
                 SOURCE => SOURCE,
                 P_PACKAGE_SOURCE_LOSSES => P_PACKAGE_SOURCE_LOSSES );

end architecture beh;


--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--------------------------------------------------------------------------------
```

3.2 OptiMOS2_30_L3.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--
-- This model is a component of the Mentor Graphics VHDL-AMS educational open
-- source model library, and is covered by this license agreement. This model,
-- including any updates, modifications, revisions, copies, and documentation
-- are copyrighted works of Mentor Graphics. USE OF THIS MODEL INDICATES YOUR
-- COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH
-- IN THIS LICENSE AGREEMENT.  Mentor Graphics grants you a non-exclusive
-- license to use, reproduce, modify and distribute this model, provided that:
-- (a) no fee or other consideration is charged for any distribution except
-- compilations distributed in accordance with Section (d) of this license
-- agreement; (b) the comment text embedded in this model is included verbatim
-- in each copy of this model made or distributed by you, whether or not such
-- version is modified; (c) any modified version must include a conspicuous
-- notice that this model has been modified and the date of modification; and
-- (d) any compilations sold by you that include this model must include a
-- conspicuous notice that this model is available from Mentor Graphics in its
-- original form at no charge.
--
-- THIS MODEL IS LICENSED TO YOU "AS IS" AND WITH NO WARRANTIES, EXPRESS OR
-- IMPLIED.  MENTOR GRAPHICS SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF
-- MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  MENTOR GRAPHICS SHALL
-- HAVE NO RESPONSIBILITY FOR ANY DAMAGES WHATSOEVER.
--------------------------------------------------------------------------------
-- File       : OptiMOS2_30_L3.vhd
-- Author     : Mentor Graphics
-- Created    : 2005-07-20
-- Last update: 2005-07-20
--------------------------------------------------------------------------------
-- Function :
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version     Author              Description
-- 2005-07-20  0.1         David GUIHAL        Created
--------------------------------------------------------------------------------

-- genhdl\schem_optimos2_30_l3/schem_optimos2_30_l3.vhd
-- Generated by SystemVision netlister 1.0 build 2005.25.1_SV
-- File created Wed Jul 20 18:42:18 2005

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.electrical_systems.all;
USE ieee.math_real.all;
USE ieee.thermal_systems.all;
LIBRARY edulib;
USE work.all;

entity OptiMOS2_30_L3 is
      generic (    model_data_Rs:real;
                        model_data_Rg:real;
                        model_data_Rd:real;
                        model_data_Ls:real;
```

```
                            model_data_Ld:real;
                            model_data_Lg:real;
                            model_data_Rm:real;
                            model_data_Inn:real;
                            model_data_Unn:real;
                            model_data_Rmax:real;
                            model_data_gmin:real;
                            model_data_RRf:real;
                            model_data_Rrbond:real;
                            model_data_Rtb:real;
                            model_data_g2:real;
                            model_data_act:real;
                            model_data_Rth1:real;
                            model_data_Rth2:real;
                            model_data_Rth3:real;
                            model_data_Rth4:real;
                            model_data_Rth5:real;
                            model_data_Cth1:real;
                            model_data_Cth2:real;
                            model_data_Cth3:real;
                            model_data_Cth4:real;
                            model_data_Cth5:real;
                            model_data_Cth6:real;
                            model_data_dRth1:real;
                            model_data_dRth2:real;
                            model_data_dRth3:real;
                            model_data_dRth4:real;
                            model_data_dRth5:real;
                            model_data_Cthb:real;
                            dev_data_ic_Tj:real;
                            dev_data_dVth:real;
                            dev_data_dR:real;
                            dev_data_dgfs:real;
                            dev_data_dC:real;
                            dev_data_dZth:real
                            );
      port (terminal Tj: thermal; terminal Tcase: thermal; terminal Tcoup: thermal;
terminal drain: electrical; terminal gate: electrical; terminal source: electrical;
quantity p_package_source_losses : out real);

end entity OptiMOS2_30_L3;

architecture beh of OptiMOS2_30_L3 is

   terminal T2: THERMAL;
   terminal S1: ELECTRICAL;
   terminal D: ELECTRICAL;
   terminal T4: THERMAL;
   terminal G: ELECTRICAL;
   terminal G1: ELECTRICAL;
   terminal S: ELECTRICAL;
   terminal TB: THERMAL;
   terminal T1: THERMAL;
   terminal D2: ELECTRICAL;

   quantity P_METALLIZATION_LOSSES: REAL;
   quantity P_CHANNEL_LOSSES: REAL;
   quantity I_CGD: CURRENT;
```

```
   quantity I_CDS: CURRENT;
   quantity QDS: REAL;
   quantity P_EPI_LOSSES: REAL;
   quantity P_DIODE_LOSSES: REAL;
   quantity I_DIODE: CURRENT;
   quantity I_CHANNEL: CURRENT;

   quantity Rsource,Rgate,Rb,Pb : real;
   quantity is_q : current;

   quantity vs1s across is1s through s1 to s;
   quantity vg1g across ig1g through g1 to g;

   quantity tc_tb across p_tb through Tb;
   quantity tc_tj across Tj;
   quantity tc_tcase across Tcase;

   constant Zthtype : real := realmin(realmax(dev_data_dZth,0.0),1.0);

begin

   MOS_SFET : entity WORK.MOS_SFET3_30(BEH)
      generic map ( MODEL_A => MODEL_DATA_ACT,
                    MODEL_DC => DEV_DATA_DC,
                    MODEL_DGFS => DEV_DATA_DGFS,
                    MODEL_DR => DEV_DATA_DR,
                    MODEL_DVTH => DEV_DATA_DVTH,
                    MODEL_GMIN => MODEL_DATA_GMIN,
                    MODEL_HEAT => 1.0,
                    MODEL_INN => MODEL_DATA_INN,
                    MODEL_RM => MODEL_DATA_RM,
                    MODEL_RMAX => MODEL_DATA_RMAX,
                    MODEL_RP => MODEL_DATA_RD,
                    MODEL_RS => MODEL_DATA_RS,
                    MODEL_UNN => MODEL_DATA_UNN )
      port map ( TJ => TJ,
                 S0 => S,
                 DD => D,
                 G => G,
                 P_METALLIZATION_LOSSES => P_METALLIZATION_LOSSES,
                 P_CHANNEL_LOSSES => P_CHANNEL_LOSSES,
                 P_DIODE_LOSSES => P_DIODE_LOSSES,
                 P_EPI_LOSSES => P_EPI_LOSSES,
                 I_CHANNEL => I_CHANNEL,
                 I_DIODE => I_DIODE,
                 I_CDS => I_CDS,
                 I_CGD => I_CGD,
                 QDS => QDS );

   ICTH_2 : entity WORK.ICTH_OPTIN3_GN(ARCH1)
      generic map ( CT => MODEL_DATA_CTH2 )
      port map ( TL => THERMAL_REF,
                 TH => T1 );

   ICTH_3 : entity WORK.ICTH_OPTIN3_GN(ARCH1)
      generic map ( CT => MODEL_DATA_CTH3 )
      port map ( TL => THERMAL_REF,
                 TH => T2 );
```

```
ICTH_4 : entity WORK.ICTH_OPTIN3_GN(ARCH1)
   generic map ( CT => MODEL_DATA_CTH4 )
   port map ( TL => THERMAL_REF,
              TH => TCOUP );

ICTH_5 : entity WORK.ICTH_OPTIN3_GN(ARCH1)
   generic map ( CT => MODEL_DATA_CTH5 )
   port map ( TL => THERMAL_REF,
              TH => T4 );

ICTH_6 : entity WORK.ICTH_OPTIN3_GN(ARCH1)
   generic map ( CT => MODEL_DATA_CTH6 )
   port map ( TL => THERMAL_REF,
              TH => TCASE );

RG : entity EDULIB.RESISTOR(IDEAL)
   generic map ( RES => 1.0E3 )
   port map ( P1 => G1,
              P2 => G );

RDA1 : entity EDULIB.RESISTOR(IDEAL)
   generic map ( RES => MODEL_DATA_RD )
   port map ( P1 => D,
              P2 => D2 );

LG : entity EDULIB.INDUCTOR(IDEAL)
   generic map ( IND => MODEL_DATA_LG )
   port map ( P1 => GATE,
              P2 => G1 );

LS : entity EDULIB.INDUCTOR(IDEAL)
   generic map ( IND => MODEL_DATA_LS )
   port map ( P1 => SOURCE,
              P2 => S1 );

LD : entity EDULIB.INDUCTOR(IDEAL)
   generic map ( IND => MODEL_DATA_LD )
   port map ( P1 => DRAIN,
              P2 => D2 );

ICTH_B : entity WORK.ICTH_OPTIN3_GN(ARCH1)
   generic map ( CT => MODEL_DATA_CTHB )
   port map ( TL => THERMAL_REF,
              TH => TB );

IRTH_B : entity WORK.IRTH_OPTIN3_GN(ARCH1)
   generic map ( RT => MODEL_DATA_RTB )
   port map ( TL => TJ,
              TH => TB );

IRTH_1 : entity WORK.IRTH_OPTIN3_GN(ARCH1)
   generic map ( RT => MODEL_DATA_RTH1 +ZTHTYPE*MODEL_DATA_DRTH1 )
   port map ( TL => T1,
              TH => TJ );

IRTH_2 : entity WORK.IRTH_OPTIN3_GN(ARCH1)
   generic map ( RT => MODEL_DATA_RTH2 +ZTHTYPE*MODEL_DATA_DRTH2 )
```

```
      port map ( TL => T2,
                 TH => T1 );

   IRTH_3 : entity WORK.IRTH_OPTIN3_GN(ARCH1)
      generic map ( RT => MODEL_DATA_RTH3 +ZTHTYPE*MODEL_DATA_DRTH3 )
      port map ( TL => TCOUP,
                 TH => T2 );

   IRTH_4 : entity WORK.IRTH_OPTIN3_GN(ARCH1)
      generic map ( RT => MODEL_DATA_RTH4 +ZTHTYPE*MODEL_DATA_DRTH4 )
      port map ( TL => T4,
                 TH => TCOUP );

   IRTH_5 : entity WORK.IRTH_OPTIN3_GN(ARCH1)
      generic map ( RT => MODEL_DATA_RTH5 +ZTHTYPE*MODEL_DATA_DRTH5 )
      port map ( TL => TCASE,
                 TH => T4 );

   ICTH_1 : entity WORK.ICTH_OPTIN3_GN(ARCH1)
      generic map ( CT => MODEL_DATA_CTH1,
                    IC => DEV_DATA_IC_TJ )
      port map ( TL => THERMAL_REF,
                 TH => TJ );

       Rsource==(model_data_Rs-model_data_Rm)*(1.0+(realmin(realmax(tc_Tj,-
200.0),999.0)-25.0)*4.0e-3);
       Rgate==model_data_Rg*(1.0+(realmin(realmax(tc_Tj,-200.0),999.0)-25.0)*1.0e-
3);
       p_package_source_losses==vs1s*vs1s/Rsource;

       is1s==vs1s/Rsource;
       ig1g==vg1g/Rgate;
       is_q==abs(vs1s/Rsource);
       Rb==model_data_Rrbond*(1.0+(realmin(realmax((tc_Tb+tc_Tj)/2.0,-99.0),999.0)-
25.0)*4.0e-3);
       Pb==Rb/(2.0*model_data_Rtb)*(is_q-realmin(realmax((tc_Tj-
tc_Tcase)/(realmax(is_q,1.0e-9)*realmax(Rb,1.0e-
6))+model_data_RRf*is_q*model_data_g2,0.0),is_q))**2.0;
       p_Tb==-Pb;

end architecture beh;


--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--------------------------------------------------------------------------------
```

