Méthodes et outils de la conception amont pour les systèmes et les microsystèmes

Juan Carlos Hamon

Année 2005
N° d'ordre_____

THÈSE

Présentée pour obtenir le titre de :
Docteur de l'Institut National Polytechnique de Toulouse.

Ecole doctorale :
Génie Electrique, Electronique, Télécommunications

Spécialité :
Conception de Circuits micro électroniques et micro systèmes.

Par :
M Juan-Carlos HAMON

Méthodes et outils de la conception amont pour les systèmes et les microsystèmes

Soutenue le 1 février 2005, devant le jury :

Directeur de thèse :	Daniel ESTEVE *Directeur de Recherche LAAS-CNRS*
Rapporteurs :	*Yannick HERVE* *Lionel TORRES* *Alain VACHOUX*
Examinateurs :	*Bernard BERTHOMIEU* *Mario PALUDETTO*
Invités :	*Tom KAZMIERSKI* *Pascal PAMPAGNIN*

Remerciements

Les travaux présentés dans ce mémoire ont été réalisés dans le cadre d'une collaboration entre Airbus France, le laboratoire d'électronique de l'ENSEEIHT (LEN7) et le Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS) au sein du groupe Microsystème et Intégration de Systèmes (MIS). Je tiens tout d'abord à remercier les directeurs successifs du LAAS, Monsieur Jean-Claude Laprie et Monsieur Malik Ghallab pour leur accueil ainsi que le Directeur du LEN7, Monsieur Thierry Bosch.

Je tiens à remercier Madame Anne-Marie GUE, responsable du groupe MIS, de m'y avoir accueilli pendant ces années de thèse.

J'adresse toute ma gratitude à mon directeur de thèse, Monsieur Daniel Estève, pour sa collaboration inestimable, sa disponibilité et les nombreuses conversations et conseils qui m'ont aidé et orienté dans mon travail. Je voudrais le remercier aussi pour la confiance qu'il a marquée à mon égard et pour m'avoir permis d'entreprendre avec lui plusieurs projets scientifiques qui ont enrichi mon travail et qui m'ont permis d'ouvrir plusieurs terrains de discussion pendant ces dernières années.

Pour réaliser mon travail, j'ai bénéficié d'un contact étroit avec l'équipe «Avionics and Simulation Products » du département d'électronique de la Société Airbus France dans le choix et le suivi des orientations scientifiques et techniques et dans la sélection des exemples de démonstration. Je remercie tous ceux qui sont intervenus dans cette relation, en particulier Monsieur Pascal Pampagnin qui a été mon guide et mon correspondant permanent au niveau industriel. Merci aussi à monsieur Jean-Michel Murawski pour sa disponibilité, son intérêt envers le projet et pour les nombreuses discussions techniques.

Je suis très reconnaissant de l'honneur que m'ont fait Monsieur Yannick Hervé, Monsieur Lionel Torres et Monsieur Alain Vachoux en acceptant la tâche d'évaluer en qualité de rapporteurs les travaux présentés. J'exprime également ma reconnaissance à Monsieur Mario Paludetto, Monsieur Bernard Berthomieu, Monsieur Tom Kazmierski et Monsieur Pascal Pampagnin, pour avoir accepté de prendre part au jury. Je les remercie tous pour l'intérêt qu'ils ont porté à ces travaux.

Je tiens à exprimer ma gratitude à mes collègues Fernando Garcia de los Rios, Hernan Duarte, David Guihal et Julien Baylac ainsi qu'aux partenaires du projet MOCAS-LAAS pour leur collaboration envers mes travaux de thèse.

Durant ces années passées à Toulouse, j'ai pu apprécier l'amitié de certaines personnes qui ont partagé avec moi des moments très importants de ma vie, un grand merci à Benoît, Alain et Emmanuelle. Je souhaiterais aussi remercier mes collègues du groupe MIS, en particulier Petra et Rémy pour avoir été toujours prêts à discuter et travailler avec moi sur la conception système. A mes nombreux colocataires de bureau Ty, Gustavo, Pierre, David, Sylvaine…

Mes remerciements à tout le personnel technique et administratif du LAAS, avec une attention particulière envers Madame Nicole Higounet pour son amabilité et son efficace collaboration dans l'organisation des multiples réunions et séminaires TOOLSYS. Merci aussi à Monsieur Christian Berty pour sa disponibilité et sa collaboration toujours opportune.

Je voudrais remercier la personne qui a partagé de plus près cette aventure avec moi, celle qui a été ma motivation permanente, celle avec qui j'ai vécu les meilleures années de ma vie, ma chère Tati : Merci pour ton amour, ta patience, ta compréhension et ton soutien sans égal.
Finalmente quiero agradecer a mis Padres y a mis Hermanos quienes siempre han creído en mi y han apoyado incondicionalmente todas mis decisiones. A pesar de los miles de kilómetros de distancia siempre han estado a mi lado. Si hoy en día puedo escribir estas líneas es gracias a ellos.

Table des matières

Liste des figures

Liste des tableaux

INTRODUCTION GENERALE

Les concepteurs de systèmes modernes doivent gérer des projets associant plusieurs disciplines et plusieurs technologies. En particulier, depuis des années, les systèmes électroniques ne sont plus conçus isolément : Ils intègrent des préoccupations de systèmes et de micro-systèmes multidisciplinaires, dans divers secteurs d'applications scientifiques et industrielles. En raison de la complexité et de l'hétérogénéité de ces systèmes, il est nécessaire de mettre en place des **méthodes et des outils facilitant l'intégration de solutions analogiques, numériques, mixtes, matérielles et logicielles dans des approches pluridisciplinaires telles que la micro-mécanique, la micro-fluidique ou les systèmes électro-optiques**. Ce problème pluridisciplinaire et le besoin d'optimiser le processus de conception pour réduire le "time to market", a conduit au développement de techniques telles que la modélisation et la validation à haut-niveau, la modélisation fonctionnelle, la réutilisation et la génération de modules de propriété intellectuelle (IP)… Ces composantes doivent être considérées dès les premières étapes de la conception. Nos propositions auront à tenir compte de nombreuses exigences, et donc à **s'appuyer sur des langages et des procédures standardisés**.

Les méthodes actuelles doivent aussi **prendre en compte la conception coopérative et la réutilisation des acquis.** Elles doivent donc être basées sur des procédures et des langages normalisés ou standardisés facilitant les échanges. Le but est alors de proposer des outils généraux et des méthodes capables de soutenir le travail coopératif entre divers participants d'un projet de conception. Dans une première étape, les méthodes utilisées doivent s'appuyer sur des modèles de haut-niveau, fonctionnels, exécutables, que nous appelons ici des **Prototypes Virtuels**. Ces prototypes permettent de vérifier, par simulation, leur conformité fondamentale avec le cahier des charges, avant d'entamer les démarches de matérialisation et de réalisation technologique.

Dans le contexte de la conception système et de la réutilisation des savoir-faire, le codesign définit un domaine nouveau et prometteur dans le développement d'outils de Conception Assistée par Ordinateur (CAO). Ses objectifs principaux sont de faire, de façon simultanée, **la conception du matériel et du logiciel** afin de réduire les temps de développement et d'élaboration, de faciliter la détection des fautes et des erreurs, et de permettre le test des prototypes virtuels avant la mise en fabrication des produits. Nous devrons trouver et définir quelle est la place précise du « co-design » dans notre démarche méthodologique.

L'intégration pluridisciplinaire autour de projets nécessite d'adopter des normes et des méthodes dans la définition des spécifications aussi bien que dans la description et la mise en place

des modèles créés à partir de telles spécifications. Les concepteurs sont également obligés de trouver des solutions aux défis techniques posés par les cahiers des charges dans des créneaux de temps de plus en plus serrés. En résumé, **la complexité croissante des systèmes et les contraintes liées aux délais imposent la définition de nouvelles méthodes et de nouveaux outils capables de répondre simultanément aux besoins des concepteurs et à la coordination de tous les acteurs du problème**. La Figure 0-1 met en avant ces principaux acteurs.

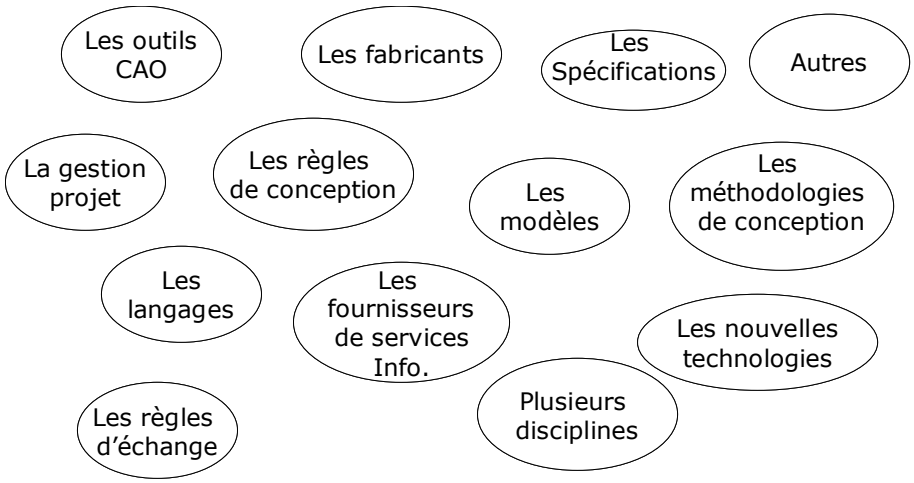


Figure 0-1. Les parties prenantes de la conception système.

Les motivations techniques, commerciales, et l'influence de la concurrence créent des intérêts forts chez les industriels, dans le développement, l'utilisation et l'optimisation de technologies permettant d'arriver aussi loin que possible dans les extensions de la conception système [Ham01]. Le développement des prototypes virtuels et la nécessité de valider leur cohérence avec les spécifications, demande l'appui d'outils informatiques permettant de modéliser, dès les niveaux d'abstraction les plus élevés, les aspects suivants:

- Les interactions du système avec son environnement opérationnel,
- L'évaluation et la définition des entrées / sorties,
- L'étude et le développement des modèles comportementaux des constituants,
- La représentation graphique des relations fonctionnelles proposées,
- L'exploration architecturale,
- L'estimation des performances et les états critiques de fonctionnement.

Les exigences générales se situent au niveau de la gestion de ces aspects et de la recherche de techniques permettant de réduire le temps de développement des produits et d'accroître les performances de la conception sur des points essentiels comme la robustesse, la sûreté de fonctionnement et la vérification. Plusieurs axes de réflexion et de développement devront être explorés :

- La réutilisation, autant que possible, des acquis et des modèles précédemment validés. Dans ce contexte, l'extension de l'utilisation des outils de CAO pour l'électronique vers d'autres domaines peut représenter une source de progrès important.
- L'utilisation de techniques telles que le codesign, la co-simulation, la création et la gestion de modules de propriété intellectuelle ont amené le monde de l'électronique

numérique au sommet de ce qu'il est convenu d'appeler l'EDA (Electronic Design Automation) jusque dans les applications commerciales.

- La partie analogique qui pourtant reste en retard par rapport à ces techniques à cause de la complexité du problème et de la difficulté à mettre en place une vraie politique de standardisation des méthodes, et des langages **:** Il n'est pas facile d'y établir une base commune de ressources informatiques et méthodologiques, à l'image de la modélisation VHDL ou de la gestion des IPs dans le domaine de l'électronique numérique.

La plus grande difficulté de l'approche est de généraliser les méthodes et de considérer la conception système comme un tout. De cette manière, en partant des spécifications ou des cahiers des charges, les concepteurs pourraient établir et valider des modèles fonctionnels et proposer des solutions architecturales. Une étape essentielle de cette problématique est la traduction ou l'interprétation des spécifications sous forme de modèles fonctionnels.

Nous pouvons illustrer globalement cette problématique et le contexte de nos activités par le biais de la figure suivante :

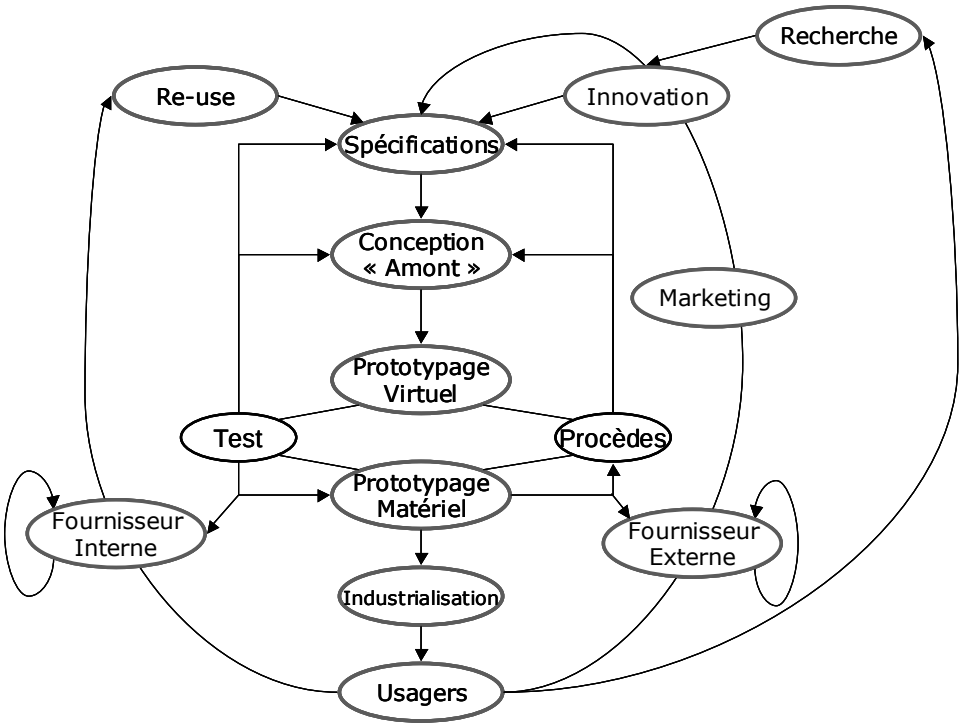


Figure 0-2. Synthèse générale de la problématique et du contexte de notre travail.

Dans ce contexte, **notre approche concerne la mise au point d'une méthodologie de « Haut-niveau » de développement de modèles fonctionnels exécutables permettant aux concepteurs d'aborder les problèmes du choix des technologies et des architectures, des partitionnements analogique/numérique d'un part et logiciel/matériel d'autre part, sur la base sûre d'une proposition structurellement fiable et cohérente avec les spécifications**. Cette méthodologie intervient avant l'établissement des choix technologiques, il n'y a pas donc lieu, à ce stade, de distinguer les micro systèmes et les systèmes. Nous parlerons, pour simplifier le discours, uniquement des systèmes dans les développements suivants. La proposition d'une méthodologie pour la conception de haut-niveau devra rechercher et prendre en compte les langages standards utilisables aussi bien pour la description système que pour les descriptions de niveaux inférieurs

d'abstraction. L'utilisation de normes et de standards facilitera l'application d'une méthodologie dans les cas réels, en évitant qu'elle reste isolée comme une proposition seulement théorique.

L'objectif de notre travail est de concevoir, de développer et de mettre en oeuvre les fondements d'un outil de co-design pris au sens général : Matériel / logiciel / pluridisciplinaire, permettant d'appliquer une méthodologie de conception descendante dans le processus de conception système. Nous aurons à tester cet outil par le biais d'exemples concrets de gestion du « codesign » dans le processus de conception : intégration de la partie analogique et arbitrages matériels / logiciels.

Une fois cette méthodologie et cet outil établis, nous analyserons quel est l'impact de leur mise en place sur la vérification « système » que nous pouvons envisager à partir du prototype virtuel. Pour cela, nous nous appuierons sur la gestion standardisée de l'information vis-à-vis de la réutilisation des savoir-faire et la génération de modules de propriété intellectuelle IP. Finalement, nous réfléchirons à une première approche vers la vérification et la validation des modèles et des prototypes, notamment dans le domaine de l'avionique.

Nos travaux de recherche ont été effectués dans le cadre du groupe MIS (Microsystèmes et Intégration des Systèmes) du LAAS dont la mission est de mettre en place des méthodes et des outils pour la conception des micro systèmes. Ils se sont développés en une collaboration entre AIRBUS France, le LEN7-ENSEEIHT-INPT et le LAAS-CNRS. Ils s'inscrivent dans une activité **conception de systèmes** mixtes à base d'électronique, dans une approche descendante vers la création de **prototypes virtuels**. Notre ligne de conduite est de proposer des outils et des méthodes capables de favoriser le **travail coopératif** entre les divers intervenants d'un projet de conception système, au stade **de la conception amont**, avant d'établir les dépendances technologiques jusqu'à établir des solutions architecturales. Le résultat de cette initiative sera l'élaboration de modèles fonctionnels exécutables permettant de vérifier, par simulation, la conformité des prototypes avec les spécifications du cahier des charges.

Ce mémoire de thèse comprend quatre chapitres :

Dans le chapitre 1 nous réalisons un bilan des outils et des méthodes existantes pour la conception système. Les outils de CAO et les méthodes de conception sont en constante évolution et leur développement reflète l'état de l'art des pratiques. Les nouveautés dans le domaine des méthodes et outils de conception de haut-niveau sont mises en œuvre par le biais d'outils logiciels permettant de les appliquer couramment. Au niveau de la conception des systèmes électroniques, une fois que les différentes approches et méthodes dépassent le niveau purement théorique, et que les langages sont adoptés, des processus d'implémentation logicielle peuvent être engagés. Ces étapes expérimentales conduisent à la réalisation d'outils d'aide à la conception qui reflètent l'état de l'art en la matière. Pendant toute la durée de notre étude, nous sommes restés attentifs aux nouveautés provenant des fournisseurs commerciaux, et aux innovations proposées par le monde académique et scientifique. En fonction de ce tour d'horizon nous, présenterons la problématique de nos travaux.

Le chapitre 2 illustre les questions qui se posent au niveau des spécifications et l'approche méthodologique proposée par le formalisme de description de systèmes à haut-niveau HiLeS. Nous consacrons un chapitre entier à ce formalisme, car nous en avons précisé et corrigé la version originale avant d'engager la mise en place d'un outil logiciel. En deuxième partie, nous présentons la mise en œuvre du formalisme HiLeS sous la forme d'un outil logiciel : HiLeS Designer 0. Nous

décrivons ses applications potentielles et nous précisons la façon dont les utilisateurs peuvent résoudre leurs problèmes spécifiques grâce à l'application de notre démarche.

Le chapitre 3 porte sur la mise en place d'une plate-forme de conception système à haut niveau portant sur l'intégration de HiLeS Designer avec un outil de vérification formel des réseaux de Petri (TINA) et une sortie terminale vers le langage VHDL-AMS. Avant la finalisation de l'élaboration du modèle « Haut-niveau », nous explorons les techniques possibles pour sa validation. En effet, la vérification de la cohérence du modèle avec les spécifications est primordiale avant de poursuivre la conception à des niveaux moins conceptuels et plus matériels.

Nous terminons ce mémoire par un chapitre 4 consacré à l'illustration par des exemples d'utilisation du formalisme HiLeS, notamment de l'outil logiciel HiLeS Designer. Le but de cette dernière partie est de traiter complètement un cas d'application en utilisant notre approche afin d'illustrer, de la meilleure façon possible, son fonctionnement. Bien sûr, nous tenterons de dégager de ces premiers traitements, les limites de notre approche, ses perspectives et la façon d'aborder des étapes nouvelles de recherche et développement.

CHAPITRE 1

1. ETAT DE L'ART DE LA CONCEPTION DE SYSTEMES A BASE D'ELECTRONIQUE.

1.1 Introduction

Les outils de CAO sont devenus des appuis incontournables pour les ingénieurs et les scientifiques, au moment d'exécuter tous types de projets, particulièrement ceux dont la complexité et le temps de développement sont importants. En accord avec l'importance de la CAO dans notre sujet, nous avons conduit, pendant toute la durée de la thèse, une veille technologique des outils disponibles sur le marché et des techniques émergentes des laboratoires de recherche et autres organismes universitaires. Malheureusement, nous n'avons pas eu la possibilité de tout tester et nos analyses ne pourront qu'être fortement influencées par les standards et les tendances d'utilisation au niveau de l'industrie et de la recherche.

Notre intérêt porte sur la recherche de méthodes et d'outils pour intervenir dans les étapes initiales de la conception. Nos travaux de recherche visent plus particulièrement, la création de modèles de haut-niveau permettant d'initier une démarche de conception descendante, c'est à dire, de réaliser la liaison entre les spécifications et les modèles les plus en amont du processus de conception. Pour cette raison, nous ne nous étendrons pas sur les outils utilisés au niveau physique des composants comme les outils « propriétaires » de SILVACO et les autres outils spécialisés dans le routage, tels que PCB d'OrCAD, pas plus que ceux dédiés à la simulation physique à base d'éléments finis. Pour ce chapitre d'analyse et de synthèse de l'existant, nous avons classé les outils selon trois champs d'application : la conception électronique en général, la conception de haut-niveau des systèmes à base d'électronique et la gestion de l'information des projets de conception système.

1.2 La problématique de la conception système

Aujourd'hui, concevoir un nouveau produit revient à imaginer la description, à l'aide d'outils informatiques, d'une réalisation matérielle possible de ce produit à partir de ses spécifications textuelles.

La qualité de conception d'un produit est caractérisée par le niveau des performances que l'on tente de fournir et de garantir : fonctionnalités, sûreté de fonctionnement, consommation énergétique, dimensionnements…, ou des performances que l'on évalue par anticipation : la fiabilité par exemple.

Nous comprenons, compte tenu de la multiplicité des performances et des critères, que des solutions nombreuses peuvent être trouvées pour chaque produit, d'autant plus nombreuses que les contraintes sont lâches.

Les exigences de conformité aux besoins du marché et le resserrement des objectifs « cible » réduisent considérablement cette variabilité des solutions possibles.

D'autres facteurs sont réducteurs. En particulier, la réutilisation de sous-ensembles déjà conçus et validés. On en comprend l'intérêt dans la réduction des délais de conception, dans la sûreté de conception, dans la mesure où le sous-ensemble a déjà été validé (expérimenté) et sauvegardé en tant que module de propriété intellectuelle (IP). Il y a aussi des inconvénients évidents dans « l'alourdissement » du produit où l'on doit rechercher des optimisations globales en même temps que la compatibilité indispensable du sous-ensemble réutilisé avec les autres contraintes du produit.

1.2.1 Le chemin de la conception descendante

Tout produit a un « cœur fonctionnel » de base ou une « technologie de base » qui détermine son originalité et sa valeur. Cela a conduit historiquement à une démarche apparemment naturelle de type « bottom-up » où il s'agit de construire, autour de cet élément central, les autres fonctionnalités ou autres technologies permettant de le valoriser. Cette pratique aurait toute sa valeur dans un contexte ou la bonne idée, originale et efficace, peut faire la différence. **Actuellement, le concepteur dispose de toute une quantité de composants et de technologies qui tendent à privilégier une « innovation de conception » faite de bons choix en terme de fonctions, de technologies d'interfaçage ou d'assemblage.**

Ainsi, cette évidence privilégie une démarche « Top-down », partant des spécifications, des composants et des technologies disponibles pour proposer le produit le plus adapté possible. Il n'est pas très utile de rentrer dans le débat trop philosophique sur les approches « Bottom-Up » et « Top-Down ». Pour être constructif, il vaut mieux parler de « Meet in the Middle » qui allie l'avantage des deux dans une démarche globalement descendante comportant des boucles d'ascendance.

Une démarche descendante va partir des spécifications pour arriver, au terme du processus de conception descente, à une représentation informatisée du produit visé que nous appelons le « prototype virtuel ». Cette représentation doit être complète pour donner accès, par le calcul, à toutes les performances que nous avons déjà énoncées : fonctionnalités, sûreté de fonctionnement, influences environnementales, fiabilité… Cela implique :

- que tous les constituants du produit soient représentés par des modèles précis,
- que les paramètres de ces modèles soient reliés aux facteurs de définition ou d'influence pour permettre les calculs prévisionnels.

Pour cela, il faut que les technologies et les fonctionnements soient parfaitement maîtrisées pour que puissent être communiqués les modèles conformes aux fournitures pressenties.

Nous distinguons ainsi deux niveaux de conception parfaitement communicants :

- la conception amont, qui apportera une représentation fonctionnelle puis architecturale du « système-produit ».
- le prototypage virtuel qui apporte une représentation virtuelle d'une réalité du produit potentiel.

Ceci détermine trois interfaces ou étapes de transformation :

- l'interface pour le passage des spécifications (cahier des charges) à la conception amont.
- l'interface entre la conception amont et le prototypage virtuel
- l'interface prototypage virtuel – conception produit. Celle-ci peut comporter d'autres étapes intermédiaires.

Pour accélérer le processus du prototypage virtuel vers la matérialisation, il faut que le descriptif comporte, de manière précise, la représentation matérielle de chaque constituant et des interfaces physiques pour en dégager une architecture concrète et des modes de connexion. Une représentation matérielle globale peut alors être construite. Cette représentation peut être mise au point par la simulation de toutes les performances influençables par la démarche de matérialisation, d'intégration ou d'assemblage : simulation électronique, thermique, mécanique, etc. Sur ces représentations, pourront être greffés des calculs estimatifs de fiabilité et d'évolution temporelle sous contraintes diverses, si les représentations intègrent bien les effets d'influence environnementaux et fonctionnels.

1.2.2 Historique et tendances

Etant donné l'importance de l'aspect informatique de notre démarche, il est tout naturel de faire le point sur la place des outils logiciels dans la conception système. Historiquement, nous pouvons constater, dans l'évolution chronologique des outils informatiques, que ceux-ci participent de plus au processus de conception système. D'une simple aide complémentaire à la fin des années quatre-vingts, nous sommes arrivés à des supports de modélisations les plus complexes de projets complets.

La Figure 1-1, fait apparaître les conclusions [Tho02] présentées sur ce sujet lors du dernier forum ENHANCE [Enh02] à Toulouse. Cette analyse porte sur la conception aéronautique, mais la réflexion est applicable à une démarche générale pluridisciplinaire telle que la nôtre. Elle indique, par les pointilles, que nous sommes en pleine transition entre le « maquetage numérique » et un véritable « prototypage virtuel » tel que nous le présentons dans ce travail.

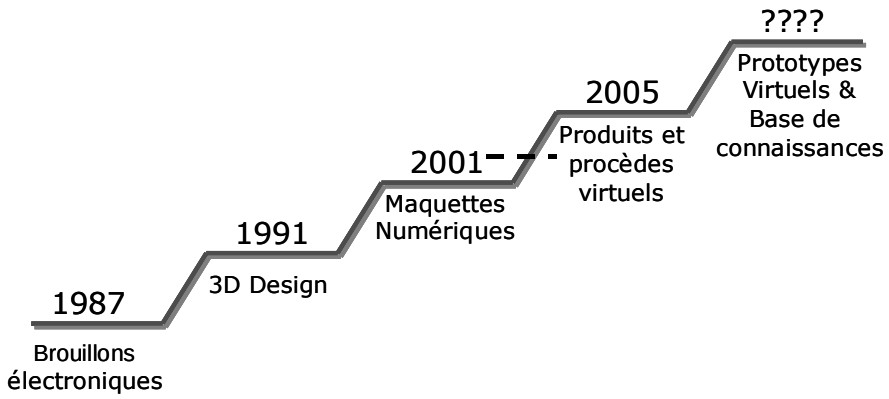


Figure 1-1. Evolution dans le temps de l'appui apporté par les outils informatiques.

Dans ce contexte, nous considérons comme stratégique le fait de conduire le travail de recherche en conception de systèmes à base d'électronique avec l'ambition de faire cohabiter, dans le même environnement de simulation, des constituants numériques, algorithmiques et analogiques pluridisciplinaires.

1.2.3 Conception Système et Co-design

A l'origine de nos réflexions, l'idée était d'explorer les techniques de codesign et co-simulation du matériel et logiciel. Aujourd'hui, nous sommes amenés à réfléchir sur le positionnement des méthodes et outils dans une approche plus générale système.

Afin de comprendre le problème dans son contexte réel, nous avons analysé (avec l'appui de la Société AIRBUS France) l'approche courante de conception des systèmes avioniques. La Figure 1-2 synthétise cette approche. Nous pouvons constater que la démarche n'est pas franchement descendante, que le « codesign » n'est pas encore appliqué et que la co-simulation est uniquement utilisée après la définition des architectures, dans les phases finales du processus. Le partitionnement matériel/logiciel est établi par expertise, dès les premières étapes, de la conception. L'implémentation du logiciel est donc fortement reliée aux architectures retenues a priori.

Cette pratique est liée aux systèmes avioniques qui sont de type « matériel dominant » dans des applications où, la réussite et le temps d'exécution sont des les facteurs critiques. Dans ces systèmes, une grande partie de l'architecture est conçue sur hardware et la partie software est rajoutée aux modèles matériels validés. L'ensemble est vérifié en fin de processus en utilisant des outils de co-simulation.

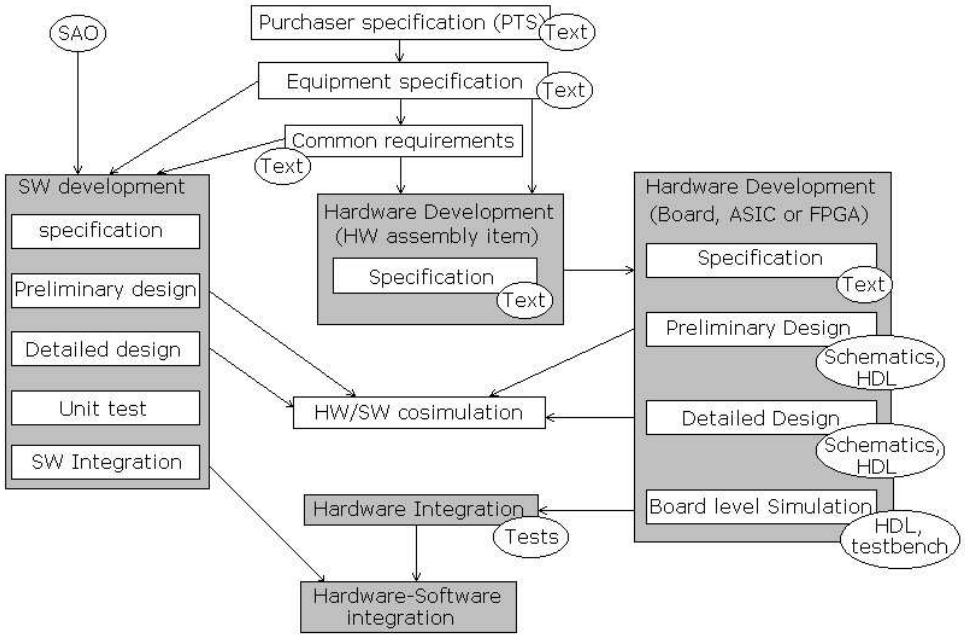


Figure 1-2. Schéma du processus courant de conception des systèmes électroniques chez Airbus France.

Dans cette analyse, l'application d'une démarche descendante, nécessite **un processus permettant d'arriver au moment crucial du partitionnement matériel/logiciel à une représentation virtuelle exécutable de la structure de conception**. A ce niveau de conception, cette représentation virtuelle exécutable aura une autre utilité pour le concepteur. Avant même la prise en compte des compromis liés aux choix technologiques, elle pourra tester la validité fonctionnelle du système. La simulation du prototype dans les étapes amont permettra aussi de détecter les points faibles par rapport aux performances attendues et de détecter les fonctionnalités qui pourraient être optimisées au sein du projet.

Le processus de codesign électronique (avec la co-vérification et la co-simulation comprises) a lieu à partir de la description comportementale des éléments du système. **La Figure 1-3 illustre notre interprétation de l'approche, le principe est de concevoir le matériel et le logiciel de façon simultanée, à partir de la définition des spécifications validées, bien avant de proposer des solutions de type partitions matériel / logiciel**. Dans ce contexte, la cible de nos travaux se situe à l'intérieur du carré rouge pointillé (indiqué par le *) du schéma de la Figure 1-3. Notamment entre les étapes de spécification et les instances de partition et d'arbitrage logiciel/matériel. Notre intention est de développer une méthode permettant d'engager ces étapes avec la certitude que conceptuellement et fonctionnellement le système à implémenter est correct.

Aujourd'hui, les communautés électronique et l'informatique ont développé leurs outils spécifiques. Pour définir des outils de conception de haut-niveau il est alors naturel de chercher à « uniformiser » les concepts des outils produits par les deux communautés. C'est la raison pour laquelle, en suivant la structure générale du schéma de la Figure 1-3 nous ferons un état de l'art non exhaustif des outils « métiers »de la conception électronique, logicielle et mixte pour enfin tenter de remonter au niveau supérieur qui est le centre de notre travail.

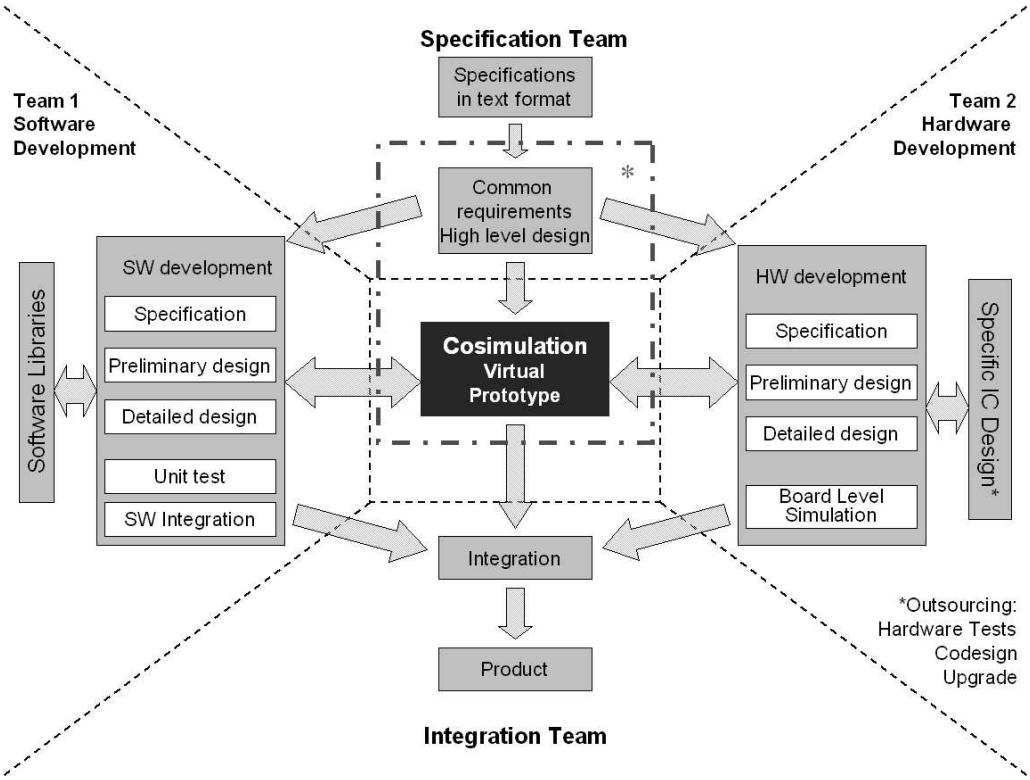


Figure 1-3. Proposition de structure de conception matérielle et logiciel. Nos travaux se centrent dans le pointillée*.

1.3 La conception électronique

Historiquement, les outils de la conception électronique ont été orientés, soit vers les systèmes analogiques, soit vers les systèmes numériques. Nous pouvons observer que les applications numériques ont évoluées plus rapidement en raison de l'utilisation croissante des calculateurs embarqués et de l'effet de masse des marchés des produits « grand public ». Nous nous intéressons ici aux systèmes mixtes qui sont un point clé au carrefour des outils analogiques et des outils numériques. En conséquence, nous distinguons trois cas : les systèmes électroniques analogiques, les systèmes hybrides et les systèmes numériques embarqués. Notons que la plupart des outils présentés sont toujours, à l'origine, des propositions d'équipes de recherche universitaire, développées ensuite par des constructeurs d'applications.

1.3.1 Simulation électronique analogique

Nous commençons ce tour d'horizon des outils et langages d'aide à la conception des systèmes à base d'électronique par l'un des outils pionniers de la simulation purement analogique, qui, au long des années, est devenu un standard industriel et académique : PSPICE. Suite à l'apparition de Spice 1 en 1972 [Mal02] à l'Université de Berkeley, Pspice [CDS03a] est devenu le standard de fait pour la simulation électronique analogique. Sa version, PSpice 9.2.3 est un des modules fonctionnels d'OrCad. Le premier simulateur de PSpice a été introduit en 1985. Depuis cette date, il a été constamment mis à jour en fonction de la technologie des ressources informatiques et des systèmes d'exploitation jusqu'au point de devenir un outil universellement utilisé dans l'industrie, dans les universités et dans les laboratoires de recherche. La plupart des fabricants de composants électroniques fournissent aujourd'hui des modèles écrits en PSpice.

PSpice est un simulateur complet pour la conception analogique. Avec ses modèles internes et ses bibliothèques largement rependues et développées, les systèmes à haute fréquence jusqu'aux circuits intégrés de basse puissance, tout peut être simulé. Dans la bibliothèque de PSpice, des modèles peuvent être édités mais les utilisateurs peuvent également créer des 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.

Malheureusement, quand il s'agit de grands systèmes, les simulations deviennent trop lourdes et demandent un temps d'exécution prohibitif.

1.3.2 La simulation mixte

Les exigences de la technologie et du marché ont imposé le développement d'outils plus puissants capables de traiter simultanément les domaines analogiques (pluridisciplinaires) et numériques. La plupart des systèmes électroniques actuels comportent des combinaisons de circuits analogiques et numériques. Ce besoin a entraîné depuis la fin des années 90, l'apparition de langages de description matérielle de systèmes à signaux mixtes [Coo01] MSHDLs. Ces types de langages offrent un grand intérêt dans une approche de conception système.

1.3.2.1 SABER

SABER [Ham01] est un outil très utilisé, développé par la société Analogy (aujourd'hui Synopsys) et orienté vers la conception système. Il offre la possibilité de faire des simulations de signaux et technologies mixtes : analogiques et numériques, grâce a l'existence de passerelles avec d'autres outils. Les algorithmes de simulation de SABER, fournissent une capacité de convergence qui permet à l'utilisateur d'arrêter et de relancer la simulation pour regarder les résultats intermédiaires et/où changer certains paramètres des composants sans quitter l'environnement de simulation. La liste d'analyses disponibles sur SABER inclut : l'analyse de Monte Carlo, l'analyse de sensibilité, l'analyse en fréquence, l'analyse de bruit, l'analyse de distorsion, le calcul de fonctions transfert, transformées de Fourier et simulation des tensions d'alimentation. Tous les modèles (numériques, analogiques et mixtes) de la bibliothèque SABER standard sont codés en langage MAST.

L'interface de co-simulation Saber/Verilog-XL [Ana01a] combine les capacités de SABER avec le simulateur pour la conception numérique Verilog-XL de Cadence Systems. Cette interface donne à SABER l'avantage de pouvoir co-simuler avec Verilog dans presque tous les principaux environnements de conception, y compris SaberSketch, des environnements de Mentor de Graphics, de Cadence ou Viewlogic. La sortie de la simulation est combinée et synchronisée en temps pour afficher et corréler les données analogiques et numériques. L'interface de Co-simulation de SABER/ModelSim [Ana01b] incorpore les simulateurs numériques EE VHDL EE PLUS de ModelSim.

L'interface Saber/Fusion [Ana01c] (ancienne STI) fournit un service efficace pour la simulation mixte analogique / numérique dans l'environnement de conception Powerview. Le STI combine le simulateur AHDL de SABER avec de ViewSim structural, de VHDL Speedwave et des simulateurs numériques de VCS Verilog. Le Saber/Fusion STI se combine également avec le logiciel d'intégration Frameway. Le résultat est une interface graphique qui fournit la co-simulation rapide de circuits conçus avec de nombreux modèles composants de bibliothèque disponibles dans les simulateurs de SABER et de FUSION.

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 monté 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.

1.3.2.2 Verilog AMS

Verilog-AMS [Acc98] a été créé sous la tutelle d'Accellera (Organisation de standards EDA) afin de mettre en place les extensions analogiques mixtes de 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.

Ainsi que VHDL-AMS (§1.3.2.3), Verilog-AMS peut être applicable aux systèmes électriques et non électriques. Le langage permet de faire des descriptions de systèmes, en utilisant des concepts comme des nœuds, des branches, et des ports. Les signaux de type analogique et numérique peuvent être présents dans le même module. Au contraire de VHDL-AMS, Verilog-AMS n'est pas un standard IEEE.

1.3.2.3 VHDL-AMS

VHDL-AMS est une norme IEEE [IEEE99] (1076,1 de 1999) qui élargit la définition du VHDL pour inclure la description des systèmes analogiques mixtes. Avec VHDL-AMS les systèmes qui étaient décrits en utilisant plusieurs outils tels que MATLAB, VHDL et SPICE peuvent être tous modélisés en utilisant un seul langage.

VHDL-AMS inclut toutes les propriétés du VHDL standard, avec en plus, la capacité de décrire les systèmes mixtes, analogiques et numériques par le biais de modèles multi abstractions, multidisciplinaires, hiérarchiques à temps continu et à événements discrets [Her02]. Au niveau conception système, VHDL-AMS peut être utilisé pour faire des descriptions de haut-niveau comme la description comportementale et le RTL (Register transfert level), les fonctions de transfert avec les transformées Z et de Laplace, des convertisseurs numérique/analogique et analogique/numérique, « phase-lock-loops » comportementaux, et les filtres analogiques et numériques. En revanche, VHDL-AMS ne permet pas de résoudre des systèmes à équations différentielles partielles ni des descriptions de caractéristiques géométriques des systèmes.

Pour la conception électronique détaillée, VHDL-AMS permet :

- Des simulations au niveau des portes logiques,
- des modélisations de circuits analogiques et de modèles au niveau transistor SPICE / VHDL-AMS,
- des descriptions de systèmes par des équations simultanées, non linéaires, différentielles et algébriques,
- de la modélisation et de la simulation des effets physiques liés au fonctionnement numérique.

Dans notre démarche, **VHDL-AMS présente l'avantage [Her02] de proposer un langage commun indépendant des fournisseurs et de la technologie**. Du point de vue technique, il permet une haute modularité facilitant les descriptions hiérarchiques. Pourtant le langage est complexe et les premières impressions de l'utilisateur peuvent être relativement décourageantes. Cette sensation est accentuée par le fait de ne pas pouvoir compter avec le support total d'une norme relativement récente et encore susceptible d'être modifiée. Cependant, une nouvelle version est en cours de préparation. Par rapport à la synthèse, VHDL-AMS inclut tous les sous-ensembles synthétisables de VHDL pour la partie numérique. Côté analogique, des premiers travaux ont été réalisés [Dob03] [DV03].

Quelques simulateurs sont déjà disponibles sur le marché :

- AdvanceMS™ [MG01a] ANACAD (Mentor Graphics),
- System Vision™ [MG03] version 8.3.0 de Mentor Graphics
- SIMPLORER® 7.0 [Ans03] développé par ANSOFT
- SMASH™ 5.1.3 [DI03] de DOLPHIN Integration,
- TheHDL de AVANTI.
- Hamster [Sim03], un simulateur gratuit pour PC, de SIMEC. Cet outil a disparu mais il s'utilise encore pour guider les « premiers pas » des utilisateurs de VHDL-AMS.
- SaberHDL™ [Syn03], de chez Synopsys propose l'option d'un simulateur intégré pour la simulation mixte. Le fabricant offrira un outil capable de supporter les langages : VHDL-AMS, MAST, HSPICE et Verilog AMS. SaberHDL pourra fonctionner sur Sun Solaris 2.6.8, Windows 2000 et RedHat Linux 7.2.

Nous avons testé, pendant ce travail de thèse, trois simulateurs VHDL-AMS. Les conclusions de l'étude comparative de David GUIHAL [Gui03] sont présentés dans les tableaux ci dessous, complétés des caractéristiques d'un troisième outil : SystemVision de Mentor Graphics :

Tableau 1-1 Tableau récapitulatif des caractéristiques et de l'ergonomie des outils VHDL-AMS étudiés pendant l'un des stages 2003 [Gui03] à Airbus.

Outil	Simplorer	Saber HDL	System Vision
Version	6.0 Pro	2003.06	
Editeur	ANSOFT	Synopsys	Mentor Graphics
Système d'exploitation	WindowsNT, 95, 98, XP et 2000	Sun Solaris Windows NT 4.0/2000 Redhat Linux 7.2	WindowsXP et 2000
Simulateur	Simec	Newton Calaveras	Eldo

Malgré l'existence d'un langage standard VHDL-AMS, les fabricants d'outils logiciels ne réalisent pas des implémentations totalement conformes à la norme. Les instructions ne correspondent pas au 100% de ce qui était prévu par le standard. De plus, certaines instructions ne sont pas opérationnelles dans les simulateurs. Les tableaux 1-1, 1-2, 1-3 et 1-4 présentent une comparaison des outils testés par rapport à leur couverture de la norme VHDL-AMS.

Tableau 1-2. Tableau comparatif des outillages de modélisation des simulateurs VHDL-AMS testés.

Outil	Simplorer	Saber HDL	System Vision
Code VHDL-AMS	Création automatique de l'entête du model via Model Agent	Utilisation d'un éditeur externe, puis lien	Editeur interne
Création du symbole	Automatique	Manuellement via Symbol Editor	Automatique et avec outil interne
Débuggage	A la création du composant	– Au moment de l'association d'un template et d'un symbole – A la génération de la netlist Au chargement dans le simulateur.	Au moment de la compilation du modèle
Création de composants hiérarchiques	Génération de code de la schématique, puis modification manuelle	Directe	Directe
Bibliothèques de composants	Très fournies en SML Eléments de base en VHDL-AMS	Eléments de base en VHDL-AMS	Eléments de base en VHDL-AMS

Tableau 1-3. Tableau comparatif de la gestion de la partie numérique.

Outil	Simplorer	Saber HDL	System Vision
Version	6.0 Pro	2003.06	2004.194.1
Editeur	ANSOFT	Synopsys	Mentor Graphics
Système d'exploitation	WindowsNT, 95, 98, XP et 2000	Sun Solaris Windows NT 4.0/2000 Redhat Linux 7.2	WindowsXP et 2000
Simulateur	Simec	Newton Calaveras	Simulateur VHDL proprietaire

Tableau 1-4 Comparatif [Gui03] de la couverture de la norme VHDL-AMS dans les outils testés Partie analogique.

Outil	Simplorer	Saber HDL	System Vision
Bibliothèques dédiées à l'analogique	Bibliothèques de base plus des modèles SML	Bibliothèques de base plus des modèles MAST	Bibliothèques de base plus des modèles spice
Instructions non gérées	• Quantités vectorielles	• Break pour initialisation, • Procedural • Quantités Verctorielles	• Break pour initialisation, • Type Record • Appel de procédure concurrent • *Generate* uniquement pour des instances de composants. • Instruction simultanée Procedural (évaluée à chaque ASP)

En conclusion, hormis les soucis d'implémentation de la part des fabricants d'outils, il apparaît que **l'approche générale et ouverte de VHDL-AMS est la plus indiquée pour notre démarche**. L'intégration naturelle de modèles de plusieurs disciplines permet d'avoir une vraie approche système en adéquation à notre problématique générale de conception.

1.3.3 Le codesign matériel / logiciel

Le codesign est l'une des techniques les plus intéressantes car elle s'efforce de mettre en place une vraie méthode de conception simultanée du matériel (100% numérique) et du logiciel. Son émergence est due à la grande et croissante ressemblance de la conception des systèmes numériques avec la conception 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**. Voici quelques exemples d'outils conçus pour le codesign.

1.3.3.1 La conception basée sur le langage C

Une bonne partie des outils destinés à la conception de systèmes mixtes numériques utilisent l'approche « C-Based System design », c'est-à-dire, les systèmes sont écrits sous la forme du code C ou C++. Nous commençons notre analyse avec SpecC :

SpecC : Créé à l'Université de Californie Irvine par l'équipe de travail du professeur Daniel Gajski au CADLAB, SpecC [DGG01] est plus une extension ou une adaptation du C, il est un langage de codesign «hardware/software» basé sur le langage C et proposé par UCI CADLAB. C'est une version élaborée d'ANSI-C dont le niveau d'abstraction le plus haut est décrit à base de machines à états finis. SpecC propose des spécifications comme canaux de communication, des représentations hiérarchiques, de la simultanéité et de l'abstraction de la synchronisation. Il est conçu pour être un langage unique qui peut être utilisé dans toutes les étapes du processus de codesign matériel / logiciel. Un projet SpecC se compose d'un ensemble de déclarations comportementales, déclarations de canaux et d'interfaces. Un comportement est une classe avec un ensemble de ports : L'ensemble des comportements secondaires, l'ensemble des canaux, l'ensemble des variables et des fonctions « privées » et une fonction principale « publique ». Par ses ports, un comportement peut être relié à d'autres comportements ou canaux afin de communiquer. La fonctionnalité d'un comportement est

indiquée par ses déclarations de fonction. Un canal est une classe qui contient la transmission. Il se compose d'un ensemble de variables et de fonctions appelées méthodes, qui définissent un protocole de communication. La version SpecC 2.0 a été développée par l'équipe du professeur Masahiro FUJITA, à l'Université de Tokyo. Elle intègre des améliorations dans la gestion des événements concurrents, des interruptions, et du parallélisme. Il est intéressant de noter que récemment l'équipe du professeur FUJITA a été contactée par des concepteurs de satellites japonais qui veulent aborder la conception des micro-systèmes embarqués du point de vue système. Pour l'instant SpecC ne comporte pas d'options pour la conception analogique et mixte. Le groupe de travail pour la version 3 a été lancé à Tokyo le 8 octobre 2002. Il étudie la faisabilité d'une extension analogique du langage. A notre avis, il reste encore du travail pour arriver à un langage système général. Le développement de l'outil à été pénalisé par la grande partie de marché couverte par son grand concurrent SystemC.

System C : Peut être le plus utilisée des approches « C based »**.** Les origines de SystemC remontent au milieu des années 90, dans les travaux de l'Université de California Irvine et du groupe Synopsys. Le premier produit était appelé « Scenic » puis « Fridge ». La première version SystemC 0.9 est sortie en 1999 avec l'incorporation des éléments récupérés de N2C –Coware.

System C est un ensemble de bibliothèques créées en langage C++, permettant de faire la description d'un système logiciel – matériel par le biais de spécifications exécutables et de plusieurs niveaux d'abstraction pour un même système. SystemC fournit la possibilité de créer des modules, processus fonctionnels et portes logiques. Le compilateur CoCentric SystemC synthétise la description « hardware » écrit en SystemC au niveau des portes logiques (gate-level netlist) ou en Verilog ou VHDL pour faire de la synthèse sur des FPGAs. Les modèles SystemC sont écrits sur le formalisme des FSM (« Finite State Machines »).

Tableau 1-5. Quelques outils qui supportent SystemC**.**

Fabriquant	Outil
Axys Design	MaxCore developer Suite
Axys Design	MaxSim
Cadence	SPW
CoFluent Studio	Cofluent Design
CoWare	CoWare N2C Design System
Forte Design System	Cynlib Tool Suite
Innoveda	Visual Elite-Architect
Mentor Graphics	Vstation TBX
Synopsys	CoCentrics System Studio
Synopsys	SCC Synopsys Cocentric SystemC compiler
Veritools	SuperC
Virtio	Virtual Prototyping to SystemC
WHDL Language	Rule Checker & Rule Generator

Cette approche très intéressante est devenue l'un des standards de fait pour la conception et synthèse de systèmes numériques mixtes matériels et logiciels. Tel que l'illustre la Tableau 1-5, la plupart de fabricants d'outils CAO proposent SystemC parmi leur produits. Au moment d'écrire ce mémoire, une initiative commence à prendre de l'ampleur, il s'agit d'étendre l'utilisation de SystemC aux systèmes électroniques mixtes matériels et logiciels. Des travaux de mise en forme d'une proposition **SystemC-AMS** ont été proposés par [GEV04]. L'approche système préconisée est

intéressante car elle **peut représenter une alternative pour le traitement des systèmes avec tout type de composantes : Analogiques, numériques et logicielles.**

Handel-C [Cel02] est un langage écrit sur la base de ISO/ANSI-C destiné à l'implémentation d'algorithmes sur « hardware », à l'exploration architecturale et au codesign. Handel-C permet la conception de matériel en utilisant des méthodes de conception de logiciel élargies avec des particularités pour le développement de matériel. Elles incluent des largeurs variables des structures (vecteurs) de données, le traitement parallèle des communications et des événements. HandelC n'utilise pas de machines à états finis, grâce à une méthode propriétaire de description des écoulements périodiques et parallèles.

Les modèles Handel-C peuvent être insérés dans une méthodologie de réutilisation car des fonctions peuvent être compilées dans des bibliothèques et être employées dans d'autres projets. Des noyaux écrits sous Handel-C peuvent être exportés comme boîtes noires d'EDIF, de VHDL ou de Verilog pour leur réutilisation.

D'après le fabricant, Celoxica, les points forts de HandelC sont :

- un langage de haut-niveau basé sur ISO/ANSI-C pour l'exécution des algorithmes sur « hardware ».
- le langage ne demande pas de grands investissements en temps de formation des utilisateurs.
- HandelC permet de faire des appels directs sur des fonctions externes écrits sous C/C++ et vice-versa.
- des extensions spécifiques pour le matériel incluant la gestion du parallélisme et des communications.
- construction des noyaux spécifiques pour la suite Celoxica DX.

Nous trouvons intéressante la compatibilité de l'outil et l'utilisation de leur propre modèle de représentation des états des systèmes, mais il est trop focalisé sur la réalisation matérielle. Il manque la généralité requise par notre approche.

N2C [Cow01] a été développé par la société CoWare. Cet outil permet de capturer les spécifications d'un système numérique dans un modèle exécutable et implantable à partir de langage C/C++. Avec N2C, l'utilisateur peut faire une spécification concourante, visant deux objectifs :

- Une implémentation et vérification de matériel et logiciel embarqué spécifique à l'application.
- L'évaluation et intégration de la propriété intellectuelle (IP) de matériel et de logiciel vers des nouveaux produits ou dérivés.

CoWare N2C est conçu pour co-exister avec la plupart des outils commerciaux : Simulateurs de HDL, simulateurs de positionnement d'instruction (ISS), outils intégrés de l'environnement de développement de logiciel (IDE), et des systèmes d'exploitation temps réel. Les outils de synthèse reçoivent la sortie de N2C pour démarrer la déclinaison et synthèse du système. Une version universitaire de N2C est disponible pour les membres d'Europractice Software Service [Eur04].

1.3.3.2 Les outils de haut-niveau

Dans une autre catégorie, nous avons placé des outils qui abordent le problème du codesign à un niveau plus élevé que la partition logiciel/matériel. Parmi eux, le projet POLIS [Clo01] de l'Université de Californie Berkeley a été développé afin de créer une méthodologie formelle unifiée pour la modélisation complète des systèmes embarqués. Cette méthodologie inclue la partition matérielle/logicielle, la synthèse automatique et la vérification. POLIS a été développé sur le modèle de calcul formel CFSM ou « Codesign Finite State Machine ». POLIS est un logiciel expérimental. Bien qu'il ait été testé sur plusieurs exemples de dimension industrielle [FLL+98][San96], il ne peut être applicable qu'à certains domaines spécifiques.

Eagle*i* [Syn00] est un outil pour la co-vérification Logiciel/Matériel depuis la post-partition logiciel matériel jusqu 'au prototype physique. Eagle*i* supporte des outils EDA de haut rendement, la simulation cycle à cycle, les accélérateurs de matériel, et l'émulation de matériel pour la conception multiprocesseur. Avec Eagle*i,* il est possible de distribuer la simulation à travers un réseau pour améliorer la vitesse de simulation. Il fournit une plate-forme d'interopérabilité UNIX/PC. Cette caractéristique peut le rendre intéressant pour son intégration dans des plate-formes de conception.

Seamless CVE [MG00a] est un outil de Mentor Graphics pour la conception électronique. Grâce à son interface « Plug-In » (SPI), il est capable de réaliser de la simulation multiprocesseur. Seamless CVE est compatible avec plusieurs outils de vérification et description [MG00b] comme ModelSim VHDL, Verilog XL, VSC et Voyager, et avec plus de 70 microprocesseurs des différents fournisseurs.

A cause de l'exécution du modèle complet d'un microprocesseur, la vitesse du simulateur peut être six ou sept fois plus lente que l'exécution temps réel. Seamless CVE accélère la co-simulation grâce à la séparation fonctionnelle du microprocesseur de son interface électronique. La suppression sélective de certains cycles dans la simulation matérielle est facultative. Les simulations matérielles et logicielles sont divisées en un simulateur d'instructions ou « Instructions Set Simulator » et un modèle d'interface ou « Bus Interface Model » pour le comportement électronique des entrées/sorties du processeur. L'arbitrage entre l'exécution des simulations est réalisé par le « Cosimulation Kernel ». Nous trouvons que cet outil demande de connaître préalablement l'architecture du système et donc est utile seulement lorsque les choix de conception ont été réalisés. Nous le classons dans la catégorie d'outils système car il permet de vérifier le fonctionnement complet de l'application.

A notre avis, l'outil le plus intéressant et le plus proche de notre démarche est **CoFluent Studio SDE,** proposé récemment par la société CoFluent Design. Il est orienté vers la conception de systèmes électroniques numériques comportant des implémentations matérielles et logicielles. Les origines de ce logiciel se trouvent à l'Ecole Polytechnique de Nantes sous la direction de Jean-PAUL CALVEZ. Le principe d'utilisation [Per04] est de distinguer clairement les représentations fonctionnelles et architecturales du système à concevoir. En effet, l'outil permet de réaliser ces descriptions indépendamment. Trois niveaux d'abstraction ont été identifiés pour l'analyse des performances :

- Le niveau Système,
- Le niveau composants
- Le niveau des communications entre composants.

La Méthodologie de Conception de Systèmes Electroniques [Cof03] (MCSE ou CoMES en anglais) est utilisée avec l'outil afin de gérer les différents niveaux de complexité d'un projet de conception.

La description finale du comportement des systèmes est générée sous deux formes de génération automatique de code. La première, destinée à la modélisation Système en C/C++, la deuxième en VHDL synthétisable orientée à l'implémentation des systèmes sous la forme de circuits intégrés.

Nous trouvons cet outil particulièrement intéressant car il propose une approche générale pour la conception de haut-niveau du domaine électronique, en considérant la modélisation du comportement des systèmes avec des modèles formels de calcul, de plus une issue vers la matérialisation est proposé. Dans de futurs travaux, il sera convenable d'approfondir cette démarche afin d'envisager des ouvertures vers une généralisation orientée VHDL-AMS donnant la possibilité de gérer des projets pluridisciplinaires.

1.4 Les méthodes pour la conception logicielle

Nous continuons maintenant avec les méthodes pour la conception logiciel. Ceci correspond dans notre mémoire au côté gauche de la Figure 1-3 que nous avons déjà présente. Ces méthodes ont été pionnières dans la façon d'aborder le problème de la conception. En effet, les approches de la conception logicielle ont fait émerger des nombreux concepts également intéressants pour la conception amont.

1.4.1 SADT et SA/RT des méthodologies à l'origine de la réflexion système

Nous trouvons les origines de nos propres réflexions dans la seconde moitié du siècle dernier où la complexité croissante des systèmes avait déjà demandé des efforts des scientifiques et des ingénieurs pour l'établissement d'outils et/ou des méthodes permettant d'alléger la tâche de spécification et de conception de ces systèmes. Parmi ces premiers travaux, ROSS avec SADT [Ros72] est une des pionniers dans la recherche d'une solution au problème de la spécification et de la conception des systèmes à haut-niveau. Développée à partir de 1972, cette approche avait pour objectifs de couvrir l'analyse de besoins, la spécification, la conception et la documentation en facilitant le partage de l'information entre les utilisateurs. Le modèle utilisé propose une description en blocs (d'activités ou de données) reliés par quatre types de liens : Entrées, sorties, contrôle et mécanismes. Les blocs SADT peuvent être décomposés en niveaux hiérarchiques. L'approche propose deux formes possibles de diagrammes : Les « actigrams » et les « datagrams » **qui représentent deux vues différentes d'un même système**. L'utilisation de SADT été basée sur des principes de délimitation du contexte du système et de la limitation de taille de l'information. Les décompositions étaient limitées à sept blocs ±2 par feuille. Cette approche simple et compréhensible n'est pas exclusive à un métier spécifique. Un banquier, un fonctionnaire,… peuvent lire un diagramme de leur domaine sans connaître la méthode. Efficace en spécification des exigences, elle pressente un certain nombre d'inconvénients dès que des phases de conception sont abordées notamment ses insuffisances pour l'expression des algorithmes de contrôle.