3.3 mos_sfet3_30.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--
-- This model is a component of the Mentor Graphics VHDL-AMS educational open
-- source model library, and is covered by this license agreement. This model,
-- including any updates, modifications, revisions, copies, and documentation
-- are copyrighted works of Mentor Graphics. USE OF THIS MODEL INDICATES YOUR
-- COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH
-- IN THIS LICENSE AGREEMENT.  Mentor Graphics grants you a non-exclusive
-- license to use, reproduce, modify and distribute this model, provided that:
-- (a) no fee or other consideration is charged for any distribution except
-- compilations distributed in accordance with Section (d) of this license
-- agreement; (b) the comment text embedded in this model is included verbatim
-- in each copy of this model made or distributed by you, whether or not such
-- version is modified; (c) any modified version must include a conspicuous
-- notice that this model has been modified and the date of modification; and
-- (d) any compilations sold by you that include this model must include a
-- conspicuous notice that this model is available from Mentor Graphics in its
-- original form at no charge.
--
-- THIS MODEL IS LICENSED TO YOU "AS IS" AND WITH NO WARRANTIES, EXPRESS OR
-- IMPLIED.  MENTOR GRAPHICS SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF
-- MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  MENTOR GRAPHICS SHALL
-- HAVE NO RESPONSIBILITY FOR ANY DAMAGES WHATSOEVER.
--------------------------------------------------------------------------------
-- File       : mos_sfet3_30.vhd
-- Author     : Mentor Graphics
-- Created    : 2005-07-19
-- Last update: 2005-07-20
--------------------------------------------------------------------------------
-- Function :
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version     Author              Description
-- 2005-07-20  0.1         David GUIHAL        Created
--------------------------------------------------------------------------------

-- genhdl\schem_mos_sfet3_30/schem_mos_sfet3_30.vhd
-- Generated by SystemVision netlister 1.0 build 2005.25.1_SV
-- File created Tue Jul 19 19:59:40 2005

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.electrical_systems.all;
USE ieee.math_real.all;
USE ieee.fluidic_systems.all;
USE ieee.thermal_systems.all;
USE ieee.radiant_systems.all;
LIBRARY edulib;
USE work.all;

entity MOS_SFET3_30 is
      generic (    model_a:real;
                        model_dVth:real:=0.0;
```

```
                        model_dR:real:=0.0;
                        model_dgfs:real:=0.0;
                        model_Inn:real:=1.0e-3;
                        model_Unn:real:=10.0;
                        model_Rmax:real:=1.0;
                        model_gmin:real:=1.0;
                        model_Rs:real:=1.0;
                        model_Rp:real:=1.0e-6;
                        model_dC:real:=0.0;
                        model_Rm:real:=1.0e-6;
                        model_heat:real:=0.0);
      port (terminal Tj: thermal; terminal s0: electrical; terminal dd: electrical;
terminal g: electrical; quantity p_metallization_losses : out real; quantity
p_channel_losses,p_diode_losses,p_epi_losses : out real;   quantity
i_channel,i_diode,i_cds,i_cgd : out current; quantity Qds : out real);
end entity MOS_SFET3_30;

architecture beh of MOS_SFET3_30 is

   terminal D: ELECTRICAL;
   terminal DA: ELECTRICAL;
   terminal S: ELECTRICAL;
   terminal D2A: ELECTRICAL;
   terminal D2: ELECTRICAL;

    quantity Vss0 across s to s0;
      quantity tc across p through Tj;

      constant Fm : real :=0.1;
      constant Fn : real :=0.5;
      constant c : real :=1.08;
      constant Vth0 : real :=2.5;
      constant auth : real :=3.27e-3;
      constant UT : real :=100.0e-3;
      constant ab : real :=40.0e-3;
      constant lB : real :=-23.0;
      constant UB : real :=34.0;

      constant b0 : real :=50.25;
      constant p0 : real :=8.7888;
      constant p1 : real :=-31.4e-3;
      constant p2 : real :=45.0e-6;

      constant Rd : real :=9.6e-3;
      constant nmu : real :=3.3;
      constant Tref : real :=298.0;
      constant T0 : real :=273.0;
      constant lnIsj : real :=-28.44;
      constant ndi : real :=1.0;
      constant Rdi : real :=3.6e-3;
      constant nmu2 : real :=0.0;
      constant ta : real :=3.0e-9;
      constant td : real :=20.0e-9;
      constant Rf : real :=0.7;
      constant nmu3 : real :=1.4;

      constant f1 : real :=239.0e-12;
      constant f2 : real :=479.0e-12;
```

```
	constant f3 : real :=540.0e-12;
	constant U0 : real :=3.0;
	constant nd : real :=0.56;
	constant nc : real :=0.5;
	constant g1 : real :=6.5;
	constant bb : real :=0.5;
	constant kbq : real :=85.8e-6;
	constant remp : real :=5.0e-12;

	constant Vmin : real :=2.1;
	constant Vmax : real :=2.9;
	constant dCmax : real :=0.33;


	--modify parameters if variation parameters are set
	constant Vth : real :=Vth0+(Vmax-Vth0)*realmin(realmax(model_dVth,0.0),1.0)-
(Vmin-Vth0)*realmin(realmax(model_dVth,-1.0),0.0);
	constant q0 : real :=model_a*b0*(T0/Tref)**nmu3;
	constant q1 : real :=(model_Unn-model_Inn*model_Rs-Vth0)*q0;
	constant q2 : real :=(Fm*SQRT(0.4)-c)*model_Inn*q0;
	constant Rlim : real :=(q1+2.0*q2*model_Rmax-SQRT(q1*q1+4.0*q2))/(2.0*q2);


	function function_parameters (
                                                model_a:real;
                                                model_dVth:real;
                                                model_dR:real;
                                                model_Rs:real;
                                                model_Rp:real;
                                                model_dgfs:real;
                                                Rd:real;
                                                Rlim:real;
                                                b0:real)return real_vector is
		variable dRd : real;
		variable bm : real;
		variable bet : real;
		variable parameters_list : real_vector (1 to 3);

	begin

	if (model_dVth=0.0)  then
	dRd:=Rd/model_a+realmin(realmax(model_dR,0.0),1.0)*realmax(Rlim-Rd/model_a-
model_Rs-model_Rp,0.0);
	else
	dRd:=Rd/model_a;
	end if;

	bm :=c/((1.0/model_gmin-model_Rs)**2.0*model_Inn*model_a*(T0/Tref)**nmu3);


	if (model_dR=0.0)  then
                        if (model_dVth=0.0)     then
                                    bet:=b0+(b0-bm)*realmin(realmax(model_dgfs,-
1.0),0.0);

                        else
                              bet:=b0;
                end if;
```

```
      else
                                        bet:=b0;
      end if;
            parameters_list:= (dRd,bm,bet);
      return parameters_list;
      end function function_parameters;

      constant parameters_list : real_vector (1 to 3) := function_parameters(
      model_a,