D'un autre côté, **SA/RT** (Structured Analysis with Real-time-Extensions) [WM85] propose, à partir d'une analyse établie sur une représentation graphique, une modélisation système dans laquelle deux facettes sont clairement différentiables : un modèle du processus statique qui lui est attaché, et

un modèle de contrôle dynamique qui en permettra l'utilisation. L'originalité de cette méthode est la prise en compte de l'aspect dynamique du système. SA/RT est donc bien adapté aux applications temps réel à fort comportement dynamique.

1.4.2 UML : le langage unifié de modélisation

UML est un langage qui émerge comme un standard de fait pour la conception de systèmes logiciels à haut-niveau, il a été proposé par l'OMG [Omg03] en 1997, avec comme objectif quatre activités principales [Bro03] du processus de conception :

- Une description 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 de systèmes complexes,
- la documentation de la totalité du projet, dès les spécifications jusqu'aux tests de fonctionnement.

L'application de UML exige l'adoption d'une méthodologie claire et l'utilisation d'un bon outil logiciel mettant en œuvre le langage. Malgré cette volonté d'unification, UML n'est pas une solution totale pour la conception système, car elle n'établie pas la façon dont les diagrammes doivent être employés et moins encore un principe d'intégration ou d'interopérabilité entre eux. **L'utilisation du langage perd beaucoup d'efficacité sans une méthodologie et sans le support d'un outil.**

Dans le langage UML, plusieurs types de représentations graphiques [Mil03] sont possibles. Cinq modèles de représentation (chacun avec un ou plusieurs types de diagrammes) conforment la sémantique UML 1.5, à savoir :

- Modèle d'utilisateur :
  - « Use case diagrams », ces diagrammes représentent le fonctionnement du système du point de vue d'un observateur externe. Leur but est de montrer ce que le système fait sans détailler le « comment ».

- Modèle structurel :
  - Diagrammes de classes : ces diagrammes statiques montrent les classes du système et leurs connexions.
  - Diagrammes à objets : il s'agit d'une simplification des diagrammes de classes. Ces diagrammes décrivent les objets avec leurs interactions.

- Modèle comportemental :
  - Diagrammes de séquence : il s'agit de diagrammes d'interaction qui montrent le séquencement des opérations du système. C'est une vue temporelle.
  - Diagrammes de collaboration : Ces diagrammes d'interaction comportent la même information que les diagrammes de séquence mais se focalisent sur les rôles au lieu des temps.
  - Diagrammes à états et leurs extensions les « Statechart diagrams » : ils illustrent les états possibles du système et les transitions qui provoquent les changements des états.
  - Diagrammes d'activités : Il s'agit essentiellement de « flowcharts ».

- Modèle d'implémentation :
  - Diagrammes de composants : ces diagrammes représentent l'équivalent matériel et ou logiciel des diagrammes de classes.
- Modèle d'environnement :
  - Diagrammes de déploiement : Ce dernier type de diagramme illustre les configurations physiques du matériel et du logiciel. D'une façon générale ils permettent de décrire les nœuds de distribution et leurs interactions dans le cas de systèmes distribués.

Hormis l'approche objet, l'avantage majeur du langage UML est la multiplicité de diagrammes qu'il offre. I**l permet au concepteur de créer différentes représentations du fonctionnement du système**. Il pêche encore par sa complexité sémantique et le manque d'interopérabilité entre diagrammes.

1.5 Autres approches métiers et mixtes

Nous terminons notre état de l'art avec des approches qui ont essayé de se hisser à un niveau plus amont pour aborder le problème de la conception système. Ici, nous avons identifié deux volets : d'un part, les outils issus des initiatives des communautés indépendantes d'aborder le « haut-niveau » relatif à leur domaine, tels que l'automatique, le logiciel et l'électronique. D'autre part, les initiatives récentes qui « fédèrent » la conception système avec des approches généralistes et hétérogènes. Nous aborderons ce second cas dans une section postérieure.

La communauté des automaticiens utilise depuis des année M**atlab® et Simulink®** comme leurs outils de base pour le calcul scientifique. Il s'agit, peut-être des plus célèbres outils de modélisation mathématique globale, Matlab® et Simulink® de chez Mathworks. On dispose en 2004 des versions 7 et 6 respectivement. Traditionnellement, ils sont utilisés pour faire de la modélisation générale de systèmes par des fonctions de transfert, avec une forte orientation vers les systèmes de contrôle et commande. Ils permettent de réaliser une modélisation détaillée des algorithmes de contrôle des systèmes dans des domaines multiples. Les versions actuelles comportent plusieurs « toolboxes » permettant de participer à la conception système à différents niveaux; parmi les applications les plus intéressantes, nous trouvons l'utilisation de Matlab®/Simulink® en combinaison avec des outils VHDL pour réaliser le test de modèles pour leur implémentation matérielle. Le lien de co-simulation avec ModelSim®1.1 [TMW03] permet de co-simuler et de vérifier du VHDL et du Verilog. Cet outil permet de réaliser des vecteurs de test « logiciels » en intégrant les solutions HDL avec les algorithmes, ceci permet de vérifier le fonctionnement du HDL par rapport au modèle original ainsi que de donner des caractéristiques comportementales aux « testbenches ». Ce principe est aussi répandu pour réaliser des essais de type « harwdare in the loop », décrits de façon plus détaillée par [Gom01].

Parmi les offres de Mathworks nous trouvons des applications pour la spécification et la modélisation des systèmes automobiles [TMW04a], la conception électronique mixte et la modélisation des composants. Ils proposent aussi des solutions pour la conception des systèmes embarqués [TMW04b] et de certaines applications aérospatiales, notamment [TMW04c] pour la conception des systèmes de commande et la validation de leurs interfaces homme/machine, la modélisation des systèmes mécaniques, des sources d'énergie et pour la modélisation détaillée de l'environnement de l'appareil en considérant des aspects tels que le vent et la gravité.

Récemment, [OG04] ont réalisé une comparaison des rôles relatifs de MATLAB/Simulink et de VHDL-AMS dans la modélisation système. Ils focalisent leur étude sur les aspects de modélisation entre la voie base des lois de Kirchhoff proposée par VHDL-AMS et la modélisation « signal flow » de MATLAB/Simulink. L'approche de VHDL-AMS semble être plus simple et plus directe par rapport à l'expression des échanges d'énergie entre les blocs constituants du système due à la bidirectionalité des connexions : Matlab demande de doubler le nombre de signaux pour réaliser une modélisation équivalente.

Dans la mesure où les outils de simulation progressent, nous pourrons envisager des modélisations couplées VHDL-AMS/Matlab dans lesquelles les équipes de conception pourront combiner leurs « savoirs-faire » en matière de systèmes de commande sous Matlab, pour élaborer des « testbenches », pour vérifier les modèles écrits en VHDL-AMS ou bien pour modéliser et synthétiser des lois de commande incluses dans ces systèmes.

Du côté informatique, de nombreux outils ont été proposés : Esterel Technologies a créé **Esterel Studio** [DD00], pour la spécification et le développement des systèmes numériques et logiciels temps réel en utilisant la représentation hiérarchique graphique SyncCharts, donc, une notation graphique conçue par Charles ANDRE [And96] à l'Université de Nice Sophia-Antipolis. Le langage de programmation synchrone et son compilateur ont été conçus à Ecole des Mines de Paris et à l'INRIA[i]. Les travaux de recherche [Cla01] sur le langage, pour la plupart d'origine française, sont à la base de la création, en avril 2000, de la société d'Esterel Technologies SA.

L'approche synchrone d'Esterel studio pour la modélisation et la programmation a été retenue afin d'éviter les erreurs et les difficultés propres de la conception de ce type de systèmes, par le biais de méthodes traditionnelles. Esterel Studio utilise une sémantique de type FSM (Finite State Machine) très pure, idéale pour concevoir des systèmes indépendants de l'implémentation et dominés par des commandes.

Esterel Studio a été également utilisé comme entrée optionnelle vers l'outil atelier de conception simultanée VCC de Cadence Design Systems. Cadence n'a pas prévu d'inclure un support Esterel natif au sein de VCC, mais travaille à l'interfaçage de VCC avec Esterel Studio.

L'outil comprend un éditeur graphique, un simulateur graphique, des outils de vérification formelle (FVT, Formal Verification Tool), ainsi que des générateurs de code Esterel. La sortie d'Esterel Studio peut se faire en langage C ou Java, pour les logiciels, ou en VHDL, si le système n'est destiné qu'à une implémentation purement matérielle numérique.

Avec des racines aussi informatiques, le projet PTOLEMY [DHJ+02], de l'Université de Californie Berkeley, est dédié à l'étude de la conception hétérogène des systèmes embarqués concurrents et particulièrement ceux qui comportent des technologies et des domaines mixtes. L'un des points le plus importants dans Ptolemy est la méthodologie pour définir et produire des systèmes embarqués. La base de cette approche est le choix des modes de calcul dès le départ de l'élaboration des modèles. L'objectif de Ptolemy est de gérer l'interopérabilité des prototypes écrits avec des différents modèles de calcul. Il introduit le concept de domaines parmi lesquels :

[i] Institut National de Recherche en Informatique et en Automatique.

- Les processus séquentiels communicants,
- le temps continu,
- les événements discrets,
- les événements discrets distribués,
- le temps discret,
- les machines à état finis,
- les réseaux de processus,
- le flot de données synchrones,
- le modèle synchrone / réactif,
- etc…

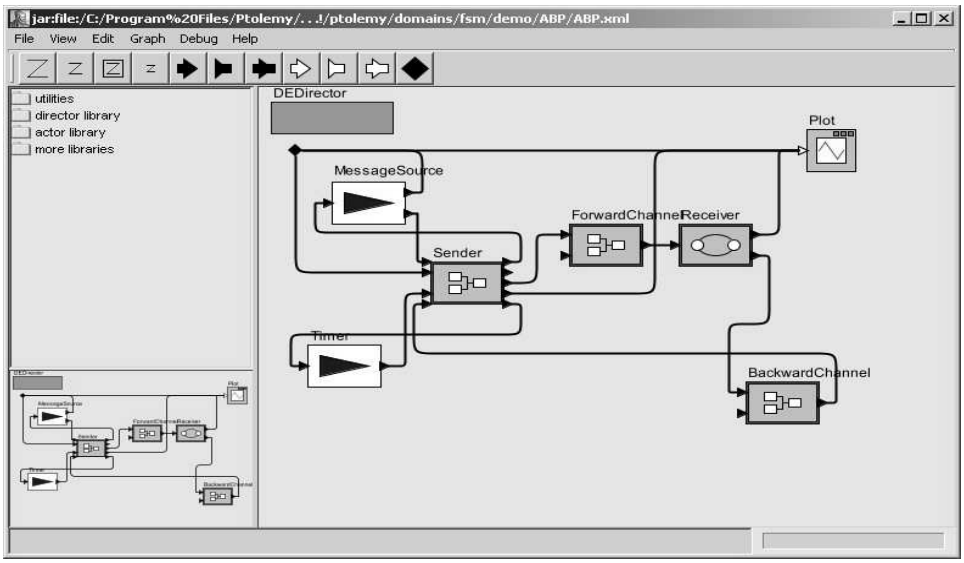


Figure 1-4. Représentation graphique d'un modèle Ptolemy sous le domaine des FSM.

Afin de faciliter la compréhension, les modèles Ptolemy sont fournis avec une interface graphique (Figure 1-4). La description visuelle représente une syntaxe alternative susceptible d'être associée aux modèles de calcul. Cela est spécialement utile avec les modèles complexes et hétérogènes. Le nom de cette interface est Vergil. Ptolemy II comporte également la capacité de supporter le format XML MoML.

Finalement nous trouvons des outils plutôt orientés vers les systèmes électroniques, notamment numériques. **MLDesigner®** est une approche pour les systèmes électroniques complexes utilisés dans des missions spatiales et planétaires discutée dans [Sal04]. Il s'agit d'une approche hiérarchique qui cherche à valider les composantes à concevoir dans une représentation comportementale de la mission où ils seront utilisés. Cet outil permet aussi la validation et la co-simulation de systèmes mixtes hardware/software. Les systèmes et les performances attendues sont représentées à haut-niveau par des machines à états finis. La partie matérielle est modélisée en termes de ressources utilisées et le logiciel comme le temps d'exécution des fonctions et procédures. MLDesigner fournit à l'utilisateur des modèles écrits sous XML, les résultats des simulations et le code généré à travers une interface unifiée.

Le principal avantage de MLDesigner est de proposer un environnement général de haut niveau pour des systèmes numériques mixtes dédiés à des fonctions critiques.

Nous trouvons aussi, **HDL Designer** [MG03]. Il s'agit d'un outil nouveau proposé par Mentor Graphics destiné à la conception système numérique de haut-niveau. Il permet de créer des projets complexes écrits en VHDL ou Verilog (ou les deux), c'est-à-dire des projets à langages mixtes pour des ASICS ou des FPGA. L'outil comporte des fonctionnalités pour la gestion de l'information dans le cas de la conception distribuée, notamment un gestionnaire de versions et un système automatique et dynamique (mise au jour en ligne) de génération de la documentation du projet [Dew03]. Pour ce faire, l'outil comporte la possibilité de réaliser automatiquement des représentations textuelles des projets, notamment sous la forme de tableaux de synthèse, par exemple des entrées et des sorties du système. Cette option nous semble très utile car elle permet de mettre en lumière, en temps réel, les versions du projet même s'il y a plusieurs sessions ouvertes.

HDL Designer est un outil construit avec une orientation claire vers le partage des documents du projet. Tel qu'il est présenté dans [Mul02], le concepteur ou l'équipe de conception peut choisir la quantité d'information à partager dans des applications où le processus de conception a lieu entre sites géographiquement délocalises et distants.

Hormis les fonctionnalités opérationnelles de l'outil, ce qui nous semble le plus intéressant c'est la proposition d'utiliser des machines a états sûrs. Selon [Zho02] les états du système peuvent être construits à base de machines à états, avec un nombre d'états $N=2^M$ où M est le nombre de registres nécessaires pour décrire ces états. Les états sont donc, tous atteignables en termes de registres qui les modélisent. Cette règle permet de résoudre le problème qui apparaît lorsque les outils d'optimisation génèrent des architectures « trop optimales » par rapport au nombre d'états mais qui ne sont pas physiquement réalisables

Finalement, considérons **Rosetta**. Les activités de standardisation de Rosetta [Per02] ont été lancées sous la supervision d'Accellera. Rosetta est un langage de conception au niveau système développé spécifiquement pour faire face à la complexité et la diversité des systèmes actuels. Rosetta [Acc03] permet aux concepteurs de développer et d'intégrer des spécifications écrites sous plusieurs modèles afin de fournir l'appui sémantique pour l'application de l'ingénierie coopérative dans la conception des systèmes électroniques.

Le comite de standardisation de Rosetta cherche à développer une syntaxe et une sémantique standard pour le langage de base. Il cherche aussi à établir les conditions pour la mise en place de l'application et d'une collection de domaines standards : Rosetta. Il s'agit aussi de développer les directives pour intégrer les modèles de Rosetta dans des flots de conception existants, notamment dans le EDA (Electronic Design Automation).

Rosetta est encore un outil en cours de développement, mais le prototype est disponible sur le site WEB de l'Université du Kansas.

1.6 Les approches « fédératrices » pour la conception système de haut niveau

Suite aux efforts des diverses communautés pour résoudre le problème de la conception système de haut niveau dans leur domaine, on peut maintenant envisager, de « remonter » la conception à un niveau supérieur qui fait abstraction des métiers de la conception classique. Tel est le

cas de UML 2.0 et plus particulièrement de SysML™, deux importantes initiatives que nous présentons ci dessous.

1.6.1 UML 2.0 : la solution de demain pour l'ingénierie des systèmes ?

Une nouvelle version du langage UML (§1.4.2) est en cours de spécification à l'heure actuelle. Le langage UML 2.0 [Dol03] propose des modifications par rapport à la norme précédente. La 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, pour le support de modèles exécutables et pour la génération et la réutilisation de composantes. Cette nouvelle version est notamment concernée par certains points faibles remarqués par la communauté des utilisateurs UML : Une sémantique trop vague, le fait d'être une sorte de représentation graphique de C++. UML est trop complexe et malgré tout non complet, et enfin, il pêche par manque d'interopérabilité entre diagrammes. Ces améliorations sont proposées sous la forme de modifications des anciens diagrammes (UML 1.5) ou avec la proposition de nouveaux diagrammes. Le résultat de cette nouvelle version donne un ensemble de 13 diagrammes dont 7 proviennent de UML 1.5 :

- les cas d'utilisation,
- le diagramme de classes,
- le diagramme d'objets,
- le diagramme d'activité,
- le diagramme de séquences,
- le diagramme de composants,
- le diagramme de déploiement.

Deux des anciens diagrammes ont été renommés : le « statechart diagram ». Il est désormais appelé « state machine diagram », le diagramme de collaboration est devenu diagramme de communication.

Dans UML 2.0 le langage a été augmenté de 4 diagrammes :

- « Composite Structure »
- « Interaction Overview »
- « Timing »
- « Package »

Comme dans les versions précédentes de UML, l'OMG ne propose pas, dans sa version UML 2.0, une méthodologie associée. La question à résoudre reste encore identique : Quelle est la méthode à suivre pour utiliser l'ensemble de diagrammes UML pour la conception et la modélisation de systèmes ?

Des travaux récents proposent l'intégration de UML et VHDL-AMS comme une alternative à la modélisation système. Une idée proposée sommairement par [Lem99] et approfondie par [CMM04] décrit une méthode pour obtenir des modèles VHDL-AMS analogiques à partir de UML. Ils ont employé les diagrammes de classe d'UML pour « transférer » la structure du code VHDL-AMS et ils ont défini l'ensemble des mots-clés pour adapter UML à la conception de systèmes analogiques. Cette proposition est intéressante au niveau structurel du code. Une proposition innovatrice [HF04] a été récemment développée, elle propose une extension du langage VHDL-AMS pour faciliter

l'interfaçage bi-directionnel avec des méthodes générales de spécification basées sur UML et avec d'autres formalismes de haut-niveau.

1.6.2 SysML™

UML est de plus en plus utilisé pour la conception de systèmes logiciels, la réussite de ce langage pourrait nous amener, dans un avenir proche, à des alternatives pour la conception de systèmes en général. En effet, une communauté très importante de l'ingénierie système considère le langage UML assez structuré et robuste pour supporter des extensions visant la couverture de la problématique de la modélisation des systèmes en général. Avec l'arrivée de la nouvelle version UML 2.0, un nouveau langage SysML™ [Sys03] a été proposé au mois de novembre 2003 pour une normalisation par un consortium de partenaires, groupe Ingénierie Système (IS) de l'OMG et l'INCOSE, de divers secteurs de l'industrie, des entités gouvernementales et des fabricants d'outils logiciels. A l'heure actuelle, la définition du langage arrive à la phase finale avant son officialisation. L'objectif [Hau03a] des partenaires du comité SysML™ est de proposer une adaptation de UML destiné aux besoins de l'ingénierie de systèmes. Le nouveau langage doit permettre de supporter la modélisation d'une large variété de systèmes complexes y compris leurs parties matérielles et logicielles, les informations sur le personnel impliqué dans le projet et même les procédures de conception et de réalisation du système. Selon les informations disponibles au moment de la rédaction de ce mémoire, la personnalisation de UML 2.0 permettra de suivre le processus de conception dès les spécifications jusqu'à la validation et la vérification. Sur ce principe, le nouveau langage permettra de représenter les aspects suivants de systèmes complexes mixtes et multi métiers :

- Structure : e.g., hiérarchie et l'interconnexion des systèmes
- Comportement : e.g., descriptions a base de fonctions et d'états
- Propriétés : e.g., modèles paramétriques, propriétés temporaires
- Expression des besoins : e.g., hiérarchie des besoins, traçabilié
- Vérification : e.g., les « test cases », vérification des résultats
- Autres : e.g., des relations spatiales

Le consortium SysML a prévu de rajouter, de compléter et de modifier certains diagrammes UML 2.0. En principe, ce nouveau langage comportera des diagrammes [Hau03b] destinés à la modélisation des besoins, du comportement et des paramètres des systèmes. D'importantes modifications auront lieu sur les diagrammes de structures [Hau04a] pour pouvoir supporter des descriptions des éléments contenus, des ports, des interfaces et des connecteurs. Les partenaires du consortium visent un langage dont ses principales caractéristiques doivent être:

- permettre l'application de la norme IA632 [Mar04] qui définit clairement les processus de l'IS et en particulier de l'Ingénierie des Exigences (Requirements Engineering),
- facilité d'utilisation,
- non-ambiguïté,
- complétude,
- adaptabilité à différents domaines,
- capacité de réaliser des échanges complets entre modèles,
- possibilité d'évolutions,
- indépendance des processus et des méthodes,
- conformité au meta-modèle UML,
- vérifiabilité.

1.7 La réutilisation et la gestion de l'information

Parmi les considérations techniques requises pour l'implémentation d'approches telle que la nôtre, nous devons tenir compte des méthodes existantes pour la gestion des informations des projets. Quel que soit notre choix de méthodes ou d'outils, nous nous trouverons, tôt ou tard, face à la nécessité de communiquer nos modèles ou d'intégrer dans notre projet ceux d'autres partenaires.

Vu l'énorme utilisation de l'Internet et de la nécessité croissante de partager des modèles à distance, nous proposons, dans le cadre des partenariats actuels, de tenir compte plus particulièrement du langage XML. Ce langage est idéal pour l'écriture des bases des données, les modules de propriété intellectuelle (IP) et pour écrire de façon structurée nos modèles en envisageant une politique ultérieure d'échange.

1.7.1 XML « Extensible Markup Language »

« Extensible Markup Language » (XML) est un langage de méta marquage qui fournit un format pour décrire des données structurées. Ceci facilite des déclarations plus précises de contenu et donne des résultats plus significatifs de recherche à travers les plates-formes multiples. De plus, XML permet une nouvelle façon d'opérer l'affichage et la gestion de données sur le WEB. Un langage de marquage est un mécanisme d'identification des structures dans un document. La spécification de XML [Wal98] définit une norme pour ajouter le marquage aux documents.

La puissance de XML [GP99] est liée au maintient de la séparation entre l'interface utilisateur et les données structurées. Le langage HTML spécifie comment afficher les données dans un moteur de recherche ou dans un certain site WEB. XML définit le contenu des données. XML sépare les données de la présentation et du processus, permettant l'affichage et le traitement des données selon les besoins de l'utilisateur avec plusieurs outils. Cette séparation des données de la présentation permet une intégration sans précédent des données des sources diverses. Des données encodées dans XML peuvent alors être fournies au-dessus du WEB au poste de travail. Aucune adaptation ultérieure n'est nécessaire pour valider l'information stockée dans des bases de données dans la mesure où le HTTP est employé pour livrer XML en ligne. Aucun changement n'est exigé pour cette fonction.

Dans XML, un ensemble illimité d'étiquettes peut être défini. Tandis que les étiquettes de HTML peuvent être employées pour afficher un mot en gras ou italique, XML fournit un cadre pour l'étiquetage des données structurées. Un élément de XML peut déclarer des données associées pouvant être un prix, une taxe de vente, une description composante, la tension d'alimentation ou n'importe quel autre paramètre ou élément d'informations désiré. La généralité de la construction à base d'étiquettes de XML facilite le partage et la transportabilité des modèles, et ouvre une capacité de manipulation des données indépendamment des applications dans lesquelles ils ont été crées. Une fois que des données ont été localisées, elles peuvent être fournies en ligne et être présentées dans un « WEB BROWSER » tel que « Internet Explorer » ou « Netscape ».

Les composants, les blocs et/ou les architectures écrites en utilisant XML sont qualifiés pour être trouvés et pour être consultés par les applications les plus récemment développées dans la gestion des bases de données orientées vers le partage par Internet. Actuellement, la plupart des fabricants de circuits électroniques offrent leurs produits en tant que propriété intellectuelle dans le

format XML. Cette voie assure la compatibilité des architectures générées avec HiLeS vers la gestion et l'affichage de documents structurés.

Ci-dessous (Tableau 1-6) une comparaison entre HTML et XML. Dans cet exemple, les informations correspondantes à des articles sont mises en forme de deux façons différentes. Avec HTML les étiquettes prédéfinies du langage permettent de construire une table qui contient l'information. Dans XML les champs de la base de données deviennent des étiquettes qui comportent l'information.

Tableau 1-6. Une comparaison des syntaxes HTML et XML

HTML	XML
<TABLE> **<TR>** **<TD>**Titre**</TD>** **<TD>**Organisme**</TD>** **<TD>**Date**</TD> </TR>** **<TR>** **<TD>**Microsystems**</TD>** **<TD>**LAAS-CNRS**</TD>** **<TD>**2002**</TD> </TR>** **<TR>** **<TD>**A380 wings**</TD>** **<TD>**Airbus France**</TD>** **<TD>**2003**</TD> </TR>** **<TR>** **<TD>**Optoelectronics**</TD>** **<TD>**LEN7**</TD>** **<TD>**2002**</TD> </TR>** **</TABLE>**	**<PUBLICATIONS>** **<ARTICLE>** **<TITRE>** Microsystems **</TITRE>** **<ORGANISME>** LAAS-CNRS **</ ORGANISME >** **<DATE>**2002**</DATE>** **</ARTICLE>** **<ARTICLE>** **<TITRE>** A380 wings **</TITRE>** **<ORGANISME>** Airbus France **</ ORGANISME >** **<DATE>**2003**</DATE>** **</ARTICLE>** **<ARTICLE>** **<TITRE>** Optoelectronics **</TITRE>** **<ORGANISME>** LEN7 **</ ORGANISME >** **<DATE>**2002**</DATE>** **</ARTICLE>** **</PUBLICATIONS>**

1.7.2 Les modules de Propriété Intellectuelle

En général, un module IP est une représentation d'un composant qui peut être intégrée dans un système supérieur. Un IP fini n'est pas modifiable. Toute modification a pour conséquence de créer une nouvelle version, un nouvel IP. La réutilisation des modules créés précédemment (Figure 1-5), permet aux concepteurs de structurer un modèle de système à haut-niveau à partir de leur IPs et d'autres développés par ailleurs.

Avec la création d'une plate-forme virtuelle d'IP, les modèles exécutables peuvent être partagés et évalués aux étapes préliminaires du processus de conception système. Des composants conçus peuvent être enregistrés dans des bibliothèques d'IP. Ces blocs d'IP ainsi que leurs « benchmarks », peuvent être empaquetés et partagés. Ceci facilite le test des IP pendant l'intégration du système ou pendant la réutilisation du composant. Des composants peuvent être simplement tirés des bibliothèques d'IP et sont facilement parcourues grâce aux étiquettes et descriptions structurées.

Les contraintes les plus importantes de l'utilisation des IPs sont:

- La gestion.
- La vérification.
- L'interconnexion (Compatibilité).

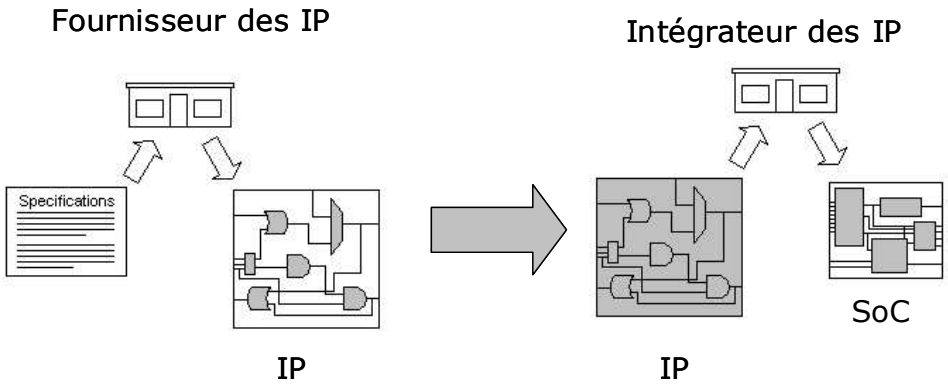


Figure 1-5. Elaboration et intégration des IPs [BS01]**.**

Notre démarche doit prévoir l'insertion de nos modèles dans un schéma de travail à base d'IP. Pour ce faire, nous assurerons, au moins, l'élaboration des modèles de haut-niveau en utilisant des langages standardisés, ainsi que la documentation des projets visant une participation dans des structures de partage de données. Nous ne développerons pas ce sujet, en revanche, nous ne négligeons pas l'intérêt d'une étude approfondie sur l'insertion de nos concepts dans une telle structure de travail.

1.8 Bilan : les outils, les langages et la conception système

Dans ce chapitre nous avons souligné notre volonté de centrer notre travail sur l'exploitation d'une démarche permettant de réaliser des descriptions de systèmes de haut-niveau. Les buts poursuivis sont une modélisation du système permettant la validation et la vérification formelle d'une part, et par la simulation d'autre part. Nous avons parcouru les différentes approches, en partant des outils de conception électronique, puis les méthodes de conception logicielle pour enfin aborder des approches orientées au niveau système. Nous avons constaté un croissant intérêt des différentes communautés pour uniformiser les méthodes et outils de conception en proposant des approches de haut niveau. Dans ce contexte « fédérateur » les équipes de l'ingénierie système ont déjà lancé la réflexion par le biais de la proposition de langages comme UML et plus particulièrement avec sa nouvelle version UML 2.0 et sa version spécialisée SysML. Malheureusement, la proposition de SysML est apparue vers la fin de nos travaux et à l'heure actuelle ces approches manquent encore d'operabilité, au fait, ils sont en cours d'établissement.