                                              model_dVth,

                                              model_dR,

                                              model_Rs,

                                              model_Rp,

                                              model_dgfs,

                                              Rd,

                                              Rlim,

                                              b0);

      constant dRd : real := parameters_list(1);
      constant bm : real := parameters_list(2);
      constant bet : real := parameters_list(3);

      constant dC1 : real :=1.0+dCmax*realmin(realmax(model_dC,0.0),1.0);
      constant dC2 : real :=1.0+1.5*dCmax*realmin(realmax(model_dC,0.0),1.0);
      constant Cox : real := f1*model_a*dC2;
      constant Cox1 : real :=remp*SQRT(model_a)*dC2;
      constant Cds0 : real :=f2*model_a*dC1;
      constant Cgs0 : real :=f3*model_a*dC1;
      constant dRdi : real :=Rdi/model_a;

      quantity Imos_Kanal : real;
      quantity Idbr_Kanal : real;
      quantity ploss_Kanal : real;

      quantity ids_diode : real;
      quantity icharge_diode : real;
      quantity Idiode_diode : real;
      quantity ploss_diode : real;

      quantity igd_cgd :real;

      quantity ploss_rdiode :real;
      quantity ploss_rmos :real;

begin

   KANAL : entity WORK.KANAL30(ARCH1)
      generic map ( MODEL_A => MODEL_A,
                    MODEL_AB => AB,
```

```
                  MODEL_AUTH => AUTH,
                  MODEL_BET => BET,
                  MODEL_C => C,
                  MODEL_FM => FM,
                  MODEL_FN => FN,
                  MODEL_KBQ => KBQ,
                  MODEL_LB => LB,
                  MODEL_NMU3 => NMU3,
                  MODEL_P0 => P0,
                  MODEL_P1 => P1,
                  MODEL_P2 => P2,
                  MODEL_T0 => T0,
                  MODEL_TREF => TREF,
                  MODEL_UB => UB,
                  MODEL_UT => UT,
                  MODEL_VTH => VTH )
   port map ( TJ => TJ,
              S => S,
              D => DA,
              G => G ,
              Imos => Imos_Kanal,
              Idbr => Idbr_Kanal,
              ploss => ploss_Kanal);

MET : entity EDULIB.RESISTOR(IDEAL)
   generic map ( RES => MODEL_RM )
   port map ( P1 => S,
              P2 => S0 );

CGS : entity EDULIB.CAPACITOR(IDEAL)
   generic map ( CAP => CGS0 )
   port map ( P1 => G,
              P2 => S );

DIODE : entity WORK.DIODE30(ARCH1)
   generic map ( MODEL_A => MODEL_A,
                 MODEL_CDS0 => CDS0,
                 MODEL_KBQ => KBQ,
                 MODEL_LNISJ => LNISJ,
                 MODEL_ND => ND,
                 MODEL_NDI => NDI,
                 MODEL_TA => TA,
                 MODEL_TD => TD,
                 MODEL_TREF => TREF,
                 MODEL_U0 => U0,
                 MODEL_UB => UB )
   port map ( TJ => TJ,
              S => S,
              D => D2A,
              D1 => DA,
              GRD => ELECTRICAL_REF,
              ids => ids_diode,
              icharge => icharge_diode,
              Idiode => Idiode_diode,
              ploss => ploss_diode );

CGD : entity WORK.CGD30(ARCH1)
   generic map ( MODEL_BB => BB,
```

```
                    MODEL_COX => COX,
                    MODEL_COX1 => COX1,
                    MODEL_G1 => G1,
                    MODEL_NC => NC )
      port map ( D => D,
                 G => G,
                 igd => igd_cgd );

   RDIODE : entity WORK.RDIODE30(ARCH1)
      generic map ( MODEL_DRDI => DRDI,
                    MODEL_HEAT => MODEL_HEAT,
                    MODEL_NMU2 => NMU2,
                    MODEL_TREF => TREF )
      port map ( TJ => TJ,
                 S => S,
                 D2 => D2,
                 D3 => DD,
                 ploss => ploss_rdiode );

   RMOS : entity WORK.RMOS30(ARCH1)
      generic map ( MODEL_DRD => DRD,
                    MODEL_HEAT => MODEL_HEAT,
                    MODEL_NMU => NMU,
                    MODEL_RF => RF,
                    MODEL_TREF => TREF )
      port map ( TJ => TJ,
                 S => S,
                 D => D,
                 DD => DD,
                 ploss => ploss_rmos );

   CHANNEL_SENSE : entity EDULIB.V_CONSTANT(IDEAL)
      generic map ( LEVEL => 0.0 )
      port map ( POS => D,
                 NEG => DA );

   DIODE_SENSE : entity EDULIB.V_CONSTANT(IDEAL)
      generic map ( LEVEL => 0.0 )
      port map ( POS => D2,
                 NEG => D2A );

   R1 : entity EDULIB.RESISTOR(IDEAL)
      generic map ( RES => 2.0E9 )
      port map ( P1 => G,
                 P2 => S );

   RD01 : entity EDULIB.RESISTOR(IDEAL)
      generic map ( RES => 100.0E6 )
      port map ( P1 => D,
                 P2 => S );


       p_metallization_losses==Vss0**2.0/(model_Rm*(1.0+(realmin(realmax(tc,-
200.0),999.0)-25.0)*4.0e-3));
                    p_channel_losses==ploss_kanal;
                    p_diode_losses==ploss_diode;
                    p_epi_losses==ploss_Rmos+ploss_Rdiode;
                    i_channel==imos_kanal+idbr_kanal;
```

```
                    i_diode==idiode_diode;
                    i_cds==ids_diode;
                    i_cgd==-igd_Cgd;
                    Qds==icharge_diode;


                    p == -p_metallization_losses*model_heat;


end architecture beh;


--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--------------------------------------------------------------------------------
```

3.4 kanal30.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--
-- This model is a component of the Mentor Graphics VHDL-AMS educational open
-- source model library, and is covered by this license agreement. This model,
-- including any updates, modifications, revisions, copies, and documentation
-- are copyrighted works of Mentor Graphics. USE OF THIS MODEL INDICATES YOUR
-- COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH
-- IN THIS LICENSE AGREEMENT.  Mentor Graphics grants you a non-exclusive
-- license to use, reproduce, modify and distribute this model, provided that:
-- (a) no fee or other consideration is charged for any distribution except
-- compilations distributed in accordance with Section (d) of this license
-- agreement; (b) the comment text embedded in this model is included verbatim
-- in each copy of this model made or distributed by you, whether or not such
-- version is modified; (c) any modified version must include a conspicuous
-- notice that this model has been modified and the date of modification; and
-- (d) any compilations sold by you that include this model must include a
-- conspicuous notice that this model is available from Mentor Graphics in its
-- original form at no charge.
--
-- THIS MODEL IS LICENSED TO YOU "AS IS" AND WITH NO WARRANTIES, EXPRESS OR
-- IMPLIED.  MENTOR GRAPHICS SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF
-- MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  MENTOR GRAPHICS SHALL
-- HAVE NO RESPONSIBILITY FOR ANY DAMAGES WHATSOEVER.
--------------------------------------------------------------------------------
-- File       : kanal30.vhd
-- Author     : Mentor Graphics
-- Created    : 2005-07-18
-- Last update: 2005-07-18
--------------------------------------------------------------------------------
-- Function :
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version     Author              Description
-- 2005-07-18  0.1         David GUIHAL        Created
--------------------------------------------------------------------------------

library IEEE;
use IEEE.math_real.all;
use IEEE.thermal_systems.all;
use IEEE.electrical_systems.all;

------ Model Interface and Parameters Declaration ------
entity kanal30 is
generic (    model_p0:real;
                  model_p1:real;
                  model_p2:real;
                  model_kbq:real;
                  model_UB:real;
                  model_T0:real;
                  model_Tref:real;
                  model_auth:real;
                  model_Fm:real;
                  model_Fn:real;
```

```
                    model_c:real;
                    model_nmu3:real;
                    model_ab:real;
                    model_UT:real;
                    model_lB:real;
                    model_Vth:real;
                    model_bet:real;
                    model_a:real);
port (terminal Tj: thermal; terminal s: electrical; terminal d: electrical;
terminal g: electrical; quantity Imos,Idbr : out current; quantity ploss : out
heat_flow);
end entity kanal30;

architecture Arch1 of kanal30 is

----------- Branch Variable Declarations ---------
quantity Vds across ids through d to s;
quantity Ugs across g to s;
quantity tc across p through Tj;

------- Time-varying Quantities Declarations-------
quantity values_list : real_vector(1 to 10);
-- This function is used to avoid convergence problems due to numerical overflow --

function limit_exp( x : real ) return real is
    variable abs_x : real := abs(x);
    variable result : real:= 0.0;
begin
  if abs_x < 100.0 then
     result:= exp(abs_x);
  else
     result:= exp(100.0)*(abs_x-99.0);
  end if;
  if x < 0.0 then
  result:= 1.0/result;
  end if;
  return result;
end function limit_exp;

function function_values (
                    model_p0:real;
                    model_p1:real;
                    model_p2:real;
                    model_kbq:real;
                    model_UB:real;
                    model_T0:real;
                    model_Tref:real;
                    model_auth:real;
                    model_Fm:real;
                    model_Fn:real;
                    model_c:real;
                    model_nmu3:real;
                    model_ab:real;
                    model_UT:real;
                    model_lB:real;
                    model_Vth:real;
                    model_bet:real;
                    model_a:real;
```

```
                  vds:real;
                  ugs:real;
                  tc:real) return real_vector is
            variable Tabs : temperature;
            variable Uds : voltage;
            variable phi : real;
            variable Ue : voltage;
            variable psi : real;
            variable b : real;
            --variable Udss : voltage;
            variable I0 : real;
            variable Imos : current;
            variable Idbr : current;
            variable ploss : heat_flow;

            variable values_list : real_vector(1 to 10);

begin

Tabs:=realmin(realmax(tc,-200.0),350.0)+model_T0;
Uds:=realmin(realmax(abs(Vds),0.0),2.0*model_UB);
phi:=(model_p0+(model_p1+model_p2*Tabs)*Tabs)*model_kbq*Tabs;
Ue:=Ugs-model_Vth+model_auth*(Tabs-model_Tref)+model_Fm*Uds**(model_Fn);
psi:=realmin(2.0*phi,phi+model_c*Uds);
b:=model_bet*(model_T0/Tabs)**(model_nmu3);

if Ue>psi then
--Udss==realmin(Uds,Ue/(2.0*model_c));
I0:=(Ue-model_c*(realmin(Uds,Ue/(2.0*model_c))))*(realmin(Uds,Ue/(2.0*model_c)));
else
I0:=phi*(psi-phi)/model_c*limit_exp((Ue-psi)/phi);
end if;

Imos:=model_a*b*I0;

if Vds<0.0 then
Imos:=-Imos;
end if;

Idbr:=model_a*limit_exp(model_lB+(Vds-model_UB-model_ab*(Tabs-
model_Tref))/model_UT);
ploss:=(Imos+Idbr)*Vds;

values_list:=(Tabs,Uds,phi,Ue,psi,b,I0,Imos,Idbr,ploss);

return values_list;
end function function_values;

begin

values_list == function_values(  model_p0,
                                                    model_p1,
                                                    model_p2,
                                                    model_kbq,
                                                    model_UB,
                                                    model_T0,
                                                    model_Tref,
                                                    model_auth,
```

```
                                                    model_Fm,
                                                    model_Fn,
                                                    model_c,
                                                    model_nmu3,
                                                    model_ab,
                                                    model_UT,
                                                    model_lB,
                                                    model_Vth,
                                                    model_bet,
                                                    model_a,
                                                    vds,
                                                    ugs,
                                                    tc);

ids == values_list(8)+values_list(9);   --ids==Imos+Idbr;
p==-values_list(10);                                        --p==-ploss;

Imos == values_list(8);
Idbr == values_list(9);
ploss == values_list(10);

end architecture Arch1;

-------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
-------------------------------------------------------------------------------
```