1.9 Conclusion

Ce premier chapitre définit tout d'abord le domaine de la Conception dans un contexte où l'innovation ne provient plus seulement des nouveautés en termes de matériaux ou de technologies : **Nous parlons d'une l'innovation de conception système.**

La conception et la simulation des systèmes électroniques ont été un argument fort de motivation pour le développement d'outils, de méthodes et langages permettant de gérer la complexité croissante des circuits et des systèmes. La CAO électronique, prise en compte à tous ses niveaux, industriel, académique et recherche, est devenue un moteur incontournable d'innovation et

de développement d'outils dédiés à la conception. Les objectifs sont toujours les mêmes, réduire le temps de production (« time to market »), anticiper les possibles sources d'erreur, réduire les coûts de fabrication, réaliser des prototypes virtuels les plus représentatifs de la réalité et en général, réaliser la conception sans faute.

Nonobstant, le haut-degré de spécialisation de la plupart des outils, le besoin de réduire le coût et le temps de fabrication a imposé la nécessité de trouver des solutions permettant d'avoir une vision globale des systèmes en prenant compte leur complexité et leur pluridisciplinarité. L'objectif est d'assurer la cohérence du système dès les étapes les plus en amont du processus de conception. Dans ce contexte, l'utilisation de langages standardisés s'impose comme une pratique incontournable. Le langage VHDL-AMS émerge comme le plus adapté car il comporte les caractéristiques nécessaires pour fournir aux concepteurs des systèmes complexes pluridisciplinaires les éléments pour leur modélisation. Cependant, VHDL-AMS n'est pas un langage de spécification. **Il est nécessaire donc, de le complémenter avec d'autres approches de niveau supérieur et ainsi construire une plate-forme complète de conception.**

L'intégration de langages de description système comme VHDL-AMS avec des techniques permettant d'appliquer des modèles formels tels que les réseaux de Petri et des langages de niveau de l'Ingénierie Système, ouvre la possibilité de construire des plate-formes de conception système capables de produire des modèles de haut-niveau permettant de représenter le fonctionnement des systèmes sans avoir besoin de rentrer dans les détails de leur fabrication mais en donnant la possibilité de les valider formellement et par simulation.

Malgré l'existence de dizaines d'outils CAO pour la conception des systèmes et leur flexibilité d'adaptation à plusieurs types de besoins et de problèmes, aucun parmi eux ne propose une vrai démarche méthodologique générale de haut-niveau. **La proposition des modèles « haut-niveau » dans les étapes amont de la définition des architectures est normalement faite par le biais de procédures non formelles et fréquemment empiriques**. Les représentations structurelles dépendent des préférences et de l'expérience technique du concepteur ou bien, elles sont faites à partir de schémas blocs, de diagrammes de flot de données ou d'architectures préétablies. Les outils qui proposent une approche méthodologique sont trop spécialisés et ne s'adressent qu'à des segments particuliers tels que la conception numérique.

Actuellement, une alternative générale semble se dégager. Elle est issue de la collaboration de plusieurs partenaires industriels au niveau mondial et elle est une conséquence directe de la progression imminente du langage UML vers l'ingénierie système, dépassant ainsi les frontières du génie logiciel : Il s'agit de SysML. Hélas, au moment du lancement de nos travaux de thèse, ce langage n'était pas encore disponible et n'est toujours pas suffisamment mûr.

CHAPITRE 2

2. HILES, UN FORMALISME POUR LA CONCEPTION SYSTEME DE HAUT-NIVEAU.

2.1 Introduction

Après avoir conclu aux principaux contours d'une approche globale de la conception système, dans le contexte des méthodes et des outils existants, nous présentons dans ce chapitre notre proposition **HiLeS**. Pour ce faire, nous commençons par un rappel des origines et motivations de cette proposition. Nous aborderons ensuite les spécifications de base de la représentation proposée, puis sa description détaillée. Nous discuterons, enfin, de sa pertinence pour la conception à haut niveau.

La modélisation au niveau système consiste à décrire ses fonctionnalités par des modèles abstraits. L'abstraction n'inclut aucune information sur la structure du matériel ou du logiciel qui permettra l'implémentation ultérieure du système [Jer02]. En partant d'un niveau d'abstraction élevé, notre démarche méthodologique doit d'abord considérer tous les éléments intervenants dans le processus afin d'établir l'organisation qui permettra de conduire les projets (Figure 2-1).

Nous proposons de construire cette organisation des éléments autour d'un **outil central** capable de fournir aux concepteurs les principes proposés par la méthodologie, et d'intégrer les modèles de calcul permettant de valider les prototypes dès les étapes les plus en amont. La démarche descendante sera utilisée comme démarche de base. **Cette approche « Top – down » doit proposer des prototypes à partir des spécifications générales**. Selon cette méthode descendante [Jer02], le système de base, le niveau « top » sera décomposé en sous systèmes qui seront eux-mêmes décomposés jusqu'a obtenir des modules simples programmables. Néanmoins, cette volonté descendante, n'exclut pas l'utilisation de connexions complémentaires « Bottom-Up », notamment pour l'intégration de modules externes (ré-utilisation), le raffinement des spécifications ou les phases de vérifications [Ham03].

La mise en place et la gestion de cet outil central doit s'appuyer sur une base de données informatiques permettant l'échange simple des informations entre les différents participants du projet, l'intégration des langages et la réutilisation des modèles précédemment validés. Cet outil sera amené aussi à soutenir des projets faisant intervenir plusieurs concepteurs

travaillant sur différents sites, distants les uns des autres. Pour cela, nous devons veiller à placer nos travaux dans le cadre des technologies récemment émergentes telles que l'ingénierie coopérative, la télé-conception et l'utilisation de XML comme langage de base pour la gestion des bases de données et des IP.

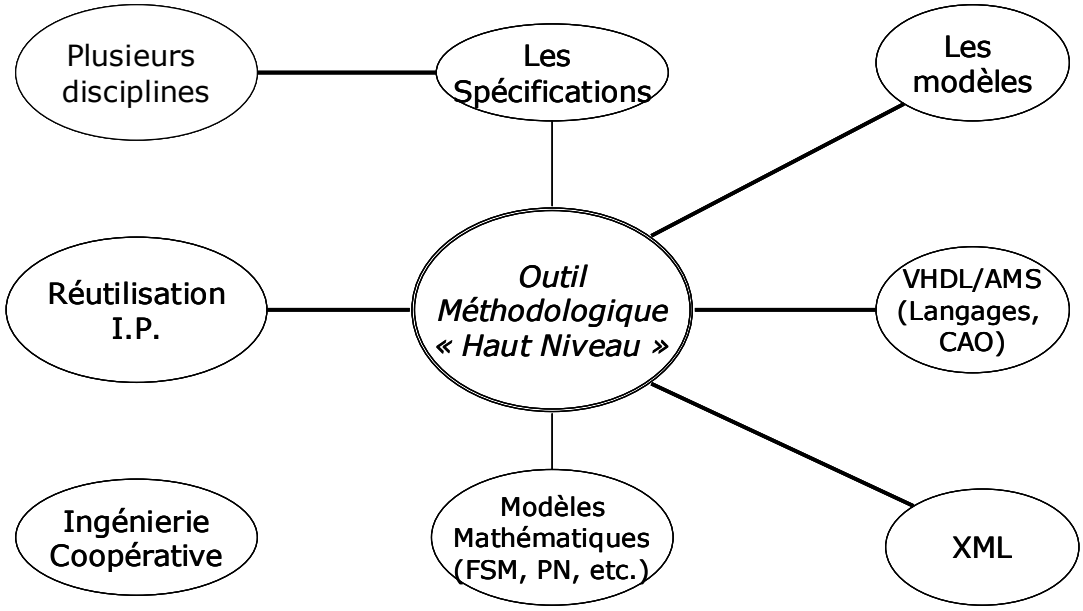


Figure 2-1. Organisation et gestion des acteurs du processus de conception « système » autour d'un outil méthodologique « haut-niveau »

Notre proposition méthodologique part des spécifications écrites en langage naturel pour les interpréter et les traduire sous la forme de modèles fonctionnels interconnectés. Pour cela, nous proposons de créer un outil informatique orienté vers la conception à haut-niveau. Cet outil devra permettre au concepteur **de mettre les spécifications du système sur un format graphique en appliquant une démarche de conception descendante**. Cette représentation graphique comportera plusieurs niveaux, selon une structuration hiérarchique. Une fois généré, ce modèle fonctionnel - graphique devra être validé ; l'architecture du prototype sera donc construite sur la base d'un modèle formel, permettant d'utiliser des méthodes ou des algorithmes de validation existants. Ensuite, le prototype pourra être interprété sous la forme de langages de modélisation et de description, afin de le rendre compatible avec les outils de simulation existants orientés vers le codesign. Il sera surtout interprété afin de pouvoir réaliser les simulations indispensables de façon comportementale et temporelle. Suite à l'inventaire des outils présentés dans le premier chapitre et sur la base des travaux antérieurs [Ham01], nous proposons cette approche originale HiLeS [Jim00] comme base de notre travail. Avec HiLeS, nous envisageons de combler la lacune existante entre les spécifications écrites sous la forme de documents de type texte et l'élaboration de représentations primaires, structurelles, fonctionnelles de haut-niveau, dans une implémentation générale mixte, associant du matériel (analogique et numérique) et du logiciel [HJA+00]. **Le but final du formalisme sera de fournir un modèle fonctionnel exécutable représentant toutes les spécifications du système.**

2.2 Historique de la démarche HiLeS

Notre projet s'inscrit dans un axe d'effort coopératif, qui vise à obtenir une représentation initiale des systèmes à concevoir avant d'engager le travail de matérialisation puis de fabrication. Le LEN7[ii] a consacré une partie importante de ses activités sur la conception électronique analogique et sur la conception numérique [CCE+99] avec l'outil [KC97] SyncCir. Le LAAS[iii], depuis sa création dans les années 60, s'intéresse à l'analyse de systèmes plus hétérogènes et pluridisciplinaires : automatique, robotique et microélectronique. L'émergence des micro-systèmes a conduit au lancement, à la fin des années 90, d'une action plus ambitieuse visant à définir et mettre en place une **méthodologie et des outils permettant la conception de micro-systèmes** en interne et en collaboration avec d'autres partenaires académiques et industriels.

Les origines plus spécifiques de ce cette action résultent par ailleurs, d'une concertation entre les équipes de conception électronique à Airbus France, et plusieurs équipes des laboratoires toulousains, du LAAS et du LEN7. Trois travaux de thèse, un stage de DEA et de multiples discussions internes sont venus nourrir notre projet :

- *Etude d'une méthodologie de conception descendante des micro systèmes : Conception d'un Micro Système pour la Surveillance de Contraintes Mécaniques en Aéronautique* , HARCHANI Noursaïd, Thèse doctorat, Institut National Polytechnique de Toulouse, France 2000. N HARCHANII est aujourd'hui ingénieur chez Altran Technologies.
- *Spécification et Conception de Micro Systèmes Basés sur des Circuits Asynchrones*, JIMENEZ Fernando. Thèse doctorat, Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS. Toulouse, France 2000. F. Jimenez est aujourd'hui professeur à l'Université des Andes, la collaboration est restée très forte avec cette université qui continue à collaborer activement dans le développement de l'outil HiLeS.
- *Une approche de co-design matériel / logiciel*, CLOUTE François, Thèse doctorat, Institut National Polytechnique de Toulouse, France 2001. F. CLOUTE est ingénieur à ST Microelectronics.
- *Plate-forme de Prototypage Virtuel, Conception Système et « Co-design » micro électronique*, HAMON Juan Carlos, Stage DEA CCMM, Institut National Polytechnique de Toulouse, 2001.
- Réunions de travail avec l'équipe « Avionics and Simulation Products » de AIRBUS France. La problématique de la conception système et de la convergence des modélisations détaillées et de haut-niveau sont au cœur des discussions au moment de la réalisation de nos travaux.
- Groupe régional de réflexion pluridisciplinaire TOOLSYS partageant la problématique de la conception système. Les activités de cette équipe sont consignées régulièrement dans le site web http://www.laas.fr/toolsys/.

Les premiers fondements théoriques de HiLeS ont été proposés par le LAAS–CNRS en 2000. HiLeS est un formalisme méthodologique de description et de spécification des systèmes à haut-niveau. Conçu comme une extension de la norme CCITT Z.100, HiLeS permet de spécifier et de décrire structurellement le comportement et la structure des systèmes dans une modélisation combinée de signaux continus, discrets, de fonctions et de signaux de commande. HiLeS peut être utilisé pour représenter des spécifications de haut-niveau ainsi que pour faire des descriptions de systèmes plus détaillées et riches.

[ii] LEN7 : Laboratoire d'électronique de l'ENSEEIHT, INP Toulouse

[iii] LAAS : Laboratoire d'Analyse et des Architectures du CNRS

Au niveau méthodologique, **HiLeS applique une démarche de conception descendante « Top-Down ». Les fonctions ainsi définies sont gérées temporellement sur la base de réseaux de Petri, SA/RT** [WM85] **et SDL** [DH97] (Specification and Description Langage). Cette particularité permet d'avoir une séparation claire entre les fonctions, les procédures et les instructions de commande de l'architecture.

L'approche initiale, proposée de manière théorique dans HiLeS, a été étudiée comme outil de spécification d'un micro système pour la Surveillance des Contraintes Mécaniques en Aéronautique [Har00]. Dans ce projet, la modélisation a été réalisée sur la base du Modèle à Evénements Discrets pour la Surveillance de Systèmes [Zam97] de l'ancien groupe OCSD (LAAS-CNRS). A partir de l'architecture générale de haut-niveau proposée par le modèle, des stratégies de commande du module superviseur et des fonctions du bloc diagnostic ont été générées. Le modèle de contrôle-commande réalisé pendant ce projet a été vérifié manuellement avant d'être transformé de façon non automatique en VHDL pour être simulé. Une fois la mise au point de la simulation achevée, les fonctions modélisées ont été implémentées sur FPGA.

2.3 Les spécifications de HiLeS

Pour répondre à la problématique exprimée dans le chapitre 1, nous avons besoin d'utiliser un formalisme ou une méthode qui, à partir des spécifications, couvre :

1. La conception amont,
2. Le prototypage virtuel.

Au niveau de la conception amont : Les exigences sont de convaincre, en amont, le maître d'ouvrage de la conformité des spécifications fonctionnelles avec ses propres spécifications textuelles initiales et de donner au chef de projet les éléments d'une structuration du travail aval pour toutes les fournitures. Ceci en prenant compte de deux objectifs fondamentaux :

- **La décomposition fonctionnelle du système :** Identification des fonctions principales du système et détermination de la façon dont elles doivent être connectées,
- **La hiérarchisation et la structuration architecturale :** La méthode choisie doit permettre de réaliser une décomposition hiérarchisée et structurée des fonctions de base du système : Cette étape est fondamentale car elle doit permettre de faire évoluer cette représentation vers une première hypothèse d'architecture pour le système réel.

Au niveau du prototypage virtuel notre démarche exige de proposer une « structure de travail » aux partenaires du projet, c'est à dire une architecture de fonctions du « système-produit », de manière à pouvoir recomposer ces fonctions en sous-ensembles spécifiques afin de les soumettre aux partenaires (internes ou fournisseurs externes). Ces derniers pourront alors faire des propositions de leurs fournitures. Cette structure doit aussi permettre d'arbitrer toute la démarche de consultation des fournisseurs par des procédures de validation visant à assurer la conformité aux spécifications, la conformité des décompositions ainsi que la conformité des propositions des fournitures. Pour remplir ces exigences nous proposons trois aspects principaux :

- **La construction d'une représentation formelle** des séquences de fonctionnement. Les modèles élaborés devront permettre de réaliser une vérification formelle de leur bon fonctionnement séquentiel : Les réseaux de Petri sont une solution attractive en proposant

une modélisation assez complète et puissante adaptée aux systèmes complexes avec la gestion du parallélisme et de la concurrence.

- **La compatibilité avec un langage standardisé** de description de systèmes pluridisciplinaires. Ce choix permettra de garantir la compatibilité et la transportabilité des modèles ainsi que leur pérennité. **La représentation finale des systèmes issus de notre démarche devra donc être simulable**. Le langage ouvert et standard VHDL-AMS est le plus indiqué pour écrire nos prototypes. Les modèles générés lors de l'application de notre démarche devront être les plus proches possibles de la norme de ce langage. Mais compte tenu des implémentations particulières des fournisseurs de simulateurs commerciaux, il faut prévoir des sorties pour des outils spécifiques si nous voulons réaliser des vérifications par simulation du comportement des modèles. Représenter fonctionnellement et globalement un « système-produit » peut poser un problème logistique lorsqu'il s'agit de réaliser une représentation de type pluridisciplinaire. Le projet européen TOOLSYS [Too99], auquel a participé le LAAS, illustre bien cette question dans un environnement automobile. Il a conclu à la nécessité d'un langage standardisé et unique pour toutes les disciplines : La proposition du langage VHDL-AMS [IEEE99] en 1999, correspond à une première étape qui pose les fondements d'une solution très attractive. L'adoption de VHDL-AMS comme langage de modélisation et de simulation globale n'interdit pas aux concepteurs des divers constituants du système, d'utiliser les outils spécifiques les mieux adaptés. **La proposition demande simplement que dans chaque discipline les modèles construits, trouvent le chemin d'une représentation VHDL-AMS afin de permettre la création d'une plate-forme commune de discussion et de simulation du « système-produit ».**

- Les informations complémentaires aux éléments constituants du modèle seront consignées dans une **fiche complémentaire** attaché à chaque bloc. Nous verrons par la suite l'importance opérationnelle de cette fiche dans toute la procédure de sélection et de choix des solutions terminales ainsi que dans la documentation du système, notamment pour les informations destinées à des instances liées à la conduite de projet.

2.4 HiLeS : proposition d'un formalisme

Nous reprenons ici une partie des éléments proposés lors de la définition de la première version théorique [Jim00] de HiLeS : Il s'agit des constituants de base qui composent le 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.

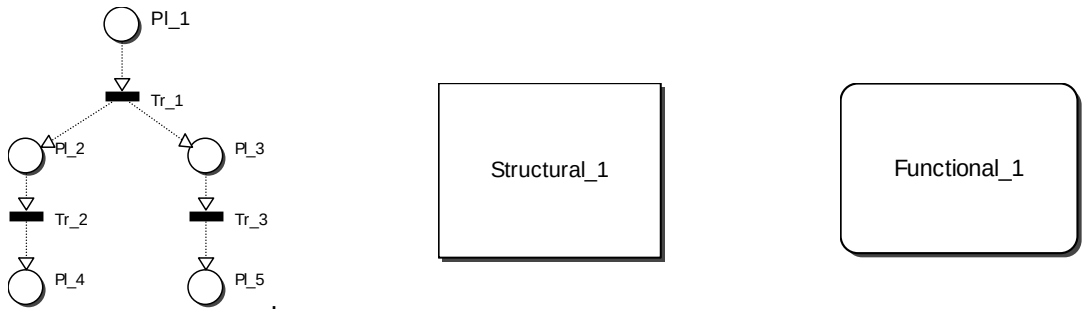
2.4.1 Les blocs

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

Les blocs structurels (Figure 2-2 b) sont représentés par des rectangles. Ils permettent la décomposition structurelle et hiérarchique du système. Ils admettent tous types de canaux d'entrée-sortie. Les blocs structurels peuvent contenir d'autres blocs structurels, des blocs fonctionnels ou des réseaux de commande hiérarchisés. L'utilisation de blocs structurels est bien caractérisée à deux niveaux différents du processus de conception :

- Au stade de la décomposition fonctionnelle où ils permettent d'établir des dépendances hiérarchiques et de décliner chaque fonction en plusieurs sous-ensembles,
- Au stade de « l'architecture » des fonctions où ils permettent de regrouper et d'agréger d'autres blocs afin de composer éventuellement des **« composants du système »**.

D'autre part, les blocs fonctionnels (Figure 2-2 c) sont représentés par des rectangles avec les coins arrondis : Ils décrivent le comportement du système sous la forme de systèmes d'équations différentielles, algébriques ou logiques. Dans l'état, la syntaxe adoptée est celle du langage VHDL-AMS. Les blocs fonctionnels ne possèdent pas des propriétés hiérarchiques.



a. Le réseau de contrôle b. Bloc structurel. c. Bloc fonctionnel

Figure 2-2 . Les éléments de base du formalisme HiLeS et son modèle de commande**.**

2.4.2 Le modèle de commande.

Le modèle de commande d'un système décrit par HiLeS est basé sur les réseaux de Petri ordinaires (Figure 2-2 a). Les réseaux de Petri sont un formalisme graphique et mathématique qui permet de décrire des systèmes dans lesquelles les notions de concurrence et parallélisme sont présentes. Il s'agit d'un quadruplet : *R=<P,T,Pre,Post>* dans lequel *P* est un ensemble fini de *places*, *T* est un ensemble fini de *transitions*, *Pre* est l'application *places* précédentes et Post est l'application places suivantes. Les places des réseaux de Petri peuvent être marquées par des jetons, l'ensemble de jetons dans les *places* du réseau indique le marquage du réseau. Un réseau de Petri est représenté par un graphe qui est construit à partir de deux types de nœuds : les *places* et les transitions, deux *places* ne peuvent être reliées directement et deux *transitions* non plus. Sous HiLeS, 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, tel que l'illustre la Figure 2-2 a. Les jetons utilisés ne sont pas colorés, indiquant qu'ils informent des activités événementielles mais qu'ils ne portent pas d'information.

2.4.3 Les canaux

Les blocs entre eux et leur réseau de contrôle sont reliés par des canaux. Les canaux transportent deux types de signaux: continus (Figure 2-3 a) et à événements discrets (Figure 2-3 b).

a. Canal continu b. Canal à événements discrets c. Canal discret type arc de réseau de Petri

Figure 2-3. Types de canaux définis dans HiLeS

Les canaux continus transportent des informations continues dans le temps, tel est le cas des signaux analogiques. Ils sont représentés par des flèches continues pleines. Les canaux à événements discrets sont de caractère logique. Plus précisément, ils peuvent être décrits par leurs valeurs vraies ou fausses : le dépassement d'un seuil, la détection d'une erreur ou bien l'arrivée d'un message parmi d'autres. Dans la sémantique HiLeS, nous représentons les « canaux discrets » par des flèches continues vides (Figure 2-3 b). Nous considérons les arcs des réseaux de Petri comme un type particulier de canal discret qui indique : L'arrivée d'un jeton à une place, l'initiation d'une fonction, la notification de fin d'une fonction vers une transition, ainsi que pour la connexion des blocs au réseau de commande. Ces canaux sont représentés, tel que l'illustre la Figure 2-3 c, par des flèches pointillées vides.

Les flots de données et de commandes sont indépendants et concourants (voir Figure 2-4). Les réseaux de Petri représentent l'exécution des commandes de contrôle et la synchronisation des blocs. Des canaux discrets du type arche de réseau de Petri entrant ou sortant du réseau sont utilisés pour représenter l'interaction de commandes et de données.

2.4.4 La gestion du temps et l'intégration de l'information et de la commande

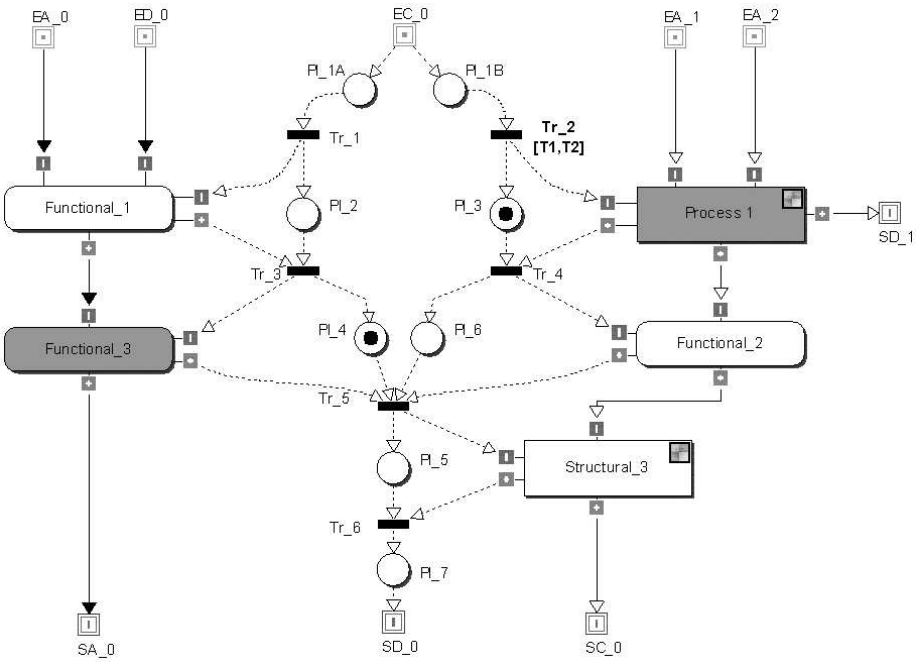


Figure 2-4. Intégration du réseau de commande et blocs dans une architecture HiLeS

Le temps d'exécution des blocs est associé au réseau en l'interprétant comme l'intervalle de tir de sa transition associée. C'est l'intervalle de temps dans laquelle la transition peut être franchie, à partir du moment ou toutes ses places aval sont marquées. Par exemple, dans la Figure 2-4, nous pouvons associer un intervalle de tir [T1 , T2] à la transition Tr_2 équivalent au temps d'exécution du Processus 1.

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.

2.4.5 La définition des hiérarchies

A partir des éléments, blocs et concepts décrits précédemment, on peut utiliser HiLeS pour faire une description de très haut-niveau du système. Les spécifications de la structure permettent de faire évoluer le modèle vers une description du fonctionnement du système (Figure 2-5). 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 une structure hiérarchique du système.

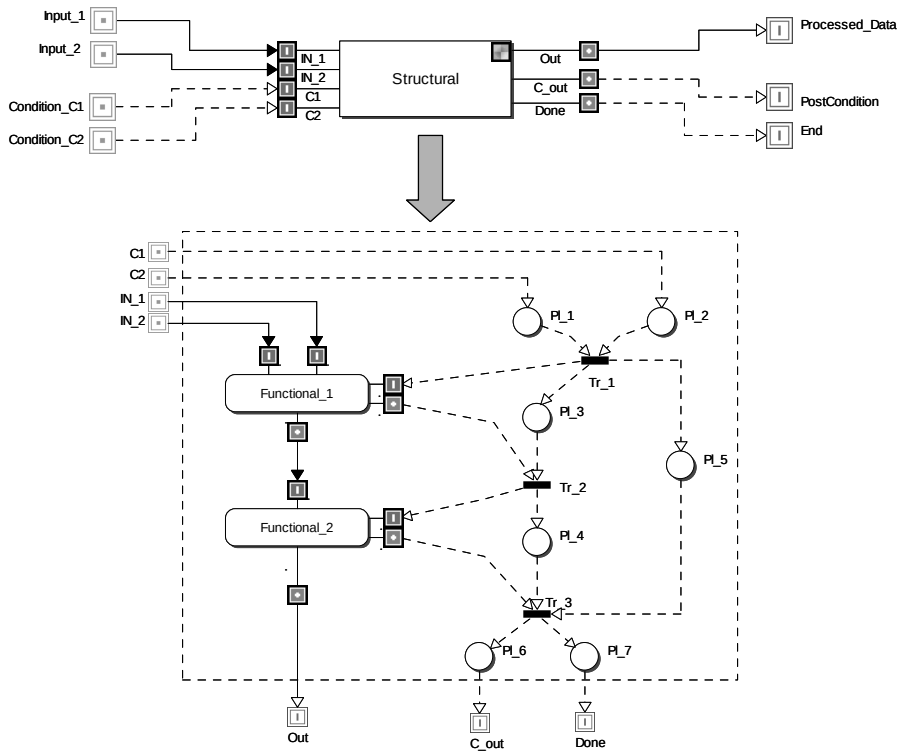


Figure 2-5. Illustration de l'évolution d'un modèle de blocs vers une structure fonctionnelle et temporisée [Ham01].

Avec les éléments mentionnés précédemment, HiLeS donne une description hiérarchique que l'on peut :

- Décomposer d'un bloc « père » en blocs « enfants »,
- Recomposer un bloc « père » à partir de plusieurs blocs « enfants ».

Chaque bloc est actionné par le réseau de Petri qui ordonne le fonctionnement du système (voir Figure 2-4). Nous avons identifié un composant élémentaire de synthèse, que nous pouvons recomposer avec d'autres en blocs de commande ou autre : **L'ensemble bloc, transition-place-transition**.

Dans notre démarche, cette approche est relativement commode, sur le plan de la conception, au moment ou l'on va se poser les questions initiales suivantes :

- Quelles sont les fonctions principales du système ?
- Comment sont-elles sollicitées ?
- Comment sont-elles interconnectées ?
- Quels sont les états fonctionnels du système ?

2.4.6 La représentation finale HiLeS

Un projet HiLeS, est une description graphique hiérarchique, fonctionnelle et structurelle d'un système. Les fonctions du système sont déclinées en plusieurs niveaux. Chaque niveau est construit à base des éléments propres du formalisme. La façon dont la modélisation est construite demande une visualisation interactive qui exige la participation de l'utilisateur, il doit « naviguer » dans le projet, littéralement plonger dans les blocs pour passer d'un niveau à l'autre.

Les contenus des blocs structurels peuvent être considérés comme des points de vue partiels. Chaque niveau local peut être une combinaison de blocs et de réseaux de Petri. A chaque niveau, les entrées et sorties provenant du niveau supérieur sont répertoriées. La Figure 2-6 illustre la structure générale d'une représentation HiLeS. Ce diagramme, inspiré d'un diagramme de clases UML, présente les rapports hiérarchiques dans HiLeS, ainsi que les relations entre ses éléments. Tant que dans UML, les losanges indiquent des rapports de composition. De cette manière, un projet (ou représentation) HiLeS est composé de blocs structurels, de blocs fonctionnels et de réseaux de Petri. Les blocs structurels sont alors composés de blocs structurels, de blocs fonctionnels et de réseaux de Petri. Les blocs fonctionnels ne sont pas déclinés. Finalement les réseaux de Petri interagissent avec les deux types de blocs, tel que l'illustrent les traits pointillés.