3.5 diode30.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--
-- This model is a component of the Mentor Graphics VHDL-AMS educational open
-- source model library, and is covered by this license agreement. This model,
-- including any updates, modifications, revisions, copies, and documentation
-- are copyrighted works of Mentor Graphics. USE OF THIS MODEL INDICATES YOUR
-- COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH
-- IN THIS LICENSE AGREEMENT.  Mentor Graphics grants you a non-exclusive
-- license to use, reproduce, modify and distribute this model, provided that:
-- (a) no fee or other consideration is charged for any distribution except
-- compilations distributed in accordance with Section (d) of this license
-- agreement; (b) the comment text embedded in this model is included verbatim
-- in each copy of this model made or distributed by you, whether or not such
-- version is modified; (c) any modified version must include a conspicuous
-- notice that this model has been modified and the date of modification; and
-- (d) any compilations sold by you that include this model must include a
-- conspicuous notice that this model is available from Mentor Graphics in its
-- original form at no charge.
--
-- THIS MODEL IS LICENSED TO YOU "AS IS" AND WITH NO WARRANTIES, EXPRESS OR
-- IMPLIED.  MENTOR GRAPHICS SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF
-- MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  MENTOR GRAPHICS SHALL
-- HAVE NO RESPONSIBILITY FOR ANY DAMAGES WHATSOEVER.
--------------------------------------------------------------------------------
-- File       : diode30.vhd
-- Author     : Mentor Graphics
-- Created    : 2005-07-18
-- Last update: 2005-07-19
--------------------------------------------------------------------------------
-- Function :
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version     Author              Description
-- 2005-07-19  0.1         David GUIHAL        Created
--------------------------------------------------------------------------------

library IEEE;
use IEEE.math_real.all;
use IEEE.thermal_systems.all;
use IEEE.electrical_systems.all;

library EDULIB;

------ Model Interface and Parameters Declaration ------
entity diode30 is
generic (    model_lnIsj:real;
                    model_Tref:real;
                    model_ndi:real;
                    model_kbq:real;
                    model_ta:real;
                    model_td:real;
                    model_Cds0:real;
                    model_U0:real;
```

```
                    model_UB:real;
                    model_nd:real;
                    model_a:real);
port (terminal Tj: thermal; terminal s: electrical; terminal d: electrical;
terminal d1: electrical; terminal grd: electrical; quantity Ids,Icharge,Idiode :
out current; quantity ploss : out heat_flow);
end entity diode30;

architecture Arch1 of diode30 is

terminal a1,b1,c1,e1,edep : electrical;

---------- Branch Variable Declarations ---------
quantity Usd across isd through s to d;
quantity Uds across d1 to s;
quantity tc across p through Tj;
quantity vb1 across ib1 through b1 to grd;
quantity vc1 across ic1 through c1 to grd;
quantity vd1 across id1 through d1 to edep;
quantity ve1 across ie1 through e1 to grd;


------- Time-varying Quantities Declarations-------
quantity values_list : real_vector(1 to 5);

quantity Idyn : current;
quantity Icds : current;

-- This function is used to avoid convergence problems due to numerical overflow --

function limit_exp( x : real ) return real is
    variable abs_x : real := abs(x);
    variable result : real:= 0.0;
begin
  if abs_x < 100.0 then
     result:= exp(abs_x);
  else
     result:= exp(100.0)*(abs_x-99.0);
  end if;
  if x < 0.0 then
  result:= 1.0/result;
  end if;
  return result;
end function limit_exp;

function function_values (
                    model_lnIsj:real;
                    model_Tref:real;
                    model_ndi:real;
                    model_kbq:real;
                    model_ta:real;
                    model_td:real;
                    model_Cds0:real;
                    model_U0:real;
                    model_UB:real;
                    model_nd:real;
                    model_a:real;
                    tc:real;
```

```
                    Usd:real;
                    Uds:real) return real_vector is
            variable Tabs : temperature;
            variable Isjt : voltage;
            variable Idiode : real;
            variable w4 : voltage;
            variable ploss : heat_flow;

            variable values_list : real_vector(1 to 5);

begin

Tabs:=realmin(realmax(tc,-200.0),999.0)+273.0;
Isjt:=realmax(limit_exp(model_lnIsj+(Tabs/model_Tref-
1.0)*1.12/(model_ndi*model_kbq*Tabs))*(Tabs/model_Tref)**1.5,1.0e-10);
Idiode:=model_a*limit_exp(log(realmax(1000.0*Isjt,1.0e-10))-
log(1000.0)+Usd/(model_ndi*model_kbq*Tabs))-Isjt;
ploss:=Usd*Idiode;
w4:=model_Cds0*1.0/(1.0-
model_nd)*model_U0**(model_nd)*(realmin(realmax(model_U0+Uds,0.0),2.0*model_UB))**(
1.0-model_nd);

values_list:=(Tabs,Isjt,Idiode,ploss,w4);

return values_list;
end function function_values;

begin

Cds : entity EDULIB.CAPACITOR(ideal)
      generic map ( cap => model_Cds0)
      port map ( p1 => edep,
                 p2 => s);
L001 : entity EDULIB.INDUCTOR(ideal)
      generic map ( ind => model_ta*model_td/(model_ta+model_td))
      port map ( p1 => a1,
                 p2 => c1);
R001 : entity EDULIB.RESISTOR(ideal)
      generic map ( res => 1.0/model_ta)
      port map ( p1 => a1,
                 p2 => b1);
R002 : entity EDULIB.RESISTOR(ideal)
      generic map ( res => 1.0)
      port map ( p1 => e1,
                 p2 => c1);
R003 : entity EDULIB.RESISTOR(ideal)
      generic map ( res => 500.0e6)
      port map ( p1 => a1,
                 p2 => grd);

values_list == function_values (
                                                    model_lnIsj,
                                                    model_Tref,
                                                    model_ndi,
                                                    model_kbq,
                                                    model_ta,
                                                    model_td,
                                                    model_Cds0,
```

```
                                                    model_U0,
                                                    model_UB,
                                                    model_nd,
                                                    model_a,
                                                    tc,
                                                    Usd,
                                                    Uds);

isd==values_list(3);      --isd==Idiode;
ib1==idyn;
vb1==-values_list(3);           --vb1==-Idiode;
ic1==icharge;
vc1==0.0;
id1==ids;
vd1==Uds-icharge/model_Cds0;
ie1==icds;
ve1==values_list(5);      --ve1==w4;

p==-values_list(3)*Usd;          --p==-Idiode*Usd;

Idiode == values_list(3);
ploss == values_list(4);

end architecture Arch1;