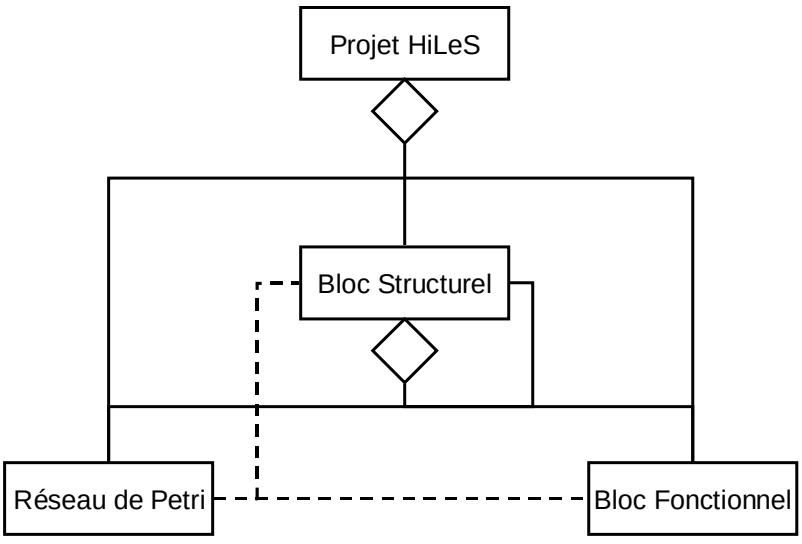


Figure 2-6 Structure des éléments d'une représentation HiLeS

Aller du plus général au particulier est une approche classique permettant de gérer la complexité des systèmes. On arrive, par cette procédure descendante (Figure 2-5), à une décomposition fonctionnelle qui peut être aussi détaillée que l'on veut, jusqu'à aboutir aux constituants les plus élémentaires. Selon les besoins de modélisation et l'état d'évolution de la conception du système, nous avons identifié trois types de représentations différentes mais néanmoins complémentaires. A terme, ces représentations seront **associées à HiLeS**.

2.4.7 Les représentations associées

Comme on vient de voir, les représentations HiLeS par niveaux de conception sont très riches. Nous verrons dans le chapitre 3 qu'elles permettent un premier niveau de vérification très intéressant. Cependant, cette représentation peut devenir très vite difficile de lecture pour le concepteur, et très certainement peut appropries pour certaines partenaires du projet. Il faut donc que notre sortie HiLeS propose d'autres formes de représentation compatibles entre elles. Nous suivons avec attention les réflexions en cours sur SysML™ dont on attend les résultats aux deux niveaux :

- Celui de la lecture structurale des spécifications textuelles.
- Celui, qui nous concerne, de la représentation des résultats de la formalisation de ces spécifications.

Il faudra, le moment venu, revenir sur cet aspect de la représentation, mais il y a trois types de ces représentations qui sont indispensables d'approfondir :

- **La représentation fonctionnelle hiérarchique :**

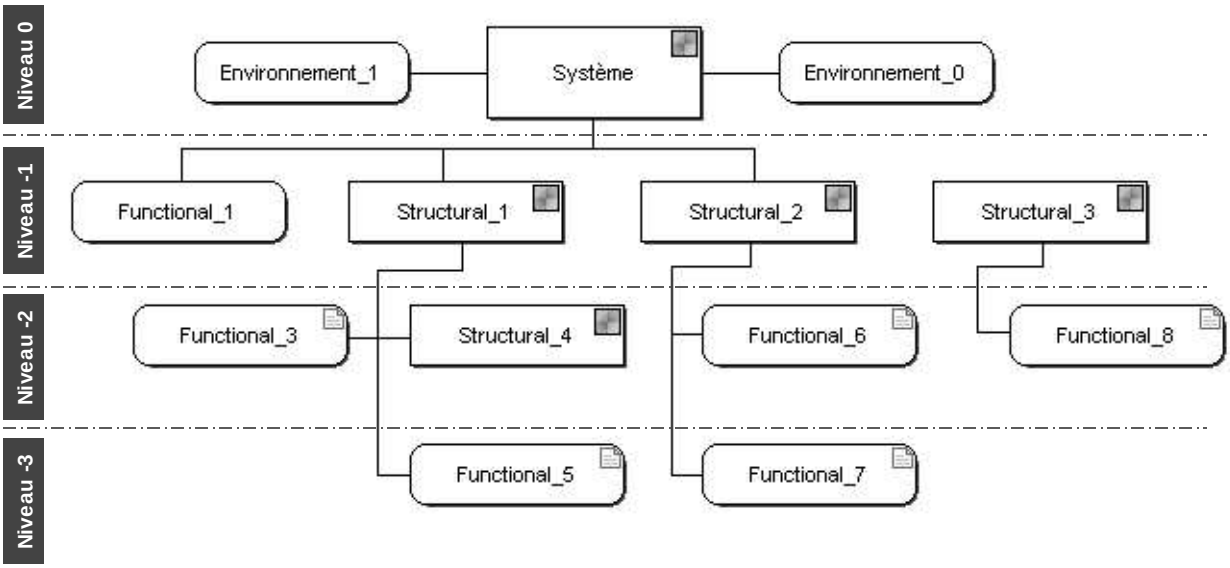


Figure 2-7. Décomposition fonctionnelle représentée sous la forme d'un arbre du système.

C'est le résultat de l'application de l'approche « Top-Down ». On oublie les caractéristiques temporelles de HiLeS pour s'attacher à la seule décomposition fonctionnelle. Les fonctions et leurs interconnexions sont représentées à différents niveaux de complexité. Ces niveaux peuvent être caractérisés par le concepteur pour avoir une idée du fonctionnement global du système. Elle possède l'intérêt de pouvoir présenter toutes les fonctions élémentaires et de servir de base pour une première visualisation architecturale. Nous pouvons considérer que cette représentation **purement relationnelle** facilitera les échanges entre les membres des équipes de conception et servira aux

instances les moins techniques d'appui pour une meilleure compréhension du système sans rentrer dans les détails.

La Figure 2-7 montre une représentation fonctionnelle hiérarchique. Les niveaux 0, -1, -2, etc., sont ceux de la décomposition hiérarchique du fonctionnement du système. Le concepteur peut estimer que sur cette forme brute toutes les caractéristiques structurales ne sont pas complètement illustrées. Il faut donc que HiLeS permette une certaine re-localisation hiérarchique des fonctions pour témoigner d'une cohérence technologique ou informationnelle.

- **La représentation architecturale :**

Elle répond aux insuffisances de la représentation précédente. On fait évoluer, **la représentation fonctionnelle du système vers une hypothèse d'architecture opérationnelle grâce à l'agrégation et au regroupement des blocs élémentaires en blocs plus complexes que l'on va pouvoir :**

  - Proposer selon une organisation de fonctions, dans la perspective de définir des composants plus conformes aux pratiques industrielles.
  - Définir et spécifier précisément toutes les entrées/sorties.
  - Spécifier comme une fourniture multi-fonctionnelle,

Déjà cette représentation architecturale, par sa cohérence technique et opérationnelle doit permettre de formuler des hypothèses sur l'implémentation du système et sur les éventuelles actions ou activités nécessaires pour la réaliser :

  - **Réutilisation :** Si le bloc ou composant a été réalisé et validé auparavant.
  - **Achat :** Si le bloc ou composant est un produit commercial.
  - **Création/conception :** Si le bloc ou composant n'existe pas ou s'il faut adapter un produit existant.
  - **Intégration :** Si le bloc ou composant peut être le résultat de l'assemblage de deux ou plusieurs blocs

- **La représentation « métier » :**

La représentation fonctionnelle est intéressante car elle apporte la liste des fonctions initiales du système en descendant jusqu'aux fonctions élémentaires ou jugées telles selon le niveau d'abstraction souhaité. Elle est l'outil de base du concepteur. L'étape de représentation structurelle précédente est indispensable car elle permet au concepteur d'exprimer au plus tôt une version cohérente du système guidée par son expertise des domaines technologiques qui pourront être envisages dans la matérialisation ultérieure. Mais, pour être utile à la dynamique de projet, il faut aller encore plus loin et pouvoir recomposer ces fonctions pour en **faire des blocs structurels correspondants aux métiers** que l'on souhaite solliciter. C'est à dire, correspondants aux domaines de compétence de chacun des fournisseurs pressentes dans la matérialisation. Cela suppose des découpages selon des partenariats stratégiques pour l'entreprise ou simplement des découpages techniques comme par exemple :

- Mesure et traitement de signal,
- Automatismes,
- Actionneurs,
- Interfaces de communication,

- Systèmes de surveillance,
- Autres.

Evidemment, cette recomposition peut être diverse puisque l'on a encore le choix des technologies et le choix des fournisseurs. Elle va conduire à plusieurs représentations concurrentes. **Des choix devront être réalisés basés sur des considérations des experts ou des expériences antérieures**. **Le plus naturel sera de consulter les fournisseurs** en faisant des lots que l'on spécifiera à partir des spécifications générales et de la modélisation HiLeS. Cette consultation peut induire différentes hypothèses et permettre des choix plus judicieux en prenant compte des données complémentaires telles que les coûts et les délais. **On voit que cette représentation « métier » débouche aussi sur des questions de conduite de projet ou l'on va construire des scénarios multiples et sélectionner celui qui répond le mieux aux exigences du projet**. Ces mécanismes de sélection se feront à travers des critères combinant des caractéristiques techniques ou autres : délais, coûts, encombrement. C'est le modèle partagé déjà envisagé par [BEY04], [Yac04] [LW98] pour des applications au niveau de l'industrie de la construction de bâtiments.

2.4.8 Une étape vers le prototypage virtuel : la compatibilité VHDL-AMS

La validation des propriétés liées aux représentations HiLes avec des réseaux de Petri ne suffit pas. Il reste la partie essentielle de la vérification par simulation, non seulement du réseau de contrôle mais aussi de l'ensemble de la modélisation système. Il est impératif de générer une représentation complète, pour exécuter une simulation complète du système, en utilisant une description comportementale et fonctionnelle y compris des algorithmes et des séquences logiques. Pour ce faire, nous proposons de choisir le langage de description VHDL-AMS comme le langage final de modélisation. Ce choix est justifié par le fait qu'il permet de décrire des systèmes pluridisciplinaires, grâce à l'application généralisée des lois de Kirchhoff. La vocation pluridisciplinaire de ce langage nous permet de faire face aux contraintes imposées par l'intégration de technologies multiples dans les systèmes modernes à base d'électronique. Depuis les applications purement électroniques, analogiques et numériques, jusqu'aux applications de type micro systèmes ou bien micro systèmes intelligents où l'on combine des sous-ensembles optiques, mécaniques, ou fluidiques avec des circuits électroniques analogiques, numériques et de plus en plus avec des modules logiciels, le tout intégré dans un seul objet : System On a Chip (SoC).

VHDL-AMS est l'approche [MM98] qui aujourd'hui réunit les mondes numérique et analogique dans une véritable démarche de conception système. Il est basé sur VHDL et en conséquence, il a déjà une définition précise de toutes les problématiques liées à un langage de programmation régulier (expressions, fonctions). En outre, il fournit des moyens pour structurer systèmes et communications. L'extension AMS couvre tous les concepts nécessaires pour une description complète des composants et des systèmes mixtes. La conception originale se présente comme une unification de la description numérique de matériel (VHDL) et la description analogique des circuits. Pour décrire le comportement du système, la norme a ajouté des dispositifs pour traiter des discontinuités, notamment l'instruction BREAK, qui permet de re-initialiser les différentes quantités et de relancer le noyau analogique du simulateur en produisant un nouveau point de simulation.

De plus VHDL-AMS est un standard IEEE, avec lequel nous assurons la pérennité et compatibilité des prototypes avec l'avantage [Her02] de proposer un langage commun indépendant des fournisseurs et de la technologie.

2.5 Une procédure de conception et de vérification « pas à pas »

Nous proposons de construire une procédure de conception « pas à pas » comme celle de la Figure 2-8. Il faut insister ici sur le fait qu'il n'existe pas de méthode rigoureuse, nous sommes dans le domaine de la « bonne pratique » : c'est avec cet état d'esprit qu'il faut considérer ces recommandations issues principalement des premiers exemples d'application conduits au laboratoire avec HiLeS Designer (c.f. §2.9).

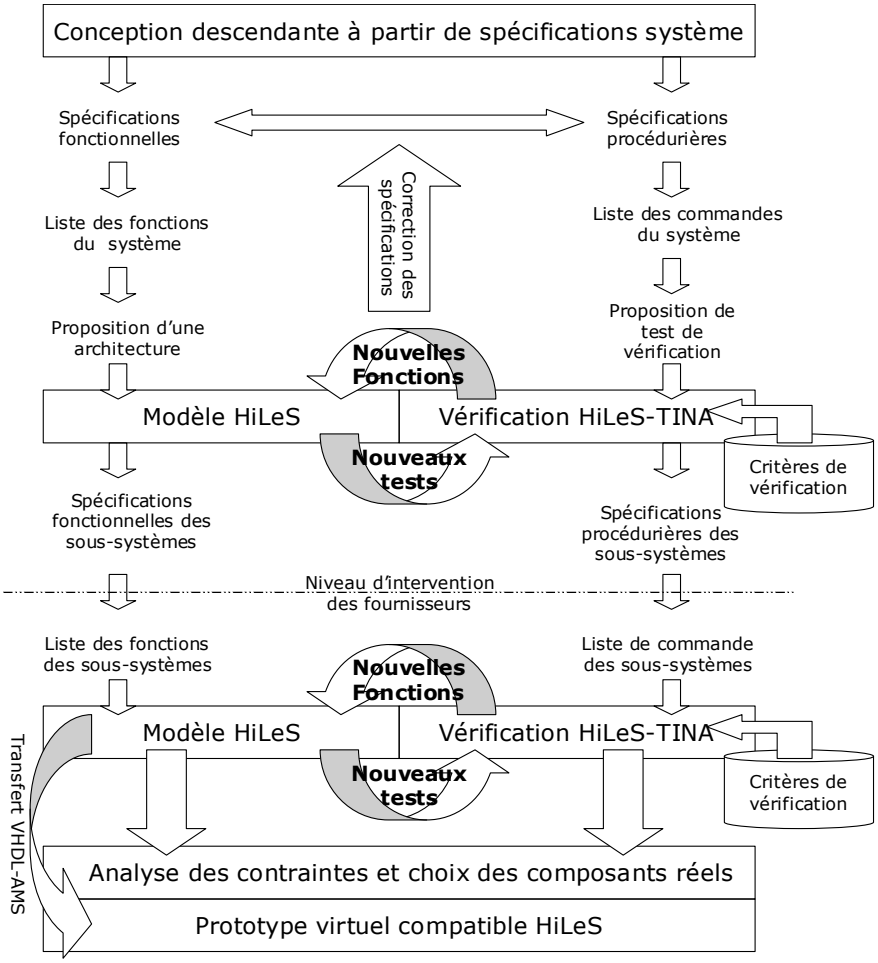


Figure 2-8 Schéma général de conception HiLeS

La première étape de lecture des spécifications permet d'accéder assez vite aux principales fonctions qui seront utiles pour commencer avec la bonne pratique d'établir un inventaire des fonctionnalités incluses dans le système.

Il faut ensuite établir les questions procédurières et les séquencements. Cela suppose d'identifier les différents états fonctionnels du système et de faire une première représentation des modalités et des transitions.

Ce qu'exprime la Figure 2-8 est que la démarche de construction structurale (à gauche) et de construction temporelle (à droite), doit s'opérer pas à pas, par des interactions successives et qu'il convient de réaliser ces deux approches de manière à :

- D'une part, se servir des fonctions pour proposer les modalités dynamiques complémentaires.
- D'autre part, se servir des séquences pour comprendre le comportement global du système et pour découvrir d'éventuelles fonctions manquantes.

Au stade de la modélisation HiLeS, nous sommes au carrefour des chemins et il faut, d'une part, retravailler les spécifications d'origine pour les compléter et les préciser, et d'une autre part, vérifier la cohérence de ces spécifications nouvelles.

Une démarche « pas à pas » reste efficace. Elle consiste à partir de la modélisation HiLeS, une fois réalisée, de solliciter les partenaires du projet, c'est à dire les fournisseurs. Pour ce faire, il convient de recomposer les blocs fonctionnels en blocs structurels conduisant à des tâches compatibles avec les fournisseurs potentiels. Notre recommandation ici est de **profiter de la description HiLeS pour proposer aux fournisseurs des spécifications formelles pour tous les sous-ensembles**. Plus encore, on peut aussi proposer d'accompagner ces spécifications d'un cadre de simulation HiLeS qui servira, en guise de « benchmark », à guider les fournisseurs dans ses propres choix.

Dans l'étape suivante, c'est le fournisseur qui va devoir apporter des réponses à ces spécifications. Nous recommandons qu'elles soient de deux natures :

- sous la forme de modèles compatibles VHDL-AMS pour permettre, au plus vite, de construire le prototype virtuel du système-produit complet,
- sous la forme de propositions matérielles chiffrées pour permettre l'analyse, l'arbitrage et la sélection de scénarios pour la conduite de projet.

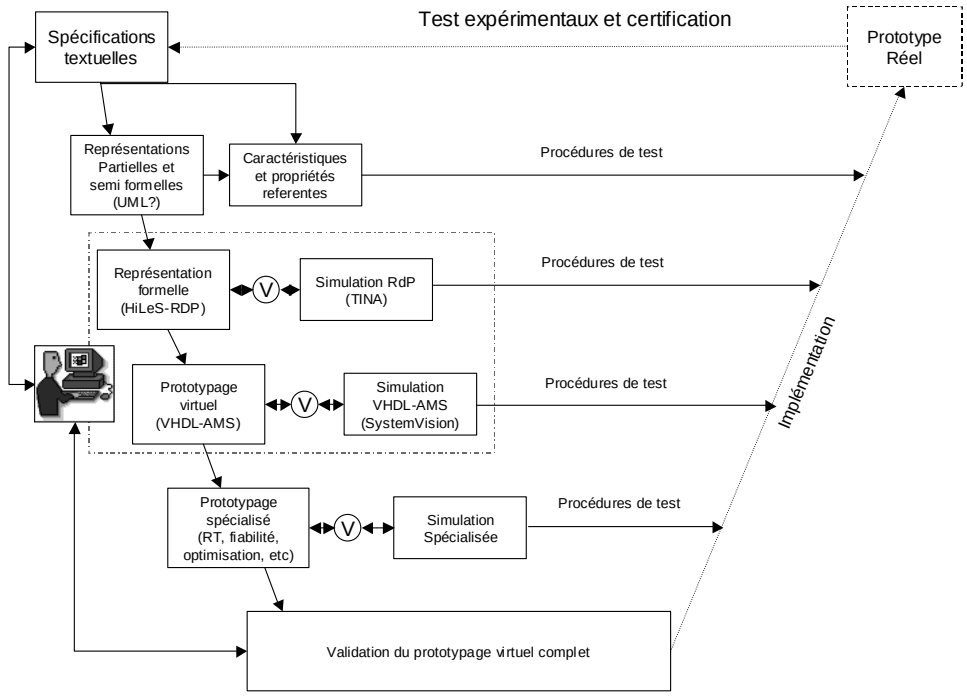


Figure 2-9. Interprétation de la démarche sous la forme d'un diagramme de conception en "V"

L'étape terminale, en ce qui concerne notre travail, est la simulation VHDL-AMS du produit. C'est aussi une étape de vérification essentielle puisqu'elle porte sur une représentation « fidèle » de

la réalité. Evidemment, plus il sera possible d'affiner cette représentation et de réaliser des simulations dans les conditions exactes d'environnement et d'utilisation, plus s'accumuleront des chances de succès « au premier coup ». Afin d'expliquer notre démarche dans un contexte de conception globale, nous l'avons illustrée sur le diagramme classique de conception en « V » (Figure 2-9). Notre domaine d'activité est délimité par le pointillé rouge.

Compte tenu de l'utilisation d'un modèle formel tel que les réseaux de Petri (RDP) pour la construction des représentations de HiLeS de haut-niveau, nous pouvons faire appel aux bonnes propriétés de ces RDP pour la vérification de certaines caractéristiques des systèmes modélisés. Malgré la validation syntaxique implicite dans HiLeS, la construction des réseaux de Petri reste sous la « responsabilité » du concepteur. La vérification structurelle du prototype exige la validation des réseaux de Petri associés. Face à cette contrainte nous avions deux options :

- soit implémenter un simulateur à l'intérieur de HiLeS,
- soit coupler l'interface graphique à un outil externe.

Nous avons retenu la seconde option. Pour cela, **nous avons choisi TINA [BRV03], un logiciel développé au LAAS-CNRS** par l'équipe de Bernard BERTHOMIEU au sein du groupe OLC[iv]. Il est capable de tourner en arrière plan et peut être lancé depuis notre outil central, HiLeS. Nous nous proposons de récupérer les résultats des analyses réalisées par TINA et de proposer au concepteur les premiers éléments de raffinement du projet. Nous rentrons dans les détails de cette vérification dans le chapitre 3 de ce mémoire. Une version complète et gratuite de TINA est disponible et téléchargeable [Ber02] sur le web. TINA est l'un des outils le plus avancés d'analyse des réseaux de Petri et l'on peut compter sur le support d'une équipe qui travaille sur son développement depuis plus de dix ans.

2.6 Capture et interprétation de spécifications

Nous voyons bien que le succès dépend d'abord de la qualité des spécifications. Il faut rassembler, au stade de l'écriture des spécifications, toutes les informations disponibles et les discuter avec toutes les compétences utiles qui peuvent être rassemblées de manière à réduire les « corrections remontantes » (illustrées dans la Figure 2-8 : en effet, toute modification des spécifications en cours de conception oblige à refaire une démarche descendante complète.

Pour aller des spécifications textuelles à une première représentation HiLeS, nous avons vu que trois volets doivent être distingués :

1. ***Un volet de description fonctionnelle représentative*** du système détaillant au mieux les fonctions que nous souhaitons voir apparaître dans le système avec les performances que nous en attendons.
2. ***Un volet de description des modes opératoires type*** duquel on dégagera des procédures de test qui serviront aux vérifications de conception.
3. ***Un volet de description des conditions d'utilisation*** duquel on dégagera les spécifications ergonomiques, sécuritaires, de fiabilité…

[iv] OLC : Outils et Logiciels pour la Communication

Normalement les spécifications du système sont écrites en langage naturel et sont issues de la réflexion d'une équipe d'ingénieurs système (ou projet) chargés de la définition du cahier des charges. Cette première étape du processus de conception doit aboutir techniquement à :

- Déterminer l'environnement opérationnel du système,
- Déterminer les acteurs qui interagissent avec le système et ceux qui en font partie,
- Identifier les requêtes et les réponses attendues du système,
- Etablir les séquences de base et les états fonctionnels du système,
- Avoir un aperçu général du fonctionnement global du système.

L'écriture des spécifications et du cahier des charges sont des étapes qui sont souvent relativement distantes du processus technique de conception. Toutes les techniques de conception système concernées par la réussite d'un projet considèrent que les besoins et objectifs sont complètement exprimés, extraits et interprétés et qu'ils sont correctement représentés. Néanmoins, dans certains cas, ils sont susceptibles d'interprétations subjectives de la part des concepteurs à chaque étape du développement des modèles de haut-niveau. En fait, beaucoup d'erreurs et de défauts constatés sur des produits finis ou sur des prototypes physiques sont dus à des défauts d'interprétation et à des omissions conceptuelles dans les premières étapes de conception du projet. L'utilisation de techniques formelles de spécifications pourrait aider les équipes de conception à surmonter ces problèmes. C'est pourquoi, des étapes de spécifications exécutables et vérifiables sont indispensables pour engager plus avant la modélisation détaillée du système, à condition, bien sûr, de pouvoir les relier aux documents de départ. Ceci est une condition incontournable pour pouvoir engager des procédures solides de traçabilité des exigences.

La majeure partie des travaux connus sur le développement des techniques de spécification de systèmes à haut-niveau a concerné la conception des logiciels et des systèmes numériques. Nous voulons ici explorer les approches possibles pour la conception de systèmes aux signaux mixtes et de systèmes plus généraux et pluridisciplinaires. En particulier, nous voudrions évaluer quel est le rôle de VHDL-AMS dans une stratégie de définition des spécifications exécutables: puisque VHDL-AMS est devenu un langage standard pour la description de systèmes mixtes, il convient d'étudier son intégration dans la démarche de conception. Le problème est que VHDL-AMS n'est pas un langage de spécification mais un langage de description de systèmes. **Il sera donc intéressant, comme nous le proposons dans HiLeS, d'élaborer une stratégie permettant de compléter VHDL-AMS avec d'autres langages et méthodes normalisées fonctionnant à un niveau de description plus élevé**.

2.6.1 De l'expression des besoins à une spécification exécutable

La première étape à réaliser dans le processus d'analyse des spécifications des systèmes est celle de recueillir, analyser et interpréter les exigences et les besoins de l'utilisateur. Typiquement, dans le contexte industriel, les clients ou les équipes spécialisées dans la définition des cahiers de charges, fournissent des spécifications écrites en langage naturel. Une fois que les besoins « utilisateur » sont définis, les concepteurs doivent les comprendre et les interpréter afin de proposer des solutions adaptées. Ce processus est souvent exposé à des interprétations subjectives, particulièrement si le document primaire contient des ambiguïtés ou des informations imprécises. Cette problématique demande de mettre en application des méthodes et des formats formels pour écrire ces « expressions des besoins » afin de les traduire en « spécifications exécutables » non-

ambiguës et formelles. Nous devons introduire cette première étape dans la classification habituelle [Jer02] des niveaux d'abstractions, pour la description de systèmes à haut-niveau :

- fonctionnelle: c'est une description des fonctionnalités du système sans rentrer dans les détails d'implémentation,
- comportementale: c'est une description basée sur les temps d'exécution et les séquences de fonctionnement,
- structurale: c'est une description au niveau hiérarchique de l'implémentation et des composantes du système,
- « Boite noire » : description du comportement des communications externes d'un composant sans rentrer dans le détail de ses fonctionnalités internes.

Les techniques actuelles d'écriture des spécifications sont plutôt orientées vers la gestion de l'information et le suivi des besoins exprimés dans les documents. Ces méthodes visent à assurer un suivi parfait de la déclinaison de chaque besoin en une ou plusieurs solutions; ceci signifie que la documentation du projet et la traçabilité des exigences sont prioritaires. Ces prérogatives amènent à définir des règles de construction pour les documents qui permettront leur intégration dans des politiques rigoureuses de gestion des données. **Une norme internationale actuellement utilisée est la EIA632 [Mar04]. Cette règle est employée comme base pour l'étiquetage des besoins dans les cahiers des charges**, ainsi que pour le suivi de leur déclinaison dans des solutions diverses, tout au long du processus de conception et de fabrication. En utilisant de telles techniques, en plus de la documentation du projet, l'expression des besoins sous la forme de texte doit révéler des informations indispensables pour construire les spécifications de système. L'interprétation de l'expression des besoins doit permettre d'identifier et de classer les fonctions principales du système, les paramètres, les performances attendues et les procédures opérationnelles prévues.

Une fois les spécifications écrites, il est nécessaire, dans notre proposition, de les transformer pour obtenir une première représentation de haut-niveau : les spécifications exécutables. Ces spécifications exécutables doivent être basées sur des modèles formels permettant de vérifier leur cohérence avec des spécifications de type texte. En plus, des spécifications exécutables bien-structurées et lisibles faciliteront le dialogue entre les partenaires du projet, les clients et les fournisseurs de composants et de services.

On peut être tenté de développer une technique automatique pour extraire et interpréter l'information à partir des spécifications exprimées en langage naturel. **Ces premières analyses montrent que cela implique la définition d'une guide d'écriture contenant :**

- un sous-ensemble restreint et rigoureux de termes du langage naturel,
- des règles d'utilisation précises de ces sous-ensembles de langage,
- un format précis de la représentation visée.

Un exemple de guide d'écriture est PENG [Sch03]. Il s'appuie sur un sous-ensemble strict et limité de l'anglais, traitable par ordinateur et conçu pour écrire des spécifications non ambiguës et précises ainsi que des cas d'utilisation. Ce genre d'outils et de méthodes peut aider à identifier les fonctions principales dans un texte, ainsi que les procédures de fonctionnement, les entrées et les sorties du système, et même certaines séquences de tests pourraient être formellement exprimées par des spécifications exécutables. Une fois que l'information est obtenue, il est nécessaire de

l'organiser et de la présenter avec certains critères qui la rendront facile à lire et à partager entre tous les partenaires du projet.

2.6.2 Représentation et vérification de l'interprétation des besoins

Les résultats de l'analyse et de l'interprétation de l'expression des besoins, doivent être employées pour construire une première représentation des spécifications. Ces modèles peuvent décrire les besoins d'une manière très générale et devenir la première étape d'un processus plus détaillé et plus formel de description. A cet instant, il est urgent de proposer une méthode générale pour traduire les données interprétées en une représentation standardisée. Une proposition dans ce sens, issue de l'ingénierie logicielle, est en train de devenir une pratique généralisée : le langage UML.(§1.6)

Etant donné le besoin d'un langage général standardisé pour la conception système, de la maturité d'UML et de la grande quantité d'outils logiciels qui mettent en œuvre ce langage, un consortium réunissant entre autres, Boeing, NASA-JPL, BAE, Astrium Space, a été crée dans l'intention **d'étendre UML (§1.6.2) à une approche générale « Ingénierie Système »**. Cette initiative intéresse non seulement l'ingénierie logicielle mais aussi l'ingénierie système en général. L'objectif principal de l'équipe de travail est de définir un langage général, orienté système, capable de prendre en compte l'expression des besoins, les performances attendues, le comportement prévu et les représentations structurelles des systèmes. Ce consortium propose un sous-ensemble particulier d'UML appelé SysML [SP04] dont une première version de proposition était attendue pour août 2004.

Malheureusement, jusqu'ici, il faut préciser que les versions officielles d'UML ne proposent pas de méthodologie pour utiliser ces diagrammes. Elles sont limitées à la définition de leur syntaxe. Ceci signifie que les utilisateurs d'UML doivent encore établir une méthodologie claire qui définirait comment ces diagrammes seront utilisés. UML seul donc ne suffit pas à résoudre le problème de la représentation de systèmes à haut-niveau. Les diagrammes d'UML devront être vérifiés sous des modélisations formelles telles que les machines à états finis (FSM) ou les réseaux de Petri. Des travaux récents [Del03] ont déjà exploré la possibilité d'obtenir des modèles formels à partir des représentations semi-formelles UML. Leur proposition est d'introduire des réseaux de Petri à partir des diagrammes de séquence UML. Une telle initiative sera très utile pour ajouter à la description, les étapes de validation et de vérification formelles à un niveau d'abstraction très élevé, comme garantie de la conformité avec les spécifications exprimées sous forme de texte.

D'autres outils et approches existantes comme le Esterel Studio, ROSETTA, Cofluent Studio ou HiLeS Designer proposent aussi des approches au problème des spécifications à haut niveau des cahiers des charges ; HiLeS, que nous proposons, est une approche qui intègre trois caractéristiques principales :

- **description graphique :** la représentation des systèmes en utilisant des blocs structuraux et fonctionnels interconnectés selon une décomposition hiérarchique du système et une description fonctionnelle détaillée,
- **vérification formelle :** les réseaux de Petri combinés au comportement du système exprimé par les blocs, les fonctions, la temporisation de la structure contrôlant les instructions et l'exécution des ordres et les activités du modèle,
- **transformation du modèle en VHDL-AMS :** une représentation complète du système est produite du formalisme de HiLeS vers le langage de description VHDL-AMS.

Même si des approches comme HiLeS semblent couvrir la totalité de la problématique de conception système à haut-niveau, l'exigence d'échange imposera probablement, à terme, une certaine compatibilité avec les normes et standards de type UML ou SysML (§1.6.2)**.**

2.6.3 Recommandations sur les développements ultérieurs

Les développements actuels encouragent la proposition d'une stratégie complète et standardisée de spécifications à haut-niveau à partir des documents d'expression des besoins et des cahiers des charges vers l'élaboration de spécifications exécutables. Pour ce faire, nous avons identifié trois besoins principaux représentés sur la Figure 2-10.

- **« Une guide d'écriture »** pour aider les concepteurs à écrire correctement, d'une manière structurée et selon un format prédéfini les documents d'expression de besoins et de spécifications.

- **« Une guide d'interprétation et de lecture »** afin d'interpréter les spécifications comme une première étape dans le développement de modèles formels. Les solutions basées sur UML devraient apporter une première étape satisfaisante vue l'importance d'une approche standardisée d'usage universel que l'on retrouvera probablement dans SysML.

- **Des approches formelles telles que celle proposée par HiLeS, mais compatibles au standard UML** peuvent permettre la vérification de cette représentation des spécifications. Actuellement cette validation formelle n'est pas possible directement sur UML, car le langage ne propose pas de formalisme mathématique associé, une solution possible pourrait être proposée par UML2. Une représentation exécutable finale en VHDL-AMS permettra de compléter la définition des premiers modèles du système à haut-niveau. Des travaux récents [HF04] proposent d'inclure dans la norme VHDL-AMS des nouveaux opérateurs permettant d'exprimer des contraintes ou de vérifier en cours de simulation des conditions particulières imposées par les spécifications.

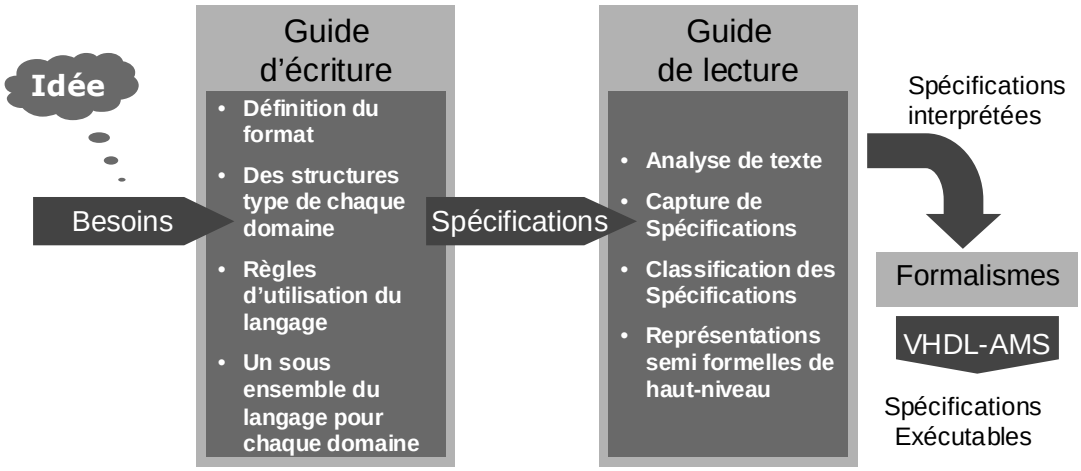


Figure 2-10 Un aperçu d'une procédure idéale de génération de spécifications exécutables.

Le déploiement des besoins interprétés selon un mécanisme de spécifications exécutables peut aider les équipes de conception à identifier quelques paramètres, conditions ou procédures qui n'ont pas été considérés dans les documents initiaux de l'expression des besoins. Ils pourront ainsi les compléter : il est essentiel de garantir une parfaite concordance entre les spécifications exécutables et les spécifications textuelles.

2.7 Une proposition pour la gestion de l'information

Dans l'approche que nous envisageons, la façon de gérer l'information est cruciale. La performance des outils informatiques et des méthodes de conception est fortement liées aux données disponibles et aux procédés de stockage et de transfert de ces données. Le principe de la conception pluridisciplinaire implique déjà le partage des informations entre plusieurs concepteurs de plusieurs domaines et éventuellement de plusieurs organismes. De plus, la réutilisation des savoir-faire rend nécessaire l'implémentation de techniques adaptées à la gestion des fichiers contenant les anciens modèles, à l'élaboration des nouveaux et à la combinaison des modèles vis-à-vis de la création des prototypes virtuels.

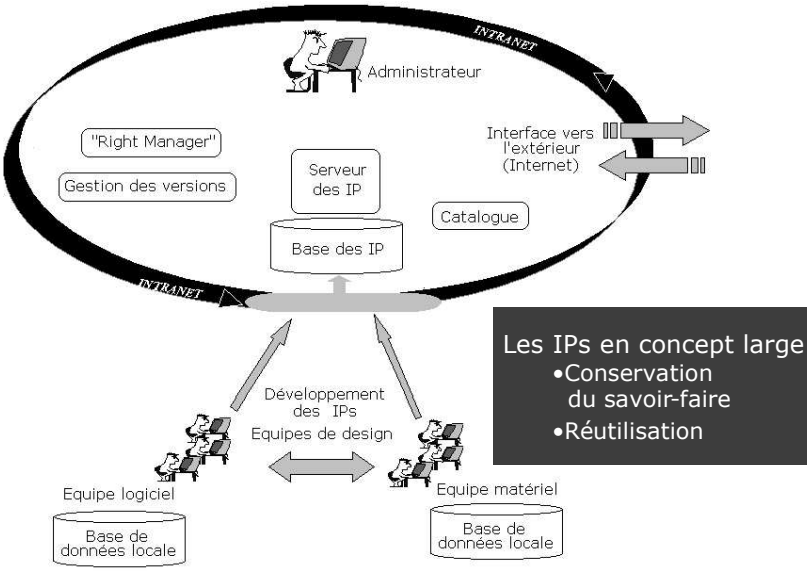


Figure 2-11. Modèle de gestion de l'information, basé sur les outils [DAR02] fournis par la Société Design and Reuse.

Depuis le début de nos activités, nous avons été particulièrement attentifs à ce problème que nous considérons comme prioritaire pour l'application d'une démarche telle que la notre. Compte tenu du fait que les objectifs de notre thèse ne couvrent pas l'élaboration de ces outils gestionnaires de l'information, nous avons pris la décision d'utiliser des logiciels commerciaux. Ce choix se justifie aussi dans le contexte d'un travail voulu le plus ouvert possible. Pour cela, nous avons pris contact avec la Société « Design & Reuse » de Grenoble. Parmi les produits de cette compagnie nous avons utilisé IP Series Manager [DAR02]. Notre idée est à terme, d'élaborer nos prototypes virtuels sous la forme de modules de propriété intellectuelle (IP) même si ces modèles restent en interne aux partenaires du projet. L'intérêt de cette orientation est de devoir écrire les modèles de façon structurée vis-à-vis de leur réutilisation, et de garder la possibilité d'insérer, dans nos projets des modèles développés par d'autres équipes suivant le même principe. De la même manière, nous pouvons ainsi disposer d'une méthode de gestion de bases de données et d'un gestionnaire de versions de nos prototypes. Dans la

configuration que nous avons convenu, la Société Design & Reuse n'a pas accès aux modèles car son rôle est seulement de fournir les outils de gestion de données. Le modèle de gestion (Figure 2-11) propose la centralisation de l'administration des bases de données sous la forme d'un catalogue es IPs internes disponible aux concepteurs via l'intranet du laboratoire. L'outil comporte également le gestionnaire des versions et des droits d'accès ainsi que les ressources d'administration. Les concepteurs pourront s'inscrire en tant qu'utilisateurs pour télécharger ou déposer des IPs.

Nous avons prévu de compléter les informations techniques de chaque projet consignées dans la base de données d'IP avec **des fiches complémentaires dont leur première version est provisoirement implémentée (c.f. § 2.9.1).** Ces fiches doivent accompagner le projet dans toute sa dimension de conception et de conduite de projet. En l'état, ces fiches vont accompagner les blocs structurels et fonctionnels HiLeS. Une réflexion déjà très approfondie de ces fiches existe [[GO04]] pour définir les contenus et les modalités XML d'échange. Cette réflexion sur le concept de modèle partagé est issue du groupe de discussion TOOLSYS (www.laas.fr/toolsys) et participe d'un projet interne coopératif LAAS (Annexe A), qui sera développé en collaboration avec les équipes de ISI[v] et du LESIA[vi] de l'INSA de Toulouse.

2.8 Positionnement des propositions HiLeS dans le contexte technique général

Afin de vérifier la cohérence de nos propositions dans le contexte de la conception des systèmes avioniques, nous nous sommes attachés à nous situer par rapport aux conclusions [Enh02] du projet ENHANCE :

ENHANCE est l'acronyme d'un projet intitulé « ENHanced AeroNautical Concurrent Engineering ». C'était un projet de recherche de 38 M€, dont 50% ont été financés par la Commission Européenne. Le projet a été piloté par EADS avec la participation de la plupart de ses partenaires habituels y compris les PME. Le projet à commencé en février 1999 et s'est terminé en avril 2002.

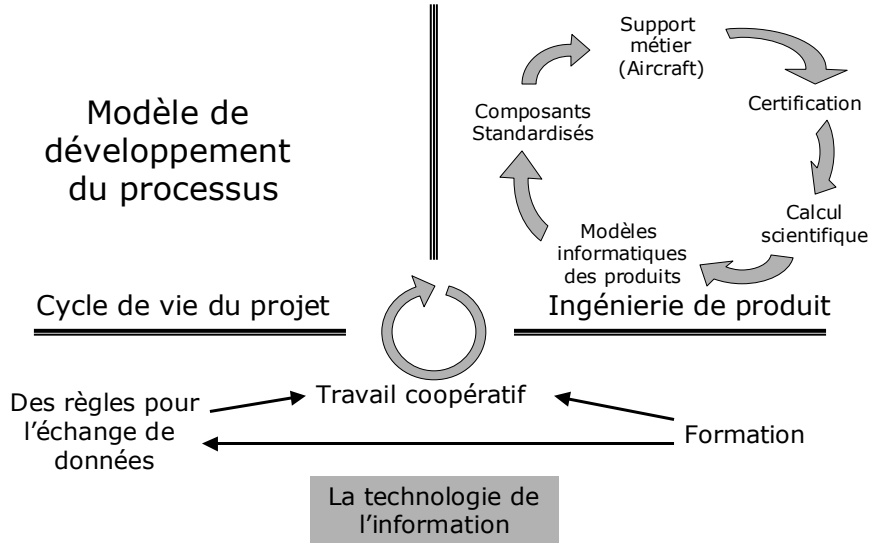


Figure 2-12. Contexte technique [Bra02] proposé dans les conclusions du projet ENHANCE pour la conception des systèmes aéronautiques.

[v] Ingénierie Système et Intégration

[vi] Laboratoire d'Etude des Systèmes Informatiques et Automatiques

Dans cette thèse, nous avons déjà fait référence à quelques éléments d'origine ENHANCE [Figure 1-1]. Nous nous occupons ici plutôt du contexte technique général, des outils et des méthodes de conception système.

La dynamique du processus de conception système, est considéré par les interactions de trois grands domaines (voir Figure 2-12) : le cycle de vie du projet, le génie du produit et la technologie.

Le premier élément ou « Cycle de vie du projet », concerne la partie « management » et les modèles du processus de production. Le deuxième élément, l'ingénierie du produit, est la partie « technique » du projet. Nous y trouvons les modèles et les méthodes de conception ainsi que les règles et les standards propres du domaine de l'aéronautique. Le troisième et dernier élément est la base de cette dynamique car de la technologie de l'information dépend l'efficacité des interactions entre les domaines de la Gestion, la production et la dimension humaine de tout projet. Les méthodes utilisées pour gérer l'information peuvent avoir des conséquences dans toute la structure et l'organisation de l'entreprise. Toute nouvelle implémentation ou amélioration doit être soigneusement analysée afin de minimiser l'impact sur la formation du personnel et de respecter les standards de communication. Dans le projet ENHANCE, nous retenons aussi l'intérêt porté au télé-travail comme un outil approprié pour la conception distribuée des systèmes complexes, en permettant la participation des partenaires experts situés dans différentes usines et différents pays.

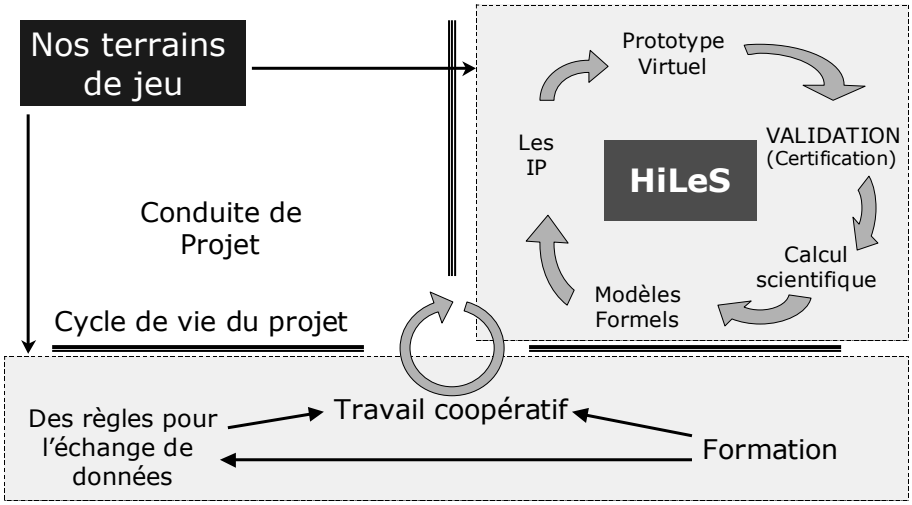


Figure 2-13. Le contexte technique de notre démarche.

Notre analyse des propositions du projet ENHANCE montre que l'on doit inclure de façon cohérente notre démarche sur le même type de schéma que celui de la Figure 2-12. Notre travail n'inclut évidemment pas la partie gestion de projet, car notre intérêt privilégie les éléments techniques de la conception système et de la gestion de l'information. Pour préciser cela, nous avons indiqué sur la Figure 2-13 ce que nous avons appelé notre « terrain de jeu ». Comme nous l'avons proposé sur le schéma de la Figure 2-1 nous proposons de construire notre démarche de conception autour de l'outil HiLeS Designer.

Notre démarche devra donc s'inscrire dans une démarche générale plate-forme fortement supportée sur les techniques de gestion de l'information qui incluent le travail coopératif ou télé-travail, dans un espace : conception de produit, conduite de projet, formation. Ces trois dimensions du problème ne sont pas développées dans ce document qui reste centré sur le seul objet de la

conception. Pour la partie conduite de projet, nous renvoyons à d'autres travaux auxquels nous avons été associés et qui développent justement l'idée d'une modélisation partagée entre Conception Système et Conduite de Projet [BEY04]. Pour la partie formation, elle a occupé une partie importante de notre activité d'animation. Nous avons, en effet, sur le thème de la conception amont et du prototypage virtuel organisé plusieurs cycles de formation réalises par le professeur Yannick. HERVE, et des conférences spécialisées dont celles réalisées par Alain VACHOUX, Masahiro FUJITA et Jean-Jacques CHARLOT.

2.9 Mise en œuvre de l'outil HiLeS Designer

Les rappels et recommandations présentés tout au long de la première partie de ce chapitre doivent être mis en œuvre sous la forme d'un outil permettant de les utiliser et de les valider sur des applications concrètes. Nous avons choisi de développer cet outil autour du formalisme HiLeS. Cet outil comporte les fonctionnalités suivantes :

- Un moteur de gestion "Top-Down" hiérarchique.
- Un gestionnaire pour la définition des multiples expressions d'une même architecture.Un moteur à réseaux de Petri avec :
    - Editeur graphique.
    - Mécanisme de vérification des réseaux de Petri.
- Un moteur VHDL-AMS :Compatibilité structurale entre HiLeS et VHDL-AMS.
    - Editeur VHDL-AMS avec détection automatique des mots clés du langage.
    - Génération semi-automatique de code
- Des fiches d'information : "Complementary Data Files" (CDF) pour enregistrer des informations complémentaires dites « non fonctionnelles » du système.

L'outil HiLeS Designer 0 que nous allons décrire est la première implémentation logicielle opérationnelle du formalisme HiLeS. Il permet aux concepteurs d'écrire leurs projets dans un environnement informatique convivial qui fournit des éléments de HiLeS et beaucoup d'autres fonctionnalités. HiLeS Designer a été créé en tenant compte des principes du formalisme et des caractéristiques du langage VHDL-AMS. Ce langage impose certaines contraintes mais facilite la construction du modèle complet. L'interface graphique de HiLeS Designer est facile à utiliser comme la majorité des interfaces graphiques d'utilisateur (GUI) pour PC. Tous les objets sont disponibles explicitement dans la barre d'outils, et leur placement dans la feuille de conception est aussi simple que "Drag and Drop".

Dans le schéma de la **Erreur ! Source du renvoi introuvable.** nous présentons un récapitulatif des principaux éléments fonctionnels de l'outil. Le diagramme illustre les modules de base que nous avons considéré pour cette première étape de développement, chaque bloc corresponde aussi aux différentes étapes de réalisation qui nous avons accompli. En vert les fonctionnalités déjà implémentées, en jaune celles que nous sommes en train de développer. Nous avons commencé par établir les bases avec la mise en place de la « sémantique » HiLeS, les blocs, les canaux et les réseaux de Petri. En suite nous avons intégré les propriétés hiérarchiques de la modélisation avec la gestion de niveaux de description des blocs. Nous avons développé aussi la possibilité d'associer plusieurs descriptions à chacun des blocs structurels et d'en choisir une pour générer le modèle total. Par rapport aux mécanismes de sortie des projets, nous avons mis en place des interpréteurs permettant de générer des représentations équivalents : des netlists pour les réseaux de Petri et du code VHDL-AMS pour la totalité du projet. Actuellement, nous continuons de travailler sur les modules

jaunes. D'une part, en collaboration avec l'équipe OLC du LAAS sur la vérification des propriétés des systèmes associées aux réseaux de Petri utilises dans la représentation HiLeS. D'autre part, nous participons à une activité menée par nos collègues de l'équipe ISI sur la définition des paramètres non fonctionnels nécessaires pour compléter la modélisation et permettre le couplage de la conception à la conduite de projet.

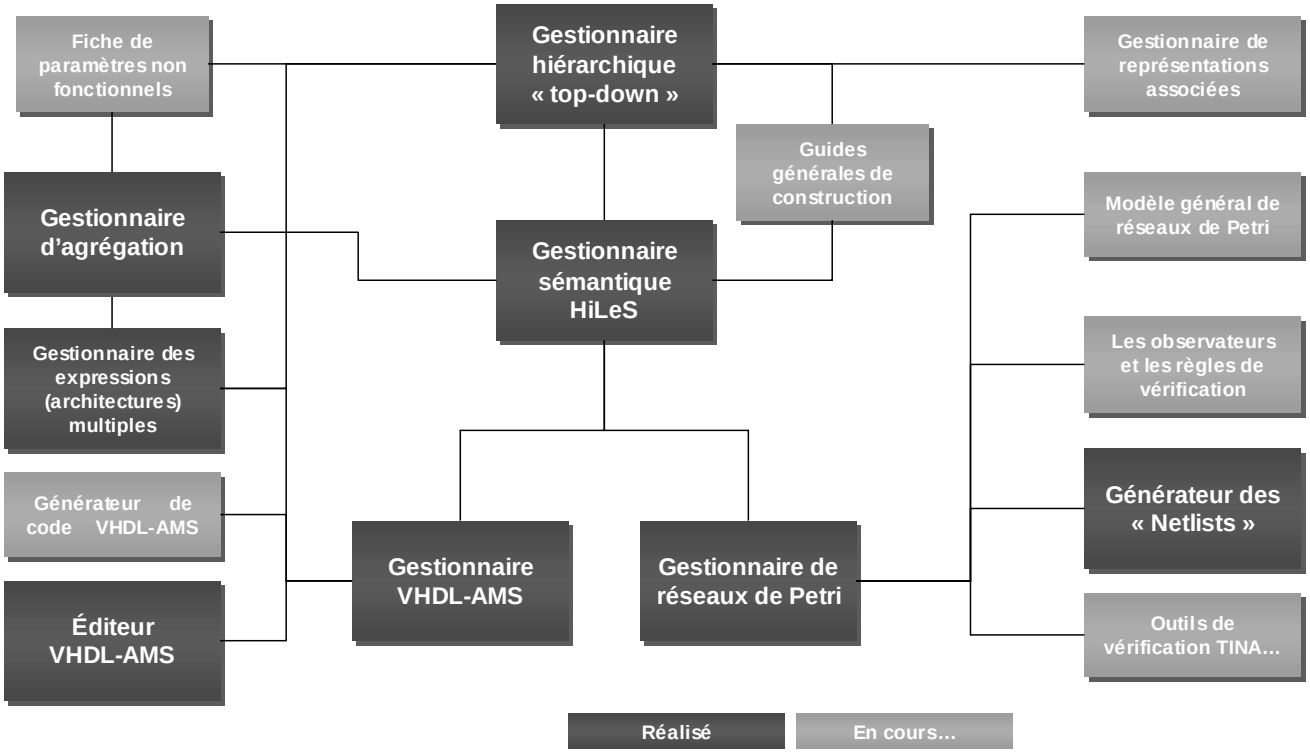


Figure 2-14. Diagramme des outils généraux de HiLeS Designer 0.

Avant de développer en détails l'implémentation de chaque module, nous présentons ci-dessous une brève synthèse des caractéristiques techniques de l'outil :

- **Fiche technique de l'outil HiLeS Designer 0 :** nous avons développé l'outil HiLeS Designer avec visual studio 6.0, plus particulièrement en utilisant Visual Basic 6.0. Voici les principales caractéristiques techniques et les besoins minimums d'installation :
  - système d'exploitation : Windows98, Windows2000 et Windows XP.
  - mémoire RAM recommandée : 64Mo
  - résolution de vidéo requise : minimum 1024 x 768 @256 couleurs
  - la quantité d'objets comprises dans un projet est uniquement limité par les performances du matériel informatique de l'utilisateur. Par exemple un projet comprenant 8300 objets HiLeS occupe 3.3Mo et demande 40 secs pour son ouverture initiale.
  - en l'état, le code comporte plus de 12000 lignes,
  - le paquet d'installation est téléchargeable via Internet.

2.9.1 Les fonctionnalités de HiLeS Designer 0

Les principales fonctionnalités de la version actuelle du logiciel sont les suivantes:

2.9.1.1 L'outil de gestion hiérarchique « Top-Down »

L'utilisateur peut définir un nombre illimité de niveaux de description. La navigation à l'intérieur et en dehors des blocs structuraux est facile. Tous les signaux définis dans les niveaux supérieurs sont automatiquement répertoriés en aval.

- **L'explorateur de projets:**

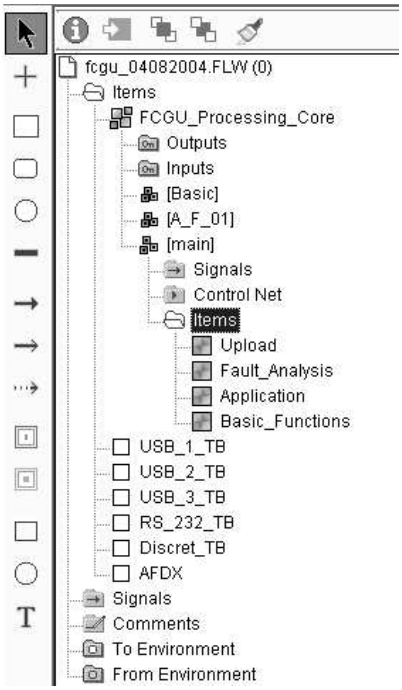


Figure 2-15 Aperçu de l'explorateur de projets HiLeS Designer.

HiLeS Designer comporte un explorateur de projets qui illustre clairement les dépendances hiérarchiques des blocs modèles, voir Figure 2-15. L'explorateur HiLeS Designer est semblable aux « browsers » habituellement employés par les logiciels commerciaux tels que « MS.Visual Studio 6 » ou Microsoft Explorer. L'explorateur offre aussi la possibilité de retrouver facilement les objets de la liste dans les feuilles du projet grâce a la fonction « *Locate Item* ». Lorsque l'utilisateur rajoute des éléments à la description initiale, l'explorateur de projets est mis au jour automatiquement. **Les icônes associées aux objets facilitent la tâche d'identification.**

2.9.1.2 L'outil d'édition et de gestion des blocs.

Hiles-Designer comporte les fonctionnalités basiques pour créer, connecter et hiérarchiser les blocs définis dans le formalisme HiLeS. Les blocs peuvent contenir des informations complémentaires à la description des fonctions représentées de manière comportementale ou structurelle. Les informations complémentaires qui doivent enrichir la description des blocs sont :

Les types généraux:

- Logiciel
- Matériel
  - le domaine : cette propriété présente l'inconvénient d'être directement associée à l'outil de simulation VHDL-AMS. A l'heure actuelle, les fabricants d'outils ne proposent des nomes de natures ni de bibliothèques standardisées. Cet inconvénient devrait être corrigé grâce au standard IEEE 1076,1.1-2004 qui définit les natures standard VHDL-AMS
    - Electrique
    - Mécanique
    - Fluidique
    - Général ou Non défini

Les fonctions générales : il s'agit d'actions courantes implémentées dans les systèmes tels que :

- Capter,

- Mesurer,
- Communiquer (recevoir/émettre),
- Traiter,
- Alimenter,
- Mémoriser,
- Actionner,
- etc.

Les paramètres de l'architecture du bloc :

Cet aspect est très important car il permet de modifier ou de définir certains paramètres depuis le niveau immédiatement supérieur, de limiter le domaine des variables concernées par la description ainsi que de définir le « generic map » lors de la transformation finale en VHDL-AMS. Cette propriété représente aussi un avantage vis-à-vis de la réutilisation de blocs.

Les paramètres non-architecturaux et non-fonctionnels :

Un certain nombre de paramètres ne peuvent être modélises par le biais des ressources proposés par le formalisme HiLeS. Ni les blocs, ni les réseaux de Petri, ni le langage VHDL-AMS ne permettent de représenter des éléments tels que :

- Les fournisseurs,
- Le coût,
- Temps et délais de développement ou fabrication,
- La technologie,
- Les contraintes de design tels que la taille, le poids, etc.…

L'affectation des attributs permettra d'établir des agrégations et des classifications de fonctions et sous systèmes selon des critères divers, permettant des analyses de temporisation globale et d'affectation du logiciel et du matériel.

Blocs structurels :

a. Bloc structurel vide b. Bloc structurel avec contenu

Figure 2-16. Illustration des blocs structuraux sous HiLeS Designer 0

Lorsque l'utilisateur décide de placer un bloc structurel, il pourra en fixer sa position et sa taille en utilisant la souris de l'ordinateur. Une fois le bloc placé, une fenêtre de dialogue va être affichée. A partir de cette fenêtre l'utilisateur aura la possibilité de choisir le nom du bloc en cliquant sur une liste d'actions prédéfinies ou bien de donner au bloc un nom particulier. L'utilisateur pourra également ajouter ces nouveaux noms à la liste d'actions associées aux blocs. Les noms ajoutés par l'utilisateur seront disponibles dès le prochain placement d'un bloc structurel. En cas de besoin, l'utilisateur peut définir une nouvelle couche à partir de ce type de bloc. L'outil permet de « plonger » dans le bloc afin d'implémenter facilement les hiérarchies, ce qui peut être défini comme une fonction « go inside ». En fait, c'est le nom de l'option permettant de réaliser cette action dans le logiciel. L'objet « bloc

structurel » doit comporter un champ destiné à garder l'information temporaire minimale de son temps d'exécution.

Lorsqu'un bloc structurel contient un ou plusieurs niveaux hiérarchiques, l'outil ajoute automatiquement une étiquette à la représentation graphique. Cela doit permettre de reconnaître cette particularité d'un simple « coup d'œil » telle que l'illustre la Figure 2-16-b. Le carré coloré dans le coin supérieur droit indique que le bloc « Structure » possède au moins un bloc à l'intérieur.