-------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
-------------------------------------------------------------------------------
```

3.6 Cgd30.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--
-- This model is a component of the Mentor Graphics VHDL-AMS educational open
-- source model library, and is covered by this license agreement. This model,
-- including any updates, modifications, revisions, copies, and documentation
-- are copyrighted works of Mentor Graphics. USE OF THIS MODEL INDICATES YOUR
-- COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH
-- IN THIS LICENSE AGREEMENT.  Mentor Graphics grants you a non-exclusive
-- license to use, reproduce, modify and distribute this model, provided that:
-- (a) no fee or other consideration is charged for any distribution except
-- compilations distributed in accordance with Section (d) of this license
-- agreement; (b) the comment text embedded in this model is included verbatim
-- in each copy of this model made or distributed by you, whether or not such
-- version is modified; (c) any modified version must include a conspicuous
-- notice that this model has been modified and the date of modification; and
-- (d) any compilations sold by you that include this model must include a
-- conspicuous notice that this model is available from Mentor Graphics in its
-- original form at no charge.
--
-- THIS MODEL IS LICENSED TO YOU "AS IS" AND WITH NO WARRANTIES, EXPRESS OR
-- IMPLIED.  MENTOR GRAPHICS SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF
-- MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  MENTOR GRAPHICS SHALL
-- HAVE NO RESPONSIBILITY FOR ANY DAMAGES WHATSOEVER.
--------------------------------------------------------------------------------
-- File       : Cgd30.vhd
-- Author     : Mentor Graphics
-- Created    : 2005-07-12
-- Last update: 2005-07-12
--------------------------------------------------------------------------------
-- Function :
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version     Author              Description
-- 2005-07-12  0.1         David GUIHAL        Created
--------------------------------------------------------------------------------

-- VHDL-AMS Model of Cgd30 generated by Paragon
-- This is a machine generated code.
-- Generated on Tue, 12 Jul 2005 11:38:15 AM

library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;

library EDULIB;

------ Model Interface and Parameters Declaration ------
entity Cgd30 is
generic (    model_Cox:real;
                  model_g1:real;
                  model_Cox1:real;
                  model_nc:real;
                  model_bb:real);
```

```
port (terminal d: electrical; terminal g: electrical ; quantity igd : out real);
end entity Cgd30;

architecture Arch1 of Cgd30 is

terminal ox: electrical;

---------- Branch Variable Declarations ---------
quantity w1 across igd1 through d to ox;
quantity Udg across d to g;


begin

-------------- Simultaneous Equations -------------
w1 == realmax(Udg,-model_bb) - (1.0/(model_g1*(1.0-
model_nc))*((1.0/(1.0+model_g1*realmax(Udg+model_bb,0.0)))**(model_nc-1.0)-
(1.0/(1.0+model_g1*model_bb))**(model_nc-1.0)));
igd == model_Cox1*Udg'dot+igd1;

Cgd : entity EDULIB.CAPACITOR(ideal)
      generic map ( cap => model_Cox)
      port map ( p1 => ox,
                 p2 => g);

Cgd2 : entity EDULIB.CAPACITOR(ideal)
      generic map ( cap => model_Cox1)
      port map ( p1 => d,
                 p2 => g);


end architecture Arch1;


--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--------------------------------------------------------------------------------
```

3.7 Rdiode30.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--
-- This model is a component of the Mentor Graphics VHDL-AMS educational open
-- source model library, and is covered by this license agreement. This model,
-- including any updates, modifications, revisions, copies, and documentation
-- are copyrighted works of Mentor Graphics. USE OF THIS MODEL INDICATES YOUR
-- COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH
-- IN THIS LICENSE AGREEMENT.  Mentor Graphics grants you a non-exclusive
-- license to use, reproduce, modify and distribute this model, provided that:
-- (a) no fee or other consideration is charged for any distribution except
-- compilations distributed in accordance with Section (d) of this license
-- agreement; (b) the comment text embedded in this model is included verbatim
-- in each copy of this model made or distributed by you, whether or not such
-- version is modified; (c) any modified version must include a conspicuous
-- notice that this model has been modified and the date of modification; and
-- (d) any compilations sold by you that include this model must include a
-- conspicuous notice that this model is available from Mentor Graphics in its
-- original form at no charge.
--
-- THIS MODEL IS LICENSED TO YOU "AS IS" AND WITH NO WARRANTIES, EXPRESS OR
-- IMPLIED.  MENTOR GRAPHICS SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF
-- MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  MENTOR GRAPHICS SHALL
-- HAVE NO RESPONSIBILITY FOR ANY DAMAGES WHATSOEVER.
--------------------------------------------------------------------------------
-- File       : Cgd30.vhd
-- Author     : Mentor Graphics
-- Created    : 2005-07-12
-- Last update: 2005-07-12
--------------------------------------------------------------------------------
-- Function :
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version     Author              Description
-- 2005-07-12  0.1         David GUIHAL        Created
--------------------------------------------------------------------------------

-- VHDL-AMS Model of Cgd30 generated by Paragon
-- This is a machine generated code.
-- Generated on Tue, 12 Jul 2005 11:38:15 AM

library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;

library EDULIB;

------ Model Interface and Parameters Declaration ------
entity Cgd30 is
generic (    model_Cox:real;
                  model_g1:real;
                  model_Cox1:real;
                  model_nc:real;
                  model_bb:real);
```

```
port (terminal d: electrical; terminal g: electrical ; quantity igd : out real);
end entity Cgd30;

architecture Arch1 of Cgd30 is

terminal ox: electrical;

---------- Branch Variable Declarations ---------
quantity w1 across igd1 through d to ox;
quantity Udg across d to g;


begin

-------------- Simultaneous Equations -------------
w1 == realmax(Udg,-model_bb) - (1.0/(model_g1*(1.0-
model_nc))*((1.0/(1.0+model_g1*realmax(Udg+model_bb,0.0)))**(model_nc-1.0)-
(1.0/(1.0+model_g1*model_bb))**(model_nc-1.0)));
igd == model_Cox1*Udg'dot+igd1;

Cgd : entity EDULIB.CAPACITOR(ideal)
      generic map ( cap => model_Cox)
      port map ( p1 => ox,
                 p2 => g);

Cgd2 : entity EDULIB.CAPACITOR(ideal)
      generic map ( cap => model_Cox1)
      port map ( p1 => d,
                 p2 => g);


end architecture Arch1;


--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--------------------------------------------------------------------------------
```

3.8 rmos30.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--
-- This model is a component of the Mentor Graphics VHDL-AMS educational open
-- source model library, and is covered by this license agreement. This model,
-- including any updates, modifications, revisions, copies, and documentation
-- are copyrighted works of Mentor Graphics. USE OF THIS MODEL INDICATES YOUR
-- COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH
-- IN THIS LICENSE AGREEMENT.  Mentor Graphics grants you a non-exclusive
-- license to use, reproduce, modify and distribute this model, provided that:
-- (a) no fee or other consideration is charged for any distribution except
-- compilations distributed in accordance with Section (d) of this license
-- agreement; (b) the comment text embedded in this model is included verbatim
-- in each copy of this model made or distributed by you, whether or not such
-- version is modified; (c) any modified version must include a conspicuous
-- notice that this model has been modified and the date of modification; and
-- (d) any compilations sold by you that include this model must include a
-- conspicuous notice that this model is available from Mentor Graphics in its
-- original form at no charge.
--
-- THIS MODEL IS LICENSED TO YOU "AS IS" AND WITH NO WARRANTIES, EXPRESS OR
-- IMPLIED.  MENTOR GRAPHICS SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF
-- MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  MENTOR GRAPHICS SHALL
-- HAVE NO RESPONSIBILITY FOR ANY DAMAGES WHATSOEVER.
--------------------------------------------------------------------------------
-- File       : rmos30.vhd
-- Author     : Mentor Graphics
-- Created    : 2005-07-13
-- Last update: 2005-07-13
--------------------------------------------------------------------------------
-- Function :
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version     Author              Description
-- 2005-07-13  0.1         David GUIHAL        Created
--------------------------------------------------------------------------------

library IEEE;
use IEEE.math_real.all;
use IEEE.thermal_systems.all;
use IEEE.electrical_systems.all;

------ Model Interface and Parameters Declaration ------
entity rmos30 is
generic (    model_heat:real;
                   model_dRd:real;
                   model_nmu:real;
                   model_Tref:real;
                   model_Rf:real);
port (terminal Tj: thermal; terminal s: electrical; terminal d: electrical;
terminal dd: electrical; quantity ploss : out heat_flow);
end entity rmos30;

architecture Arch1 of rmos30 is
```

```
---------- Branch Variable Declarations ---------
quantity Vd across dd to d;
quantity i through d to dd;
quantity tc across p through Tj to thermal_ref;

------- Time-varying Quantities Declarations-------
quantity R : real;
quantity Tabs : real;

begin

-------------- Simultaneous Equations -------------
Tabs==realmin(realmax(tc,-200.0),999.0)+273.0;
if not (Tabs>0.0) use
R==model_dRd;
else
R==model_dRd*(model_Rf+(1.0-model_Rf)*(Tabs/model_Tref)**model_nmu);
end use;
ploss*R==Vd*Vd;
i*R==-Vd;
p==(-(ploss*model_heat));

end architecture Arch1;

--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--------------------------------------------------------------------------------
```