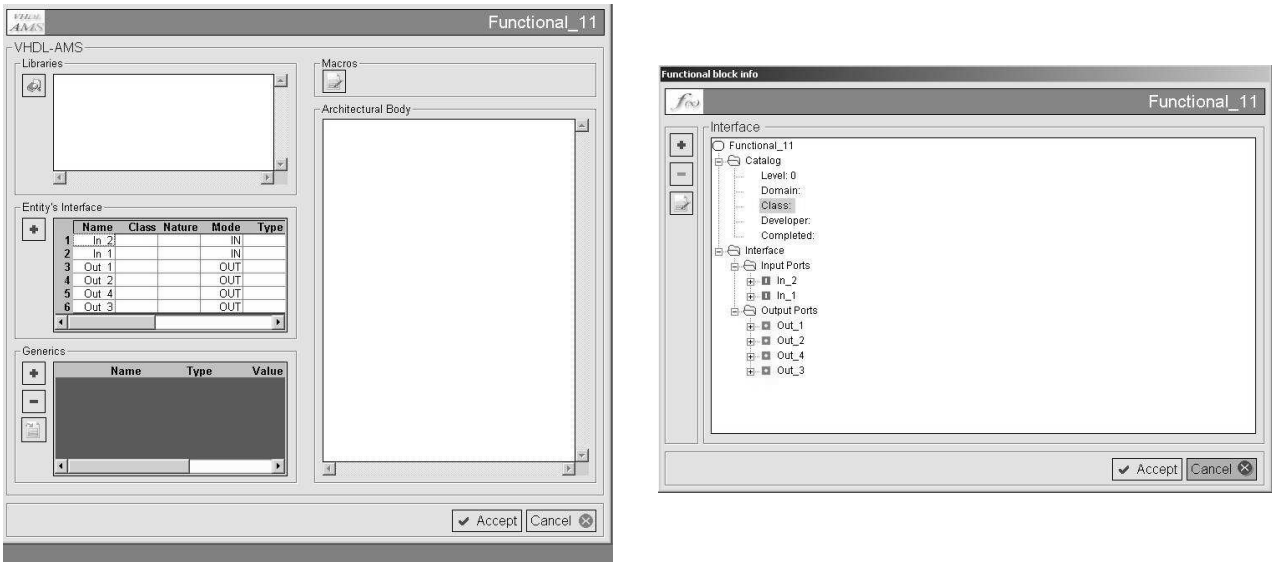
Blocs fonctionnels :

a. Bloc fonctionnel vide b. Bloc fonctionnel avec contenu.

Figure 2-17 Illustration type d'un bloc fonctionnel sous HiLeS Designer 0

Pour placer un bloc fonctionnel, le concepteur fixe sa position et sa taille en utilisant la souris de l'ordinateur. Une fois le bloc placé, une fenêtre de dialogue est affichée. A partir de cette fenêtre l'utilisateur a la possibilité de taper le nom du bloc. Les entrées et les sorties du bloc sont répertoriées dans une liste dont l'utilisateur a l'accès depuis les options associées aux blocs fonctionnels. Les blocs contenant des fonctions sont marqués avec une icône rouge et bleue au coin supérieur droit du bloc tel que l'illustre la Figure 2-17b. Pour définir le contenu du bloc, l'outil fournit des options adaptées.

Définition des fonctions :



a. Interface VHDL-AMS b. Fenêtre de synthèse de la structure du bloc

Figure 2-18. Eléments des l'interfaces de description fonctionnelle

Avec click droit, une liste d'options est affichée sur l'écran. Dans la Figure 2-18 nous présentons deux des fenêtres dédiées à cette interface. En choisissant INFO, il est possible d'écrire des équations, des algorithmes ou bien directement du code VHDL-AMS. Pour cela, une fenêtre d'édition est mise en place avec les options classiques d'un éditeur de code. La liste des ports du bloc est

disponible en permanence pendant l'utilisation de l'éditeur des fonctions. Lorsqu'un bloc fonctionnel contient des équations, du code ou des algorithmes, l'outil ajoute automatiquement une étiquette à la représentation graphique. Cela doit permettre de reconnaître cette particularité visuellement. L'objet « bloc structurel » doit comporter un champ destiné à garder l'information temporaire minimale de son temps d'exécution.

Les ports

Pour la gestion des signaux, les blocs HiLeS utilisent des ports. Chaque fois que le concepteur trace une connexion entre blocs, des ports sont créés automatiquement. Les ports prennent en compte le sens des signaux.

Ports locaux d'entrée – sortie

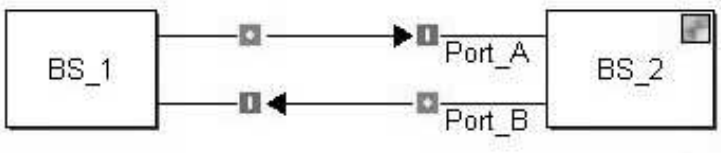


Figure 2-19. Ports d'entrée – sortie du même niveau

Les blocs de même niveau sont reliés par des canaux. A chaque fois que l'utilisateur trace un canal entre deux blocs, un port est automatiquement créé. Les ports de sortie sont identifiés par des carrés rouges pleins tandis que les blocs d'entrée le sont par des carrés bleus pleins, voir la Figure 2-19

Ports externes d'entrée – sortie : Les Services

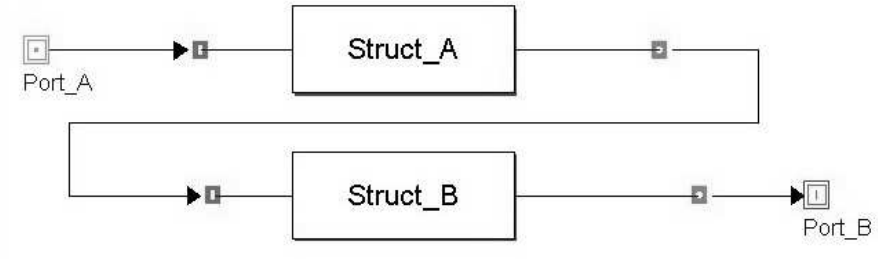


Figure 2-20. Illustration des ports de niveaux différents

Pour « traverser » les différents niveaux du projet, l'outil comporte des ports spéciaux. Ces ports transportent les signaux d'entrée et de sortie vers l'intérieur ou vers l'extérieur des blocs. Les noms donnés aux ports sont copiés automatiquement à l'intérieur du bloc concerné. Les ports sont représentés par des carrés vides rouges et bleus, voir Port_A et Port_B dans la Figure 2-20. Pour mieux comprendre, regardons le schéma de la Figure 2-19, le bloc BS_2 contient des sous blocs, voir l'icône dans le coin supérieur droit, ses ports ont été définis comme Port_A et Port_B. La Figure 2-20 montre l'intérieur du bloc BS_2. Les ports d'entrée et de sortie ont été copiés avec le même nom ce qui permet une interconnexion aux autres blocs.

Continuité des signaux : les ports assurent la continuité des signaux d'un niveau à l'autre. Le type des signaux est vérifié automatiquement par l'outil, de cette façon, l'utilisateur est assuré de

garder la cohérence de ces connexions. Pour le cas particulier des arcs des réseaux de Petri, l'outil vérifie le formalisme du modèle.

La connectivité dans HiLeS

Voici une synthèse des règles de connexion des éléments du formalisme HiLeS. Ces règles de base ont été mises en application dans la syntaxe implémentée sur l'outil HiLeS Designer.

Tableau 2-1. Tableau des connexions entre éléments de HiLes Designer.

Destination / Origine	Place	Transition	Bloc. structurel	Bloc. fonctionnel	Service d'entrée	Service de sortie
Place		A			A	A
Transition	A		A	A		
Bloc structurel	A	A	CC, CE, A	CC, CE	CC, CE, A	
Bloc fonctionnel		CC	CC, CE	CC	CC	
Service d'entrée	A		CC, CE	I		
Service de sortie	A		CC, CE	CC		

A : arc du réseau de Petri.
CC : canal continu
CE : canal à événements discrets
En gris les connexions non autorisées

2.9.1.3 Le gestionnaire d'architectures à expressions multiples.

Cette fonctionnalité de HiLeS Designer permet de décrire la construction structurale de blocs par le biais d'une ou de plusieurs architectures. Ceci permet à l'utilisateur de définir différentes solutions pour la même fonction sans changer son interface. L'utilisateur peut choisir une des options comme architecture par défaut dans la mesure où l'on ne peut utiliser qu'une architecture à la fois pour produire des modèles. Ce principe est inhérent à la construction classique entités-architectures de VHDL et a été déjà suggéré par le "paradigme des modèles re-configurables" [DPK00] pour la conception de systèmes.

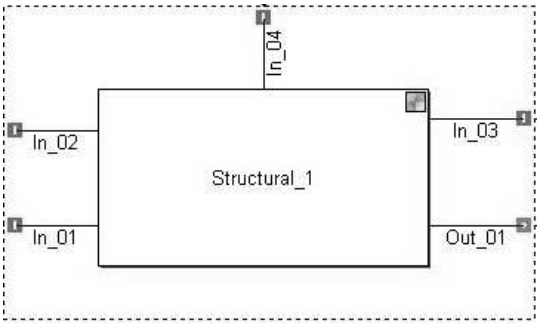


Figure 2-21. Définition de la structure Structural_1

Afin d'illustrer ce principe, prenons par exemple le bloc structurel (Structural_1) de la Figure 2-21. Il comporte quatre entrées : In_01, In_02, In_03 et In_04 ; et une sortie, Out_01. L'icône affichée sur le coin supérieur droit indique que le bloc comporte également une architecture intérieure. Les entrées / sorties du bloc définissent une interface que nous allons conserver tout au long de cet exercice qui va consister à décrire une même entité de plusieurs façons.

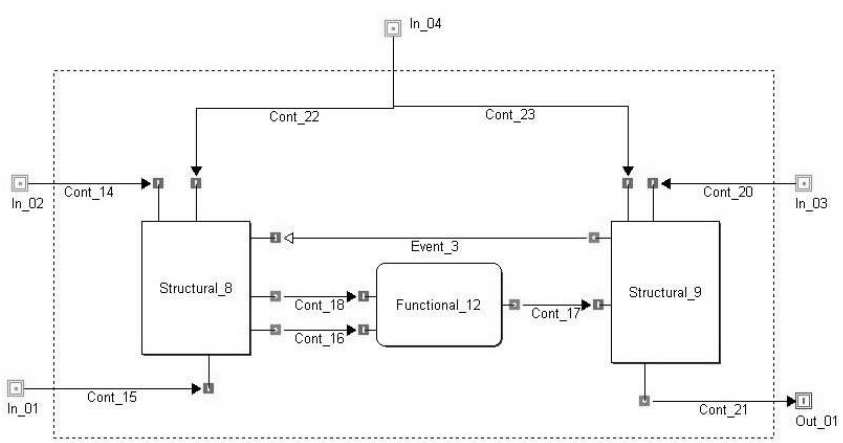


Figure 2-22. Architecture 1 de Structural_1

Nous proposons ici trois solutions possibles pour le bloc Structural_1 (Figure 2-23, Figure 2-22 et Figure 2-24). *HiLes Designer* permet au concepteur de créer les trois sous-architectures et de les garder à l'intérieur de « Structural_1 », même si l'on n'utilise que l'une d'entre elles.

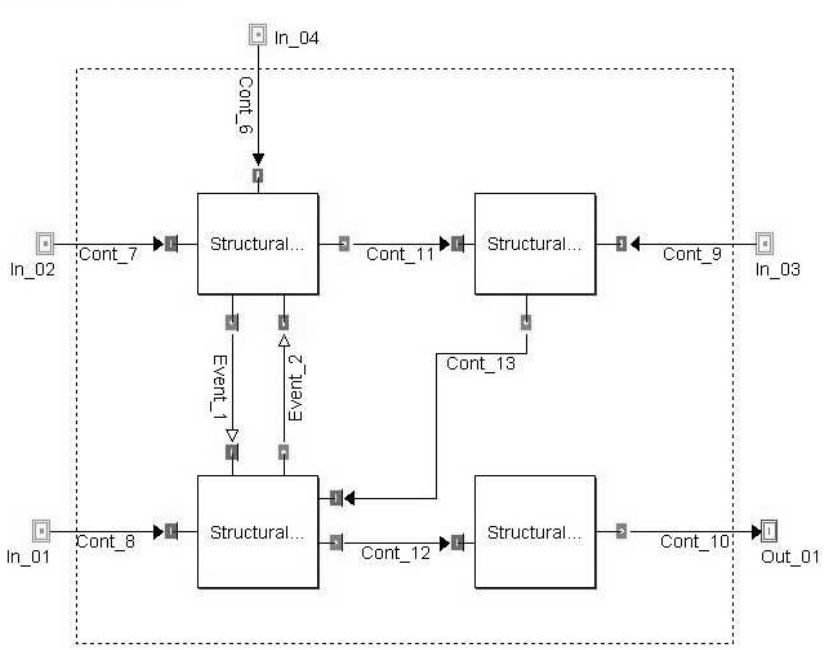


Figure 2-23. Option d'architecture 2 de Structural_1

Toutes les architectures sont connectées avec leur niveau supérieur par le biais d'une interface commune définie lors de la mise en place du bloc « Structural_1 ». Le concepteur pourra alors déclarer une architecture par défaut qui sera utilisée lors de l'analyse du réseau de contrôle et pour la génération du modèle VHDL-AMS. De cette façon, plusieurs options architecturales peuvent être considérés et surtout documentées.

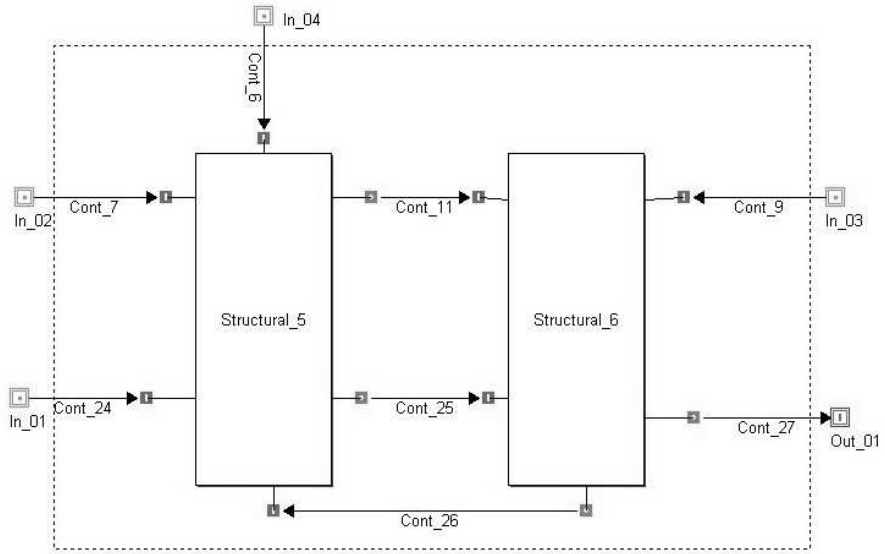


Figure 2-24. Option d'architecture 3 de Structural_1

Les applications envisagées pour cette fonctionnalité sont :

- **L'exploration architecturale :** les concepteurs peuvent proposer et combiner des options multiples pour résoudre des problèmes gardant l'environnement général (interface) de la fonction.
- **Support à la décision** [Ham03] : en appliquant une approche « Top-Down » il est souvent nécessaire de prendre des décisions au sujet de la mise en place des fonctions. Ceci permettra de documenter les options lors de l'implémentation d'une solution particulière, ainsi, les différentes alternatives considérées pourrant apparaître dans un projet HiLeS Designer même si uniquement une parmi elles est finalement retenue.
- **Configurations** : l'utilisateur peut définir une expression sous plusieurs configurations,
- **Une étape future d'exploration architecturale automatisée :** le développement ultérieur du logiciel peut inclure quelques techniques "intelligentes" pour profiter des expressions multiples. Cette exploration peut être faite sous des critères imposés par le concepteur. Nous envisageons des applications à base d'algorithmes génétiques comme des extensions possibles au projet GESOS [BRE04] du LESIA.

2.9.1.4 L'encapsulation : un outil d'agrégation et de regroupement

Les représentations fonctionnelles HiLeS nécessitent d'évoluer vers des structures architecturales pour pouvoir engager des procédures de partition, de sélection de scénarios et d'optimisation. Ces représentations architecturales seront issues de l'agrégation ou du regroupement des éléments du formalisme HiLeS. Pour ce, faire nous avons implémenté dans l'outil HiLeS Designer la possibilité d'encapsuler deux ou plusieurs éléments en un seul bloc structurel.

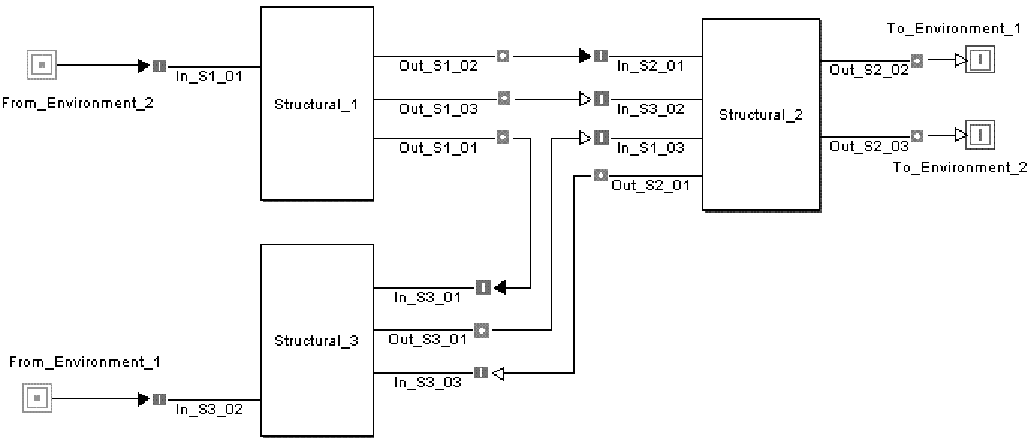


Figure 2-25. Ensemble de blocs avant l'encapsulation

Prenons l'exemple de la Figure 2-25, ici nous observons 3 blocs connectés entre eux et avec l'environnement. Nous voulons en faire un bloc contenant « Structural_1 », et « Structural_2 ». Avec HiLeS Designer nous pouvons sélectionner les blocs et créer un bloc nouveau. La Figure 2-26 montre le bloc résultant de cette agrégation. Les ports du bloc « Group_Struct » correspondent aux ports des blocs originaux en connexion avec l'extérieur.

L'encapsulation prend en compte les caractéristiques générales de chaque bloc, ainsi, les paramètres génériques définis pour chacun, feront partie de la description du nouveau bloc. Ce principe s'applique pour l'ensemble des constituants de la sémantique HiLeS. Ainsi, nous pouvons imaginer, par exemple, le regroupement des places et transitions d'un réseau de contrôle d'un projet sous un seul bloc centralisant l'intelligence du système ou bien la construction d'ensembles de blocs avec des caractéristiques communes.

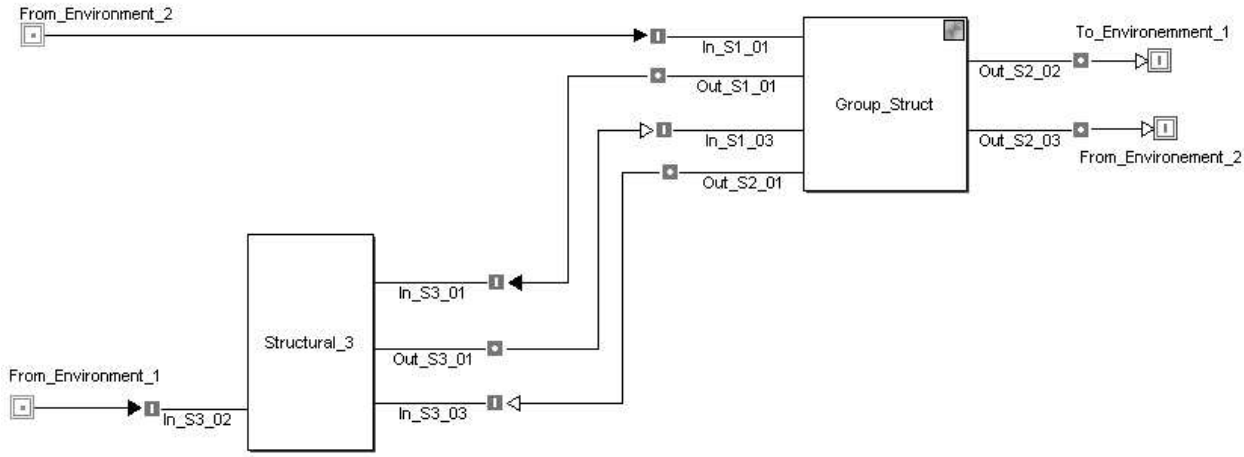


Figure 2-26. Génération d'un bloc après l'encapsulation.

A l'heure actuelle l'outil fonctionne manuellement, c'est-à-dire, l'utilisateur de HiLeS Designer doit sélectionner les éléments qu'il souhaite agréger puis exécuter la commande correspondant. La Figure 2-27 montre la fenêtre que nous avons mis en place pour faciliter le processus d'encapsulation. Lors que l'utilisateur décide de regrouper des blocs l'interface présentée (titre en bleu) permet de

construire la liste d'éléments HiLeS qui constitueront le nouveau bloc, ainsi que de définir son nom et celui de l'architecture qu'ils vont intégrer.

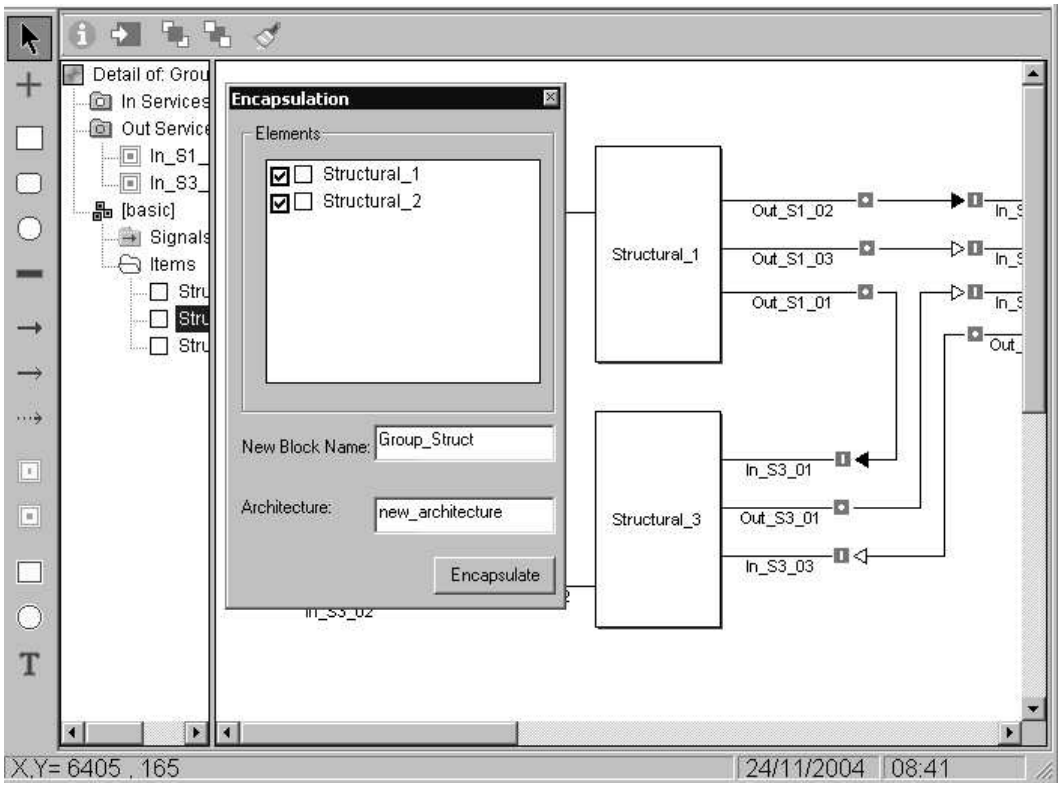


Figure 2-27. Fenêtre de sélection et encapsulation de blocs

Nonobstant, cette première version rudimentaire, nous pouvons envisager des modes de fonctionnement dans lesquels l'outil pourra regrouper des blocs selon des critères divers et même le faire de façon automatique en suivant des règles d'encapsulation ou des attributs préalablement définis sur chaque élément.

Grâce à l'encapsulation, nous pouvons envisager des représentations différentes d'un même projet, non seulement des architectures réalisables mais aussi des simplifications du système permettant de le considérer sous des points de vue différents (c.f. §2.4.7) : fournisseurs, nature ou métier associés à leur fonctionnalité.

2.9.1.5 Gestionnaire de réseaux de Petri

HiLeS Designer 0 inclut les outils destinés à éditer, manipuler et analyser les réseaux de Petri :

- **l'éditeur :** l'interface graphique prédéfinit des objets pour la construction des réseaux de Petri. Il vérifie les règles de construction de base en ligne,
- **le générateur de « Netlist » :** pour la validation formelle, il exporte le réseau de commande sous la forme de netlists compatibles avec le format de texte de TINA.
- **l'interface TINA :** des analyses de base des "bonnes propriétés" sont lancées automatiquement depuis HiLeS Designer.

Ci dessous nous présentons la liste et la spécification des éléments fournis par Hiles Designer 0 pour la représentation graphique du modèle de commande.

Les places :

Figure 2-28. Symbol d'une place.

Les places sont représentées par des cercles de taille fixe. L'utilisateur peut les positionner à sa guise en utilisant la souris. Ensuite une fenêtre de dialogue permet d'écrire le nom de la place. La Figure 2-28 illustre une représentation typique d'une place. Le nom PN à côte droite du cercle peut être édité en faisant double click sur la place. Dans l'outil, N est un indice incrémenté automatiquement.

Figure 2-29 Place avec marquage

Si l'utilisateur décide de marquer un jeton dans la place, l'outil fournit l'option pour le faire : En faisant click droit sur la place. Le modèle de commande de HiLeS est donc, les réseaux de Petri non colorés : un seul jeton à la fois est accepté dans chaque place. Le point noir au centre de la place PN de la Figure 2-29 représente la présence d'un jeton dans la place.

Les transitions :

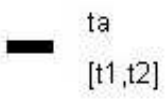


Figure 2-30. Symbole d'une transition avec son intervalle de tir

Les transitions sont aussi des objets de la boite à outils. Au même titre que les autres objets HiLeS, elles peuvent être placées en les déplaçant avec la souris. Les transitions comportent deux informations, le nom de la transition et l'intervalle de tir. Si l'utilisateur ne spécifie pas les limites de l'intervalle, celle-ci est déclarée automatiquement entre 0 et l'infini : [0,W[. L'outil dispose d'une fenêtre spéciale pour rentrer les valeurs de l'intervalle, l'accès à cette option s'effectue avec le bouton droit de la souris en sélectionnant la transition.

Les arcs des réseaux de Petri :

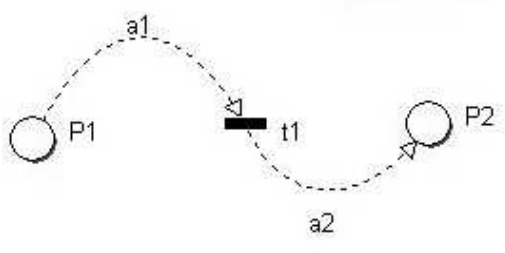


Figure 2-31 Les arcs des réseaux de Petri.

Les arcs sont représentés par des flèches pointillées. Les arcs peuvent être identifies par des noms associés. Dans HiLeS, tous les arcs ont un poids de un. L'outil ne permet pas la connexion directe de deux transitions ou de deux places.

2.9.1.6 Compatibilité VHDL-AMS

Tel que nous avons mentionné (§3.2) au début de ce chapitre lors la présentation des généralités de notre démarche, nous avons fait le choix d'utiliser le langage de description matériel VHDL-AMS pour décrire les différents modèles créés sous HiLeS.

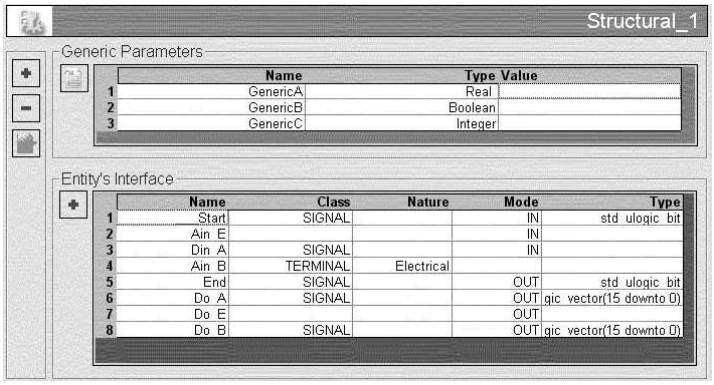


Figure 2-32. Interface de configuration des blocs structuraux implémentés dans HiLeS Designer.

Afin de générer un code VHDL-AMS valide, il est nécessaire de préciser un certain nombre de caractéristiques concernant les objets VHDL-AMS utilisés. Ainsi HiLeS Designer propose une interface permettant d'obtenir du concepteur les informations nécessaires au code VHDL-AMS en l'orientant au maximum (choix des types, des natures). L'interface graphique de l'outil HiLeS Designer permet d'éviter la phase d'écriture en VHDL-AMS de la description structurelle d'un bloc. La Figure 2-32 montre la fenêtre d'interface pour la configuration des blocs structuraux sous HiLeS Designer.

- **Editeur VHDL-AMS :** HiLeS Designer comporte un éditeur local de VHDL-AMS permettant de compléter la description fonctionnelle des blocs. Cet éditeur identifie et souligne les mots-clés du langage. Le code qui peut être crée en utilisant cet éditeur correspond à l'architecture du bloc. Le code correspondant à la construction de l'interface n'a pas besoin d'être écrit par le concepteur car il est directement extrait de la structure du système par l'outil. La Figure 2-33 illustre la fenêtre de dialogue de l'éditeur interne VHDL-AMS de HiLeS Designer.

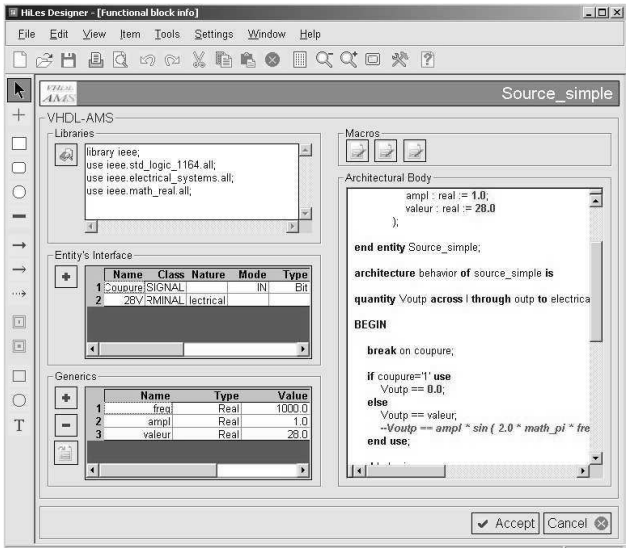


Figure 2-33 Fenêtre de l'éditeur interne VHDL-AMS.