4. CAPTEUR OXYGÈNE

```
library ieee;
library edulib;
use ieee.electrical_systems.all;
use ieee.math_real.all;
use work.all;

entity capteur_oxygene is
   generic( Dw : REAL:=2.8e-7;
            DL : REAL:=1.0e-3;
            alpha : REAL:=1.38e11;
            w : REAL:=25.0e-4;
            e : REAL:=70.0e-4;
            s_capteur : REAL:=81.0e-4;
            Surf_Diff : REAL:= 0.25;
            Surf_Membrane : REAL:= 0.98);
   port( terminal OEXT :  ELECTRICAL );
end entity capteur_oxygene;

architecture beh of capteur_oxygene is
   terminal OS: ELECTRICAL;
   terminal OL: ELECTRICAL;
   terminal OM: ELECTRICAL;

   constant q : real := 1.7e-19;

   constant coeff : real := 1.88e13;   -- ppb -> /cm3

   constant RM1 : real :=   (1.0/(q*s_capteur))*(w/Dw)*(1.0/coeff);
   constant RM2 : real :=   (1.0/(q*(Surf_Diff-s_capteur)))*(w/Dw)*(1.0/coeff);
   constant RL1 : real :=   (1.0/(q*s_capteur))*(e/DL)*(1.0/coeff);
   constant RL2 : real :=   (1.0/(q*(Surf_Diff-s_capteur)))*(e/DL)*(1.0/coeff);
   constant RS : real :=    (1.0/(q*alpha*s_capteur));

   constant CLL : real :=    2.25e-3*coeff*q;       --volume mort : 2.25e-3
   constant CLi : real :=    s_capteur*e*coeff*q;

begin

      process
      begin
            report "RM1 = " & real'image(RM1);
            report "RM2 = " & real'image(RM2);
            report "RL1 = " & real'image(RL1);
            report "RL2 = " & real'image(RL2);
            report "RS = " & real'image(RS);
            report "CLL = " & real'image(CLL);
            report "CLi = " & real'image(CLi);
            wait;
      end process;


   RM1_ENT : entity EDULIB.RESISTOR(IDEAL)
      generic map ( RES => RM1 )
      port map ( P1 => OEXT,
```

```
                P2 => OM );

   RM2_ENT : entity EDULIB.RESISTOR(IDEAL)
      generic map ( RES => RM2 )
      port map ( P1 => OEXT,
                 P2 => OL );

   RL1_ENT : entity EDULIB.RESISTOR(IDEAL)
      generic map ( RES => RL1 )
      port map ( P1 => OM,
                 P2 => OS );

   RL2_ENT : entity EDULIB.RESISTOR(IDEAL)
      generic map ( RES => RL2 )
      port map ( P1 => OL,
                 P2 => OS );

   RS_ENT : entity EDULIB.RESISTOR(IDEAL)
      generic map ( RES => RS )
      port map ( P1 => OS,
                 P2 => ELECTRICAL_REF );

   CLL_ENT : entity EDULIB.CAPACITOR(IDEAL)
      generic map ( CAP => CLL )
      port map ( P1 => OL,
                 P2 => ELECTRICAL_REF );

   CLi_ENT : entity EDULIB.CAPACITOR(IDEAL)
      generic map ( CAP => CLi )
      port map ( P1 => OM,
                 P2 => ELECTRICAL_REF );

end architecture beh;
```

5. Fusible

5.1 fuse_dim_v1.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 2005 Mentor Graphics Corporation
--
-- This model is a component of the Mentor Graphics VHDL-AMS educational open
-- source model library, and is covered by this license agreement. This model,
-- including any updates, modifications, revisions, copies, and documentation
-- are copyrighted works of Mentor Graphics. USE OF THIS MODEL INDICATES YOUR
-- COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH
-- IN THIS LICENSE AGREEMENT.  Mentor Graphics grants you a non-exclusive
-- license to use, reproduce, modify and distribute this model, provided that:
-- (a) no fee or other consideration is charged for any distribution except
-- compilations distributed in accordance with Section (d) of this license
-- agreement; (b) the comment text embedded in this model is included verbatim
-- in each copy of this model made or distributed by you, whether or not such
-- version is modified; (c) any modified version must include a conspicuous
-- notice that this model has been modified and the date of modification; and
-- (d) any compilations sold by you that include this model must include a
-- conspicuous notice that this model is available from Mentor Graphics in its
-- original form at no charge.
--
-- THIS MODEL IS LICENSED TO YOU "AS IS" AND WITH NO WARRANTIES, EXPRESS OR
-- IMPLIED.  MENTOR GRAPHICS SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF
-- MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  MENTOR GRAPHICS SHALL
-- HAVE NO RESPONSIBILITY FOR ANY DAMAGES WHATSOEVER.
--------------------------------------------------------------------------------
-- File       : fuse_dim_v1.vhd
-- Author     : Mentor Graphics
-- Created    : 2005-03-23
-- Last update: 2005-03-23
--------------------------------------------------------------------------------
-- Description: MODELE DE FUSIBLE SPECIFIQUE DIMENSIONNEMENT derive du
--                  fusible SABER
--                  parametrage LITTLEFUSE
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version     Author              Description
-- 2005-03-23  1.0         David GUIHAL       Created
--------------------------------------------------------------------------------


library edulib;
library ieee;
use ieee.electrical_systems.all;
use ieee.fluidic_systems.all;
use ieee.mechanical_systems.all;
use ieee.radiant_systems.all;
use ieee.std_logic_1164.all;
use ieee.thermal_systems.all;
use work.all;
use work.fuse_data.all;
```

```
library MGC_AMS;
use MGC_AMS.conversion.all;


entity fuse_dim_v1 is
       generic (    constant state_sw : real := 0.0;
                            constant calibre_fus : real;
                            constant fuse_state : fuse_state_enum := continu;
                            constant temp_fuse : real := 20.0;
                            constant type_fus : fuse_type;
                            constant supplier : fuse_supplier;
                            constant type_dimensionnement : real
                     );
       port (       terminal p,m : electrical
                   --terminal TFUSE: THERMAL
                    );

end entity fuse_dim_v1;

architecture beh of fuse_dim_v1 is

--------------------
--Variables internes
--------------------
constant     calibre : real := 1.5;           -- Minimum current required to blow
the fuse.
constant     rblown : real := 100.0e6; -- Fuse resistance when blown.
constant     rtc : real := 0.0037;            -- Temperature coefficient of fuse
resistance.
constant     temp0 : real := 23.0;            -- Measurement temperature for fuse
parameters (deg K).
constant     tdelay : time := 0 sec;          -- Delay, from the time the fuse
temperature goes
                                                                    -- above tempmelt,
until the fuse blows.

constant     temp_stress : real :=293.0;      -- temperature correpondant a un
stress de 70% en K

----------------------------------------
-- Define the internal thermal connections
----------------------------------------

   terminal TFUSE: THERMAL;
   quantity TFUSE_Quantity across TFUSE;
   terminal SW: ELECTRICAL;
   signal BLOWN: real := 1.0;                -- Assertive state which determines if
fuse
                                                        -- is blown, (equals 2
if blown, 1 if not)
   signal blowtime : real := 0.0;            -- Internal state to monitor time of
fuse blow
   terminal TM: THERMAL;
   terminal TA: THERMAL;

  constant r0 : real       := get_fuse_data(type_dimensionnement, get_r0, supplier,
type_fus, calibre_fus);    -- Fuse resistance measured at temp0.
```

```
  constant tempmelt : real := get_fuse_data(type_dimensionnement, get_tempmelt,
supplier, type_fus, calibre_fus);  -- Melting point of the fuse material (deg C).
  constant rth1 : real     := get_fuse_data(type_dimensionnement, get_rth1,
supplier, type_fus, calibre_fus);
  constant cth1 : real     := get_fuse_data(type_dimensionnement, get_cth1,
supplier, type_fus, calibre_fus);
  constant rth2 : real     := get_fuse_data(type_dimensionnement, get_rth2,
supplier, type_fus, calibre_fus);
  constant cth2 : real     := get_fuse_data(type_dimensionnement, get_cth2,
supplier, type_fus, calibre_fus);
  constant tauth : real     := get_fuse_data(type_dimensionnement, get_tauth,
supplier, type_fus, calibre_fus);


  constant   rblw_r0: real := rblown - r0;                          -- Value of rblown
minus r0.



begin


       Domain_proc : process
       begin
       ----------------------Quiescent_Domain--------------------------------------
              if domain = quiescent_domain then

                report "r0 = " & real'IMAGE(r0);
                report "tempmelt = " & real'IMAGE(tempmelt);
                report "rth1 = " & real'IMAGE(rth1);
                report "cth1 = " & real'IMAGE(cth1);
                report "rth2 = " & real'IMAGE(rth2);
                report "cth2 = " & real'IMAGE(cth2);
                report "tauth = " & real'IMAGE(tauth);

                    if state_sw = 1.0 then
                          blown <= 2.0;
                          report "mode dégradé : fusible : grille au début de la
simulation !";
                    else
                          if fuse_state = reel then
                                if temp_fuse <= tempmelt then
                                      blown <= 1.0;
                                      blowtime <= 0.0;
                                else
                                      blown <= 2.0;
                                      blowtime <= 0.0;
                                      report "Fuse blown at DC, ambient temp. >
tempmelt";
                                end if;
                          elsif fuse_state = continu then
                                if temp_fuse <= tempmelt then
                                      blown <= 1.0;
                                      blowtime <= 0.0;
                                else
                                      blown <= 1.0;
                                      blowtime <= 0.0;
```

```
                                        report "Fuse blown at DC, ambient temp. >
tempmelt";
                                end if;
                            end if;
                    end if;

      ------------------------Time_Domain-----------------------------------------
-
             elsif (domain = time_domain) then
                    loop
                            wait on TFUSE_Quantity'above(tempmelt +
273.15),TFUSE_Quantity'above(temp_stress + 273.15);

                            if TFUSE_Quantity >= (tempmelt + 273.15) and fuse_state =
reel then
                                    blown <= 2.0;
                                    blowtime <= now;
                                    report "Fuse melted at time = " & real'IMAGE(NOW);
                                    exit;
                            elsif TFUSE_Quantity >= (tempmelt + 273.15) and
fuse_state = continu then
                                    blown <= 1.0;
                                    blowtime <= now;
                                    report "Fuse melted at time = " & real'IMAGE(NOW);
                                    exit;
                            end if;

                            if (temp_stress + 273.15) <= TFUSE_Quantity and
TFUSE_Quantity < (tempmelt + 273.15) and fuse_state = continu then
                                    report "Continuous Fuse stressed at time = " &
real'IMAGE(NOW);
                            elsif temp_stress <= TFUSE_Quantity and  TFUSE_Quantity <
tempmelt and fuse_state = reel then
                                    report "Real Fuse stressed at time = " &
real'IMAGE(NOW);
                            end if;

                            if TFUSE_Quantity < (temp_stress + 273.15) and fuse_state
= continu then
                                    report "Continuous Fuse normal at time = " &
real'IMAGE(NOW);
                            elsif TFUSE_Quantity < (temp_stress + 273.15) and
fuse_state = reel then
                                    report "Real Fuse normal at time = " &
real'IMAGE(NOW);
                            end if;

                    end loop;
             end if;
       wait on domain;
       end process;


   R1 : entity EDULIB.RESISTOR(IDEAL)
      generic map ( RES => RBLW_R0 )
      port map ( P1 => P,
                 P2 => SW );
```

```
   CTHERMAL1 : entity EDULIB.CTHERMAL(LINEAR)
      generic map ( CTH => CTH1 )
      port map ( TH1 => TFUSE,
                 TH2 => THERMAL_REF );

   CTHERMAL2 : entity EDULIB.CTHERMAL(LINEAR)
      generic map ( CTH => CTH2 )
      port map ( TH1 => TM,
                 TH2 => THERMAL_REF );

   TEMPCONSTANT1 : entity EDULIB.TEMPCONSTANT(IDEAL)
         generic map ( level => temp_fuse + 273.15)
      port map ( TH1 => TA,
                 TH2 => THERMAL_REF );

   RTHERMAL1 : entity EDULIB.RTHERMAL(LINEAR)
      generic map ( RTH => RTH2 )
      port map ( TH1 => TM,
                 TH2 => TA );

   RTHERMAL2 : entity EDULIB.RTHERMAL(LINEAR)
      generic map ( RTH => RTH1 )
      port map ( TH1 => TFUSE,
                 TH2 => TM );

   R2 : entity EDULIB.RESISTOR(IDEAL)
      generic map ( RES => 100.0E6 )
      port map ( P1 => P,
                 P2 => ELECTRICAL_REF );

   \$1I6\ : entity WORK.SW_1PNC(IDEAL)
      generic map ( ROFF => 1.0E9,
                    RON => 0.0,
                    TDBRK => TDELAY,
                    TDMK => 0 SEC )
      port map ( SW_STATE => BLOWN,
                 P1 => P,
                 P2 => SW );

   R_tc_fuse_Renault : entity WORK.R_TC_FUSE(LINEAR_tauth)
      generic map ( ALPHA => RTC,
                    R_COLD => R0,
                    TEMP_COLD => TEMP0,
                    tauth => tauth,
                    TFUSION => TEMPMELT )
      port map ( P1 => SW,
                 P2 => M,
                 TH1 => TFUSE );

end architecture beh;
```

5.2 r_tc_fuse.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 2001 Mentor Graphics Corporation
--
-- This model is a component of the Mentor Graphics VHDL-AMS educational open
-- source model library, and is covered by this license agreement. This model,
-- including any updates, modifications, revisions, copies, and documentation
-- are copyrighted works of Mentor Graphics. USE OF THIS MODEL INDICATES YOUR
-- COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH
-- IN THIS LICENSE AGREEMENT.  Mentor Graphics grants you a non-exclusive
-- license to use, reproduce, modify and distribute this model, provided that:
-- (a) no fee or other consideration is charged for any distribution except
-- compilations distributed in accordance with Section (d) of this license
-- agreement; (b) the comment text embedded in this model is included verbatim
-- in each copy of this model made or distributed by you, whether or not such
-- version is modified; (c) any modified version must include a conspicuous
-- notice that this model has been modified and the date of modification; and
-- (d) any compilations sold by you that include this model must include a
-- conspicuous notice that this model is available from Mentor Graphics in its
-- original form at no charge.
--
-- THIS MODEL IS LICENSED TO YOU "AS IS" AND WITH NO WARRANTIES, EXPRESS OR
-- IMPLIED.  MENTOR GRAPHICS SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF
-- MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  MENTOR GRAPHICS SHALL
-- HAVE NO RESPONSIBILITY FOR ANY DAMAGES WHATSOEVER.
--------------------------------------------------------------------------------
-- File       : r_tc_fuse.vhd
-- Author     : Mentor Graphics
-- Created    : 2001
-- Last update: 2005/03/24
--------------------------------------------------------------------------------
-- Description: Electrical Resistance Model with Dynamic Thermal
--              Characteristics
--              Valid only for temp > temp_cold + 273.15
--                        Fuse Behavior
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version     Author              Description
-- 2001        1.0         Mentor Graphics     Created
-- 2005/03/24  1.1         David GUIHAL        Architecture linear_tauth added
--                                             matching Renault specifications
--------------------------------------------------------------------------------

-- Use proposed IEEE natures and packages
library IEEE;
use IEEE.THERMAL_SYSTEMS.all;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity r_tc_fuse is

  generic (
    r_cold    : resistance;             -- Electrical resistance at temp_cold
    temp_cold : real      := 27.0;    -- Calibration temperature (deg C)
    tfusion : real := 419.0;                  -- Temp Melt
    tauth : real := 1.0e-3;
    alpha     : real    := 0.0);    -- Linear temperature coefficient
```

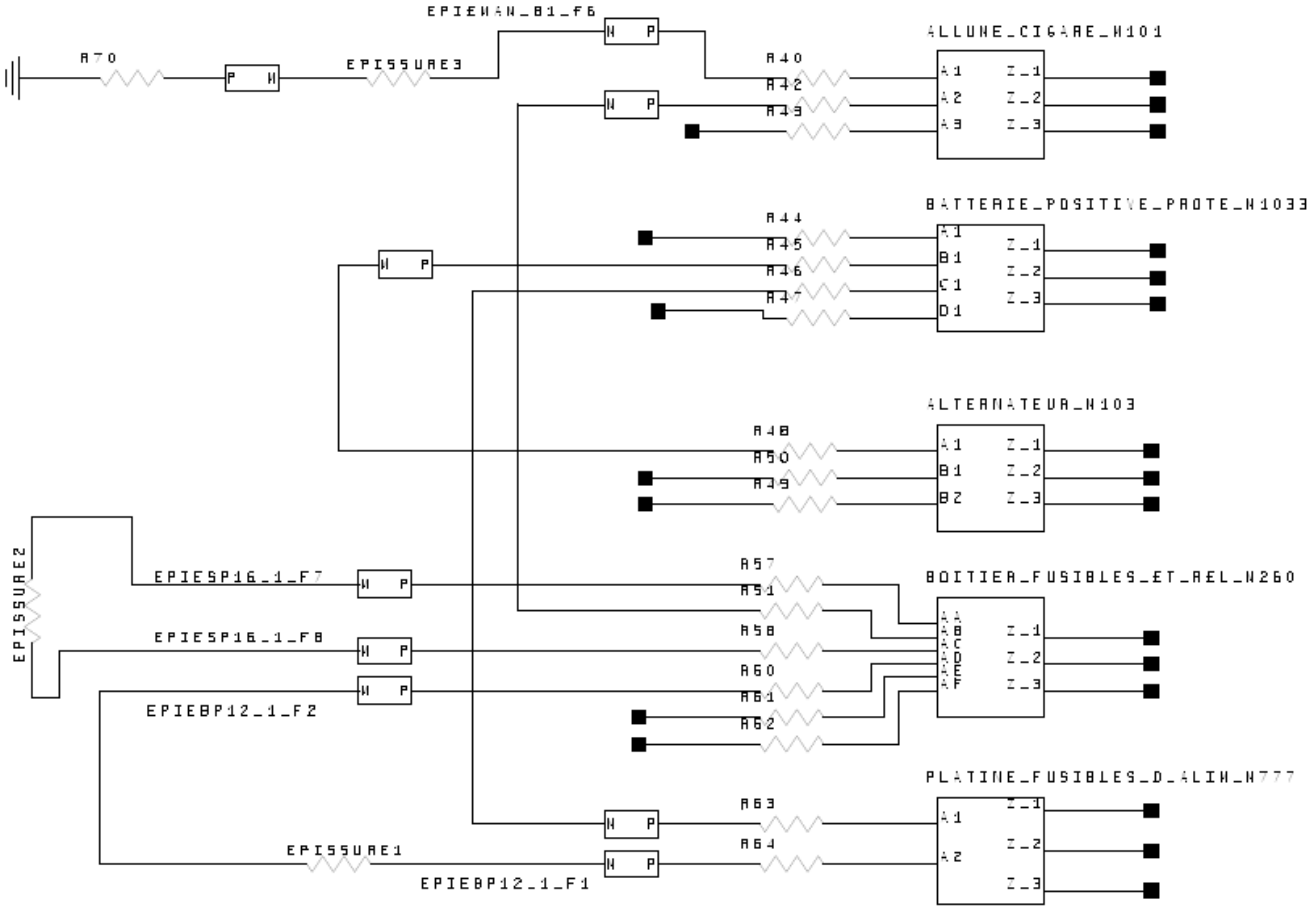
```
  port (
    terminal p1, p2 : electrical;
    terminal th1    : thermal);

end entity r_tc_fuse;

-------------------------------------------------------------------------------
-- Linear Architecture
-- resistance = r_cold*[1.0 + alpha*(temp - (temp_cold+273.15))]
-------------------------------------------------------------------------------
architecture linear of r_tc_fuse is

  quantity v across i through p1 to p2;
  quantity r_temp : resistance;
  quantity r_temp_eff :resistance;
  quantity temp across hflow through th1 to thermal_ref;

  signal r_switch : resistance := 0.0;   -- Resistance of ideal "melt" switch


begin

  r_temp == r_cold*(1.0 + alpha*(temp - (temp_cold + 273.15)));

  if not r_temp'above(1.0e-6) use
       r_temp_eff == 1.0e-6;
  else
       r_temp_eff == r_temp;
  end use;

  v      == i*(r_temp + r_switch'RAMP(1.0e-6, 1.0e-6));
  hflow  == -1.0*v*i;

  melt : process
  begin
    wait until Domain = time_domain;
    wait until temp'ABOVE(tfusion + 273.15);
      r_switch <= 1.0e6;
      break;
      report "Fuse melted at time = " & real'IMAGE(NOW);
  end process melt;

end architecture linear;

architecture linear_tauth of r_tc_fuse is

  quantity v across i through p1 to p2;
  quantity r_temp : resistance;
  quantity r_temp_eff :resistance;
  quantity tj : temperature;
  signal tj_fusion : real := 0.0;
  quantity temp across hflow through th1 to thermal_ref;   --

  signal r_switch : resistance := 0.0;   -- Resistance of ideal "melt" switch


begin
```

```
  r_temp == r_cold*(1.0 + alpha*(tj - (temp_cold + 273.15)));

  if not r_temp'above(1.0e-6)use
       r_temp_eff == r_temp;
  elsif temp'above(tfusion + 273.15) use
       r_temp_eff == r_cold*(1.0 + alpha*(tj_fusion - (temp_cold + 273.15)));
  else
       r_temp_eff == r_temp;
  end use;

 process
 begin
       wait on temp'above(tfusion + 273.15);
       tj_fusion <= tj;
       wait;
 end process;

 break on tj_fusion;

  v      == i*r_temp_eff; -- + r_switch'RAMP(1.0e-6, 1.0e-6));
  hflow  == -1.0*v*i;

  tauth*tj'dot + tj == temp;


  melt : process
  begin
    wait until Domain = time_domain;
    wait until temp'ABOVE(tfusion + 273.15);
      --r_switch <= 1.0e6;
      break;
      report "Fuse melted at time = " & real'IMAGE(NOW);
  end process melt;

end architecture linear_tauth;

--------------------------------------------------------------------------------
-- Copyright (c) 2001 Mentor Graphics Corporation
--------------------------------------------------------------------------------
```

ANNEXE 4 : CARACTERISATION MODELE CAPTEUR

Paramètres	Valeur				
Surface Membrane (cm2)	0,9844				
Surface Puit (cm2)	0,10738729				
Surface Diff. Memb. (cm2)	0,49				
Type Capteur	9,00E-02				
Surface Capteur (cm2)	8,10E-03				
Rayon Volume Mort (cm)	0,375				
Profondeur Volume Mort (cm)	4,00E-03				
Volume Mort (cm3)	2,25E-03				
Calcul Volume Mort (cm3)	2,25E-03				
Epaisseur ss Membrane (cm)	7,00E-03				
Epaisseur Membrane (cm)	2,50E-03				
Coeff Diffusion Membrane (cm2/s)	2,80E-07				
Coeff Diffusion Electrolyte (cm2/s)	1,00E-03				
q	1,70E-19				
Calcul Param. Modèle Lumped					
RM1	6,48E+24				
RM2	1,09E+23				
RL1	5,08E+21				
RL2	8,54E+19				
Cll	3,83E-22				
Cli	9,64E-24				
		Calcul Potentiels			
		Coeff ppb -> /cm3			1,88125E+13
			Concentration Exterieur (ppb)		8210
			Concentration Exterieur (/cm3)	3,45E+11	1,54451E+17
				5,79E+09	
			(RM2+RL2)/(RM1+RL1)	2,70E+08	1,68E-02
				4,54E+06	
			courant	7,20E-09	5,00E-07
			Concentration Niv Capteur (/cm3)	1,81E-10	1,01E+17
			Concentration Niv Capteur (ppb)		5359
			alpha		6,78E+10

ANNEXE 5 : SCHEMATIQUE ALLUME-CIGARE



LISTE DES PUBLICATIONS

- D. GUIHAL, L. ANDRIEUX, D. ESTEVE, "VHDL-AMS model generation from other HDL language: application to a MOSFET switching power device", Rapport LAAS No06437 9th International Forum on Specification and Design Languages (FDL'06), Darmstadt (Allemagne), 19-22 Septembre 2006, pp.77-82, ISBN13 = 978-3-00-019710-9

- D. GUIHAL, L. ANDRIEUX, D. ESTEVE, A. CAZARRE "VHDL-AMS Model Creation", Rapport LAAS No05431 International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES'2006), Gdynia (Pologne), 22-24 Juin 2006, pp.549-554, ISBN10 = 83-922632-1-9

- D. GUIHAL, L. ANDRIEUX, "Génération d'un Modèle VHDL-AMS à partir d'un autre Langage de Description de Matériel", JNRDM 2006 Rennes, 10-12 Mai 2006, ISBN13 = 978-2-9527172-0-5

AUTEUR : GUIHAL David

TITRE : Modélisation en langage VHDL-AMS des Systèmes Pluridisciplinaires

Directeurs de Thèse : Daniel ESTEVE – Laurent ANDRIEUX

Lieu et Date de Soutenance : LAAS-CNRS 25 Mai 2007

RESUME

Ce travail de thèse porte sur la problématique d'élaboration de modèles de systèmes hétérogènes. Il a associé le laboratoire de recherche LAAS-CNRS et la société MENTOR GRAPHICS. Il prend place au sein d'un processus de conception qui se fonde sur les recommandations de l'EIA-632 et sur une ingénierie guidée par les modèles. L'objectif de notre travail est de montrer en quoi le langage VHDL-AMS est adapté à la problématique de modélisation et de simulation de la solution physique au sens des recommandations de l'EIA-632. Dans un premier temps, ce manuscrit présente un état de l'art sur les besoins en modélisation pour la conception système, et dresse un bilan sur les différents langages de modélisation susceptibles d'y répondre. Afin de proposer la norme VHDL-AMS (IEEE 1076.1-1999) comme solution, notre travail s'est attaché à présenter et proposer une méthode à mettre en œuvre pour converger vers cette norme. Notre démarche s'appuie sur l'ingénierie guidée par les modèles avec une place prépondérante jouée par les transformations de modèle. Nous avons développé ce concept de transformation en vue d'une convergence vers le VHDL-AMS : nous développons la notion de meta modèle avec, entre autre, la création d'un meta modèle du langage VHDL-AMS. Celui-ci va permettre une vérification de la conformité des modèles créés, mais aussi l'écriture de règles de transformations au niveau meta modèle. L'intérêt des industriels possédant un existant de modèles écrits dans un langage de description de matériel propriétaire autre (par exemple le langage MAST) en vue d'une migration vers la norme VHDL-AMS, nous a permis d'éprouver cette méthodologie dans de nombreux cas concrets. Nous avons aussi comparé cette approche à une méthodologie que nous avions précédemment définie, nécessitant une expertise dans les deux langages source et cible. Cela nous a permis de conclure positivement sur la faisabilité d'une telle transformation avec une semi-automatisation et une expertise encore nécessaire à certaines étapes. A titre de démonstration, nous avons développé de nombreux modèles mixtes confirmant les aptitudes du VHDL-AMS à pouvoir être le support principal du prototypage virtuel, ainsi que la validité de notre méthode de transformation. Nous avons notamment réalisé la modélisation VHDL-AMS d'un système très hétérogène de mise à feu d'une charge pyrotechnique, qui valide notre méthodologie. La validation des modèles en conformité avec les spécifications est une des perspectives identifiées de nos travaux, à approfondir.

Mots clés : Modélisation de Systèmes hétérogènes, langage VHDL-AMS, Meta Modèle, Transformation de modèles, EIA-632.

ABSTRACT

This PhD work is focusing on heterogeneous system modeling issues with the involvements from the French research laboratory LAAS-CNRS and the company Mentor Graphics. The work is part of a well defined design process based on EIA-632 recommendations and on model driven engineering. The objective is to demonstrate how behavioral language such as VHDL-AMS can answer to the modeling and the simulation problems related to the physical design under EIA-632 recommendations. The thesis starts from the state of the art of system design modeling requirements, and follows by evaluation of different modeling languages. VHDL-AMS standard (IEEE 1076.1-1999) has been selected for being best physical solution available. In order to migrate from proprietary standards to VHDL-AMS, our work focuses on defining appropriate methodology. Our process flow is based on the model driven engineering where model transformation concept is being used. We use the transformation concept to converging to VHDL-AMS code. We also use the meta model concept to create the VHDL-AMS meta model which will be useful for checking the conformity and for defining transformation rules at the meta model level.

Companies that are looking for solutions to migrating from proprietary models such as MAST to VHDL-AMS provide real test cases to our transformation methodology. We also compare such approach to a preliminary methodology based on using modeling expertise in both source and target languages. This highlights the feasibility of such transformation with a possible semi-automation and with experts involved at particular steps. Several models have been created to validate VHDL-AMS as virtual prototyping modeling language. In particular, a model for heterogeneous firing system of a pyrotechnic charge dedicated to an onboard missile location has been developed from specifications. It is an important device for our perspective projects.

Keywords : Heterogeneous Systems modeling, VHDL-AMS language, Meta Model, Model Transformation, EIA-632.