- **Générateur semi-automatique de code :** HiLeS Designer produit une représentation VHDL-AMS équivalente du système, y compris les réseaux de Petri, les blocs structuraux et le contenu des blocs fonctionnels. La génération de code n'est pas 100% automatique parce que la description interne des fonctions dépend du code écrit par l'utilisateur.

Nous allons détailler dans notre chapitre 3 la manière dont nous réalisons notre modèle final écrit en VHDL-AMS. Nous appellerons par la suite cette caractéristique de compatibilité comme la « passerelle VHDL-AMS ».

2.9.1.7 Complementary Data Files (CDF)

Afin de compléter les informations sur des fonctions, HiLeS Designer 0 fournit les CDF. Ces fiches attachées à chaque bloc contiennent les données qui ne peuvent pas être exprimées dans le modèle VHDL-AMS, nous avons appelé ces données « les paramètres non fonctionnels ». Une des applications potentielles des CDF est le suivi historique du projet. Ces fiches peuvent contenir tous les critères et justifications des décisions, des choix de mise en place et même des choix technologiques. Les informations consignées dans ces fiches permettent de construire des scénarios possibles de réalisation du système avec des critères de sélection établis par l'analyse des paramètres non-fonctionnels. Cette fonctionnalité permet aussi de remonter certains informations jusqu'au niveau de la conduite de projet. Les CDF peuvent contenir des données concernant les délais de fabrication ou de livraison, les coûts des composants ou bien les renseignements des fournisseurs.

2.9.1.8 Structure des fichiers d'un projet HiLeS Designer 0

Afin d'organiser les fichiers créés lors de l'utilisation de HiLeS Designer, nous avons défini une structure de fichiers qui constituent le projet. Quand un projet est enregistré, l'outil crée automatiquement un dossier de base avec le fichier *.flw qui comporte le modèle de base HiLeS lui-même, c'est à dire l'arbre construit sur la base des blocs et du réseau de contrôle.

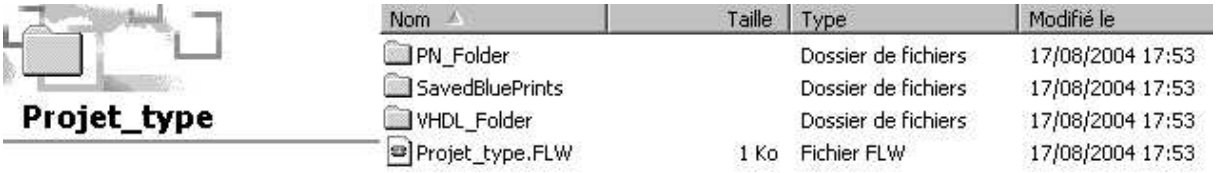


Figure 2-34. Structure des fichiers d'un projet HiLeS Designer 0

La Figure 2-34 illustre la structure d'un projet HiLeS Designer. Le dossier **Projet_type** contient le fichier **Projet_type.FLW** et les dossiers de la structure. Le dossier ***PN_Folder*** contient les netlists des réseaux de Petri et les fichiers de résultats de la validation réalisée par TINA. **SavedBluePrints** est le dossier par défaut des images et graphiques associés au projet. Finalement le dossier VHDL_Folder est destiné à contenir les fichiers équivalents *.vhd du projet.

2.10 Conclusion

Nous avons présenté tout au long de ce deuxième chapitre les éléments constitutifs du formalisme HiLeS, ainsi q'un résumé des principales caractéristiques de notre implémentation logicielle : l'outil HiLeS Designer 0.

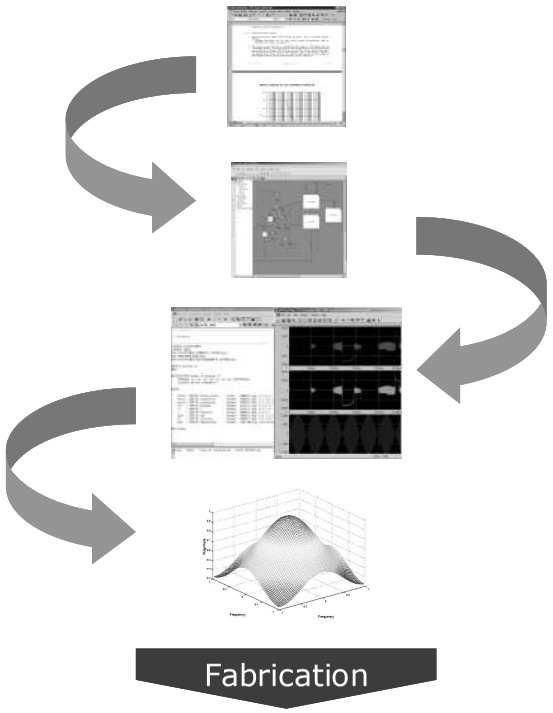
Le formalisme HiLeS propose de réaliser des modélisations de systèmes complexes pluridisciplinaires sous une représentation à base de blocs fonctionnels et structuraux, de canaux continus et discrets ainsi que des réseaux de Petri. Ces structures peuvent être hiérarchisées et déployées en plusieurs niveaux d'abstraction. Leurs activités internes sont coordonnées par le biais des réseaux de Petri qui interagissent avec les blocs. La modélisation est complétée par la génération d'un modèle complet équivalent écrit en VHDL-AMS. Ce langage standardisé IEEE assure la portabilité et la pérennité des modèles.

La mise en œuvre du formalisme sous la forme d'un outil logiciel permet au concepteur de l'appliquer dans un environnement informatique classique et convivial qui propose des fonctionnalités qui enrichissent le formalisme et facilitent son utilisation, à savoir : la possibilité d'opérer par décomposition-agrégation de blocs pour arriver à plusieurs expressions d'une même architecture. L'outil est par ailleurs conçu pour permettre de réaliser l'interface avec un outil de vérification de propriétés formelles des réseaux de Petri, un éditeur interne VHDL-AMS capable de reconnaître les mots réservés du langage, des fiches d'informations complémentaires ou CDF, ainsi que les outils de base pour une génération semi-automatique de code VHDL-AMS.

Jusqu'ici nous avons introduit HiLeS sans présenter en détail ses mécanismes de vérification et simulation. En conséquence, dans le prochain chapitre, nous détaillerons la façon dont nous interconnectons HiLeS Designer 0 avec d'autres outils afin de constituer une base de plate forme de conception système de haut-niveau.

Nous proposons finalement, dans la Figure 2-35, une classification d'outils et de langages par rapport à la démarche générale de conception système. Cet tableau nous permet d'identifier la position de HiLeS Designer 0 parmi d'autres outils. Loin d'être exhaustifs, nous nous sommes limités aux plus connus et/ou plus utilisés.

La démarche de conception :



	Activités de conception	Activités de modélisation et de traitement de l'information	Les outils :	Les langages :
Spécifications	Les objectifs Les modes d'emploi Les conditions environnementales Les exigences	L'analyse des besoins du marché Les produits et procèdes antérieurs L'innovation à promouvoir (L'idée)	Plan de rédaction Ingénierie des exigences (requirements engineering) Analyse des risques **Guides d'écriture**	**Naturel** SA/RT SysML™
Conception Amont	Proposer une architecture, Valider sa conformité aux modes d'emploi	Bibliothèque de fonctions élémentaires Modélisation de l'état de l'art Modélisation temporelle des modes d'emploi	PTOLEMY **Hiles Desiger 0** ESTEREL Studio HYPERFORMIX Cofluent HDL Designer MLDesigner Rosetta TINA ROMEO…	SADT SA/RT SDL UML **SysML** XML (gestion de l'information) Esterel RdP …
Prototypage Virtuel	Choix des composants Vérifier l'adéquation de ses fonctionnalités Introduire les effets de proximité Simuler le système dans sa globalité Simuler le système dans son environnement	Modèles physiques Modèles des fournisseurs Bibliothèques (CAO) Directives de conception Representation physique (Rendering & haptic)	CATIA AUTOCAD ANSYS MATLAB®/Simulink® CADENCE (PSPICE/ORCAD) AdvanceMS, SystemVision SaberHDL DOLPHIN HAMSTER SIMPLORER SABER Pspice ...	FORTRAN VHDL **VHDL-AMS** SystemC-AMS Langage C/C++ MAST JAVA …
Vérification et Optimisation	Appliquer les procédures de validation L'optimisation • Robustesse • Tolérances • Coût • Sûreté de fonctionnement • Fiabilité • Performances	Modèles économiques Modèles de fatigue Modèles d'influence	iSIGHT MATLAB®/Simulink® Nombreux outils internes développés par des universitaires et laboratoires scientifiques	Esterel Les RdP …

Figure 2-35. Les outils et les langages dans une démarche générale de conception système.

CHAPITRE 3

3. UNE PLATE-FORME DE CONCEPTION AMONT

3.1 Introduction

Dans ce troisième chapitre, nous allons mettre en œuvre les recommandations telles que nous les avons présentées dans les chapitres précédents, sous la forme d'une plate-forme de conception système conçue autour de l'outil HiLeS Designer. Nous définirons tout d'abord ce que nous entendons par plate-forme pour nous pencher ensuite sur sa mise en œuvre et son utilisation. Bien sûr, l'interopérabilité des modèles et des outils est le cœur opérationnel. Stratégiquement, nous considérons que notre plate-forme doit fonctionner à deux niveaux :

- La conception amont centrée sur HiLeS.
- Le prototypage virtuel basé sur le langage VHDL-AMS.

Autour de ces deux niveaux, doivent s'organiser des outils qui entretiennent des passerelles entre eux permettant d'assurer l'interopérabilité entre les constituants et la réalisation de toutes les fonctions de la plate-forme. Notre ambition ici sera limitée aux passerelles suivantes :

- Vers les spécifications,
- Vers la vérification structurelle et fonctionnelle,
- Vers la simulation VHDL-AMS ou prototypage virtuel.

Nous détaillerons la façon dont nous avons conçu et réalisé ces passerelles qui permettent l'interconnexion de HiLeS Designer avec les autres éléments de la plate-forme. Comme nous l'avons déjà mentionné dans le chapitre 2, nos objectifs ne visent pas le développement des outils de simulation : nous avons surtout privilégié l'interfaçage de HiLeS Designer avec les outils existants pour constituer la première étape d'une plate-forme de conception système ouverte à d'autres initiatives.

3.2 Plates-formes et modes d'utilisation

Une plate-forme est, par définition, un support technique proposant un ensemble d'outils informatiques et/ou des moyens matériels partagés, permettant de concevoir et/ou de réaliser un objet ou de développer une solution à un problème spécifique. Ce concept est aujourd'hui présent dans plusieurs secteurs d'activité, car la complexité croissante des applications exige des méthodes unifiées et l'intégration de plusieurs outils pour aboutir à des solutions appropriées et performantes. Une plate-forme doit être implémentée de façon à assurer l'interopérabilité et l'accessibilité de ses éléments constituants et partenaires, en minimisant les difficultés de passages d'une étape à l'autre. Il s'agit de garantir une totale cohérence dans les évolutions et transformations subies par l'action, le processus ou le produit à l'intérieur de la plate-forme.

Le concept de plate-forme est très répandu dans les applications électroniques, microélectroniques et informatiques. Dans ce dernier cas, plusieurs outils existent dans le but d'exécuter ou de développer [Nak03] des applications logicielles. L'application la plus ancienne et la plus courante des plate-formes est la CAO électronique, notamment dans les étapes finales de conception et de réalisation des produits électroniques : aujourd'hui, nous trouvons des produits commerciaux qui proposent plusieurs modules pour aider le concepteur dans la simulation des circuits jusqu'au « lay-out ». C'est le cas de la plate-forme OrCad [CDS03b] de chez Cadence qui intègre Capture, PSpice, PCB Layout et Specctra routage automatique. La Figure 3-1 illustre l'organisation très séquentielle de cette plate-forme.

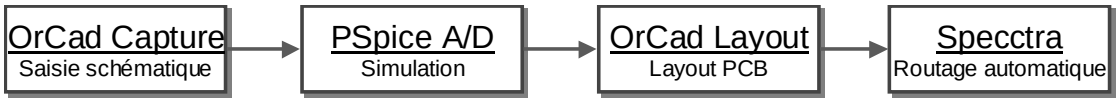


Figure 3-1 Schéma de l'organisation de la plate-forme OrCad Ultra [CDS03b].

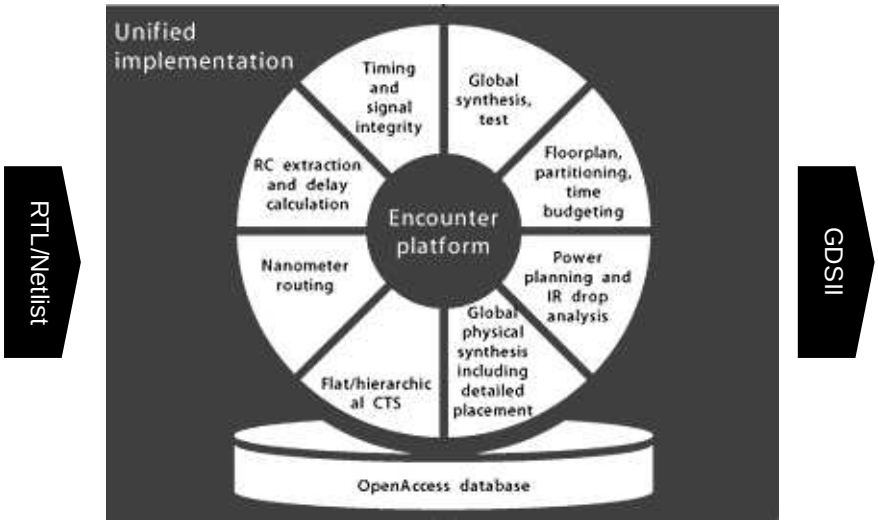


Figure 3-2 Diagramme des éléments de la plate-forme ENCOUNTER [CDS04a].

Au niveau de la microélectronique, nous trouvons des plate-formes telles que ENCOUNTER [CDS04a] de chez Cadence qui permettent de réaliser le cycle complet de synthèse des circuits intégrés. ENCOUNTER est dédiée à la conception des circuits intégrés numériques avec une

précision de montage nano métrique. Le principe d'opération consiste à prendre une description RTL (Register transfer level) du circuit, de la transformer et de l'optimiser pour aboutir à une représentation GDSII (Graphic Design Station II). La Figure 3-2 illustre l'organisation générale et les modules logiciels qui composent cette plate-forme.

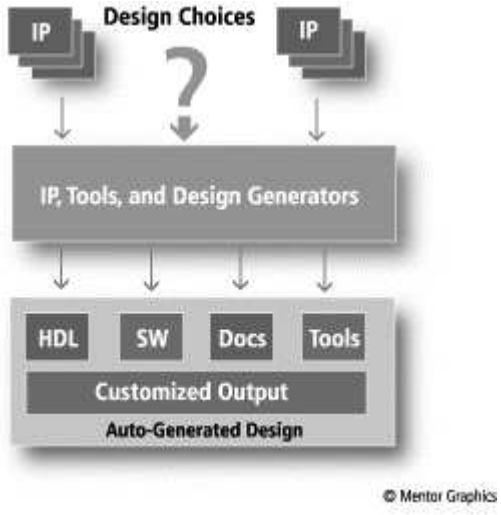


Figure 3-3. Schéma de la plate-forme « Platform Express » de Mentor Graphics

D'autres fabricants proposent des plate-formes crées avec l'objectif d'accélérer au maximum le processus de conception. Tel est le cas de Platform Express™ [MG04] de Mentor Graphics. Ce produit permet de concevoir des SoC à partir d'une base de données de modules IP construite en XML. Cette plate-forme propose des fonctionnalités permettant de tester plusieurs configurations et combinaisons d'IP sur un projet.

Il faut observer que, même si les outils sont nombreux et organisés pour couvrir l'ensemble des besoins des ingénieurs, de nombreuses insuffisances subsistent et font l'objet de recherches importantes :

- La question des spécifications des produits et de la réutilisation des acquis,
- La question de la synthèse automatique des solutions techniques, notamment des parties analogiques,
- La question de l'organisation des tests apportant les supports de validation, vérification, certification…

Revenons aux systèmes à base d'électronique. D'autres plate-formes s'adressent d'avantage à l'intégration de « Systems on chip » (SoC) et l'assemblage électronique en mettant en jeu des options technologiques hétérogènes et en nécessitant des arbitrages difficiles : « hardware/software ». Ces options et ces arbitrages posent aussi des questions difficiles encore sans réponses comme l'illustre la problématique qui nous a été posée au lancement de notre travail par les concepteurs d'AIRBUS, sur la partition matériel/logiciel : y a t'il une approche méthodologique qui permettrait d'analyser, au plus tôt, ces alternatives technologiques et ces arbitrages matériel/logiciel ? Cela a donné la motivation fondamentale de notre travail et le choix initial d'un objectif de plate-forme HiLeS (Figure 1-3).

A des niveaux moins généraux, il existe des plate-formes qui sont utilisées pour résoudre des problèmes spécifiques tels que le développement de systèmes embarqués [VG01], la mise en œuvre de réseaux de télécommunications [IUN98], parmi d'autres… Par exemple, les plates-formes « matérielles » permettent d'implémenter plusieurs solutions en utilisant une même base « hardware », c'est le cas des cartes électroniques programmables, permettant d'utiliser une base fixe de hardware pour des applications diverses [ZWI99]. Les plate-formes matérielles sont aussi utilisées pour trouver plusieurs solutions différentes à une même problématique [ZAB+99][CSC+04], en particulier comme des ressources reconfigurables.

Dans notre cas, le point de départ est bien l'électronique mais considérée dans un environnement de système hétérogène : nous voulons y proposer un ensemble d'outils permettant de réaliser la première étape du processus de conception, l'élaboration des modèles « amont » issus des spécifications et représentatifs du fonctionnement du système. Idéalement ces modèles peuvent être considérés comme des **spécifications exécutables qui évolueront vers des hypothèses d'architectures du système, en assurant une cohérence complète entre la modélisation et les spécifications**. Ces modèles de haut-niveau doivent permettre d'engager des procédures de validation, vérification et certification, dès les premières étapes de la conception.

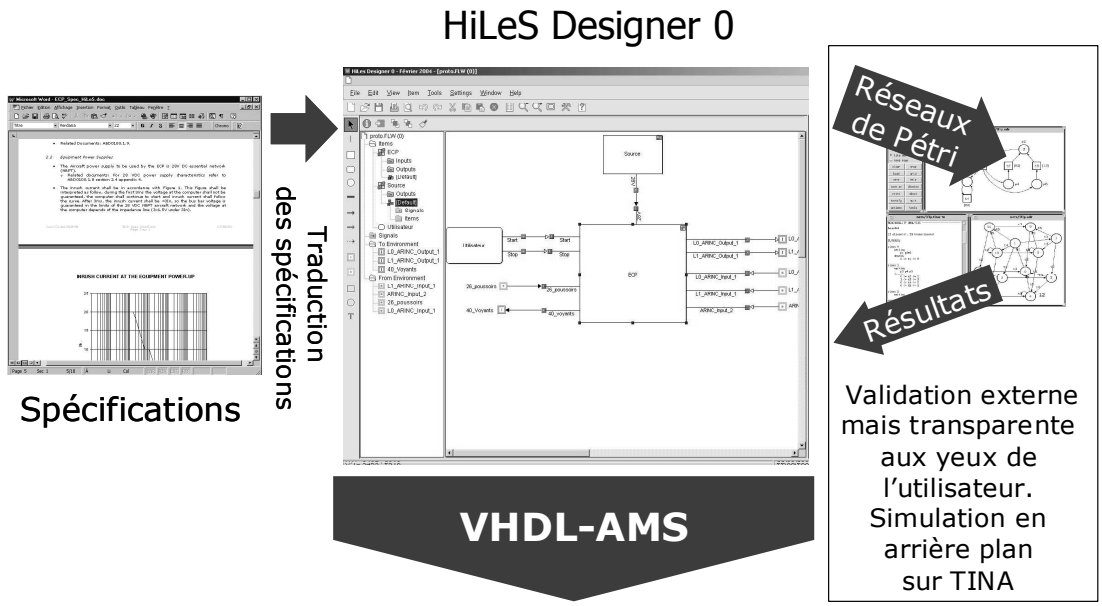


Figure 3-4. Diagramme général de notre concept de plate-forme.

Dans la Figure 3-4, nous représentons le concept général et le domaine d'intérêt de notre travail (déjà indiquée par des pointillés sur la Figure 1-3 du chapitre1). Nous pouvons le considérer comme une présentation synthétique de la plate-forme proposée. Nos options, à ce jour, pour le fonctionnement de cette plate-forme sont :

1. Le cahier des charges et spécifications du système seront interprétées par le concepteur et mis en forme avec l'interface d'utilisateur de notre outil : HiLeS Designer. Cette première représentation désignera les fonctions principales du système, leurs interconnexions et leurs dépendances hiérarchiques. Elle permettra de décrire son comportement général, par des réseaux de Petri associés.
2. Le modèle fonctionnel structurel, notamment la partie réseaux de Petri, sera validé en arrière plan en utilisant l'outil TINA par l'analyse des caractéristiques et propriétés des

réseaux de Petri. Pour ce faire, nous avons profité des modes opératoires de TINA et nous avons développé les interfaces nécessaires.

3. Nous proposons finalement de générer un modèle complet écrit en langage VHDL-AMS. Celui-ci est l'objectif à atteindre, cependant, à l'heure actuelle cette fonctionnalité n'est pas complètement opérationnelle. Ici, notre objectif n'est pas de créer des nouveaux outils de simulation, mais d'écrire nos modèles pour les simuler en utilisant des outils logiciels commerciaux.

Cette plate-forme de conception système à haut niveau devra à terme comporter les constituants suivants :

- Un outil ou mécanisme d'interprétation et de capture des spécifications,
- Un outil de représentation graphique permettant de mettre en forme les spécifications interprétées,
- Un outil de vérification des propriétés du modèle formel utilisé pour décrire le comportement du système,
- Un outil de génération automatique du code du système modélisé,
- Un outil de simulation permettant de vérifier, par exécution, le fonctionnement de la totalité du modèle (prototype virtuel).

En utilisant ces éléments de base, le concepteur pourra générer, à partir du cahier des charges, un premier modèle exécutable de haut niveau pour simuler et vérifier le comportement attendu de son système. Cette étape de conception intervient avant toute réalisation de prototypes physiques et avant même de définir précisément les choix technologiques, et les procèdes de fabrication… Tel que nous l'avons mentionné précédemment, ces modèles de haut niveau s'inscrivent dans une démarche générale de conception dans laquelle on doit faciliter trois actions essentielles visant la qualité du produit final :

- **La validation :** Pour estimer la viabilité d'un système en cours de conception,
- **la vérification :** Pour garantir que le système conçu répond correctement et complètement aux exigences spécifiées,
- **la certification :** Pour confirmer que le système conçu est conforme à des normes établies en interne ou en externe et démontrer que les choix conceptuels n'altèrent pas le comportement du système et qu'ils n'introduisent pas des états non attendus de fonctionnement.

Dans notre approche, nous n'abordons pas, pour les résoudre, tous les aspects considérés ci-dessus. Nous limitons notre ambition à un espace de travail (Figure 3-4) permettant de représenter les spécifications des systèmes graphiquement avec le formalisme HiLeS, de vérifier les propriétés du modèle formel ainsi établi sous la forme de réseaux de Petri, et de générer un modèle équivalent écrit en langage VHDL-AMS.

3.3 Le passage des spécifications à un modèle formel

Cette étape est sûrement la plus difficile à résoudre en terme d'outils d'aide à la conception. Nous l'avons abordé dans une publication collective [KHG+04] et nous l'approchons en collaboration avec nos collègues du groupe ISI au LAAS (Ingénierie Système et Intégration) dans le cadre du projet

interne MOCAS [Too04], que notre travail a contribué à initier. Nous présentons ce projet, plus en détails dans l'annexe A.

Pour le concepteur, le problème se pose comme le besoin d'extraire les éléments suivants des spécifications textuelles :

- Une définition claire, bien comprise, du système, de ses objectifs techniques et non techniques.
- Une identification des fonctions principales qui composent ce système en distinguant toutes celles qui font l'objet d'innovations de celles qui sont de « simples » réutilisations.
- L'anticipation de tous les modes d'interaction du système avec son environnement d'utilisation et les procédures de vérification qui pourront et devront être appliquées sur la représentation virtuelle et sur la réalisation matérielle finale.

Sans avoir résolu totalement le problème, grâce aux traitements de quelques exemples, [HSE+03] [HEP03] [MCEB04] [HEP+03] [PHM+04], voici les pistes que nous pensons être des éléments de progrès.

3.3.1 La représentation élémentaire HiLeS

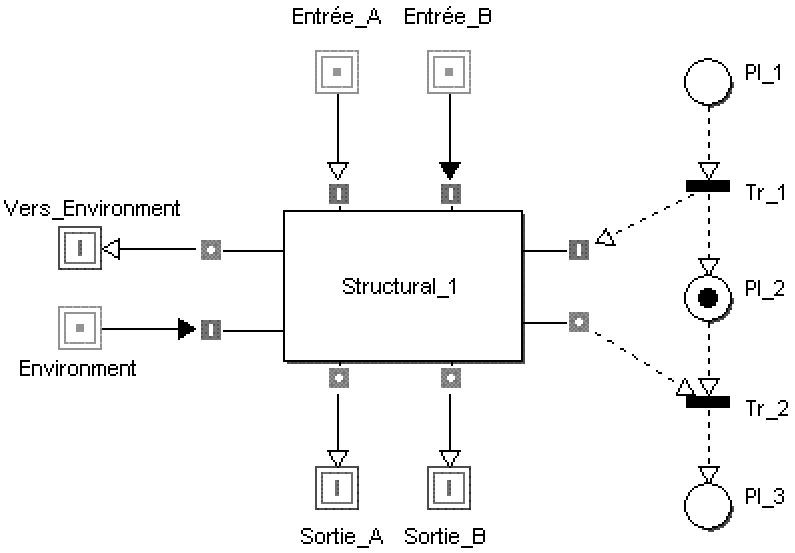


Figure 3-5. Exemple d'un module HiLeS.

Un bloc HiLes modélise une ou plusieurs fonctions par la durée globale d'exécution, identifie les entrées/sorties et dispose d'un élément de synchronisation : mise en route, arrêt, par exemple. Cette représentation est, au delà de la modélisation fonctionnelle ou comportementale, la représentation générique la plus riche possible. Sous cette forme ou sous des formes équivalentes, elle s'impose comme une étape importante (conception amont) dans la représentation d'un système. Elle est suffisante pour les premières vérifications fonctionnelles, structurelles ou séquentielles.

Notre objectif est de retrouver cette représentation à tous les niveaux pour pouvoir appliquer une démarche descendante systématique.

- **Niveau zéro :** fonction système, spécifications du fonctionnement global, identifications des acteurs extérieurs, entrées/sorties. Ce niveau zéro de représentation constitue l'environnement du système et dans la plupart des cas est défini ou imposé par le cahier des charges.

- **Les niveaux 1, 2, 3 :** Sont une décomposition hiérarchique statique de la fonction système, c'est-à-dire une déclinaison en sous fonctions de plus en plus détaillées jusqu'à l'identification de toutes les fonctions élémentaires souhaitables pour le système. La connaissance experte est essentielle, elle s'applique à la lecture du document des spécifications en s'appuyant éventuellement sur un dictionnaire de fonctions que nous préconisons d'établir, ainsi que sur une taxonomie des spécifications, incontournable au moment de définir les dépendances hiérarchiques et les priorités du système. **Le rôle de l'ingénieur est crucial car c'est lui qui va identifier les états fonctionnels du système**.

Cette démarche de simple description permet aussi d'identifier et de différencier les blocs et les sous-ensembles correspondants à la réutilisation d'acquis antérieurs.

La question qui se pose à ce stade de la description est double :

- Définir les interactions pour identifier les hiérarchies fonctionnelles et les interconnexions entre fonctions.
- Définir les séquences pour identifier et décrire tous les états du système.

Pour ce faire, nous préconisons d'inventorier les modes d'utilisation et les exigences sécuritaires et d'en définir les séquences. **Là encore un inventaire pré-établi (checklist) peut accélérer la procédure** : mise en route, arrêt, arrêts d'urgence, exécution des fonctionnalités programmées, tests embarqués, tests externes, etc. Chacune de ces procédures fera l'objet d'une représentation totale de la séquence correspondante et des éventuelles interactions entre les processus.

L'identification des interactions et des modes séquentiels des cas d'utilisation est très vite un travail d'une grande complexité que la disponibilité de méthodes et d'outillages spécialisés peut rendre plus rapide et plus performant. Il paraît plus stratégique d'opérer de manière : « Top-Down ». On écrit les diagrammes d'interaction entre les séquences dont on a établi l'inventaire, puis on décrit chaque séquence avec ses temporisations. Cependant, il y a probablement des cas où l 'approche « bottom-up » s'impose.

Au bilan de cette étape, on dispose d'une représentation complète, hiérarchique et temporisée. Exprimée en réseaux de Petri, cette architecture fonctionnelle temporisée peut être l'objet d'une vérification systématique. **Pour ce faire, une dernière étape est l'extraction du réseau de Petri complet ou partiel, représentant le système ou un sous-système, chaque fonction étant représentée par son délai d'exécution.**

3.3.2 La relation aux représentations UML

Comme nous l'avons déjà souligné, les méthodes et outils dédiés à la traduction des spécifications textuelles en un modèle formel permettant de réaliser la vérification et le choix architectural sont encore à développer. Les recommandations précédentes que nous avons déduites

du traitement de quelques exemples, doivent être approfondies. Toutefois, nous pensons pouvoir insister sur l'intérêt des composantes suivantes :

- La description très détaillée du niveau zéro,
- La description fonctionnelle hiérarchique en s'appuyant sur un inventaire de séquences type,
- La représentation formelle pour laquelle nous préconisons d'utiliser les réseaux de Petri,
- L'extraction du modèle complet « Réseau de Petri » pour pouvoir faire appel à une procédure de vérification.

Evidement, il faut que cette démarche descendante soit standardisée et nous attendons avec impatience la consolidation du langage SysML (§1.6.2) en cours de création à l'heure actuelle, issu de UML2. En l'état, nous pouvons faire les rapprochements suivants entre les recommandations que nous avons exprimées et la représentation UML.

Tableau 3-1 Rapprochement de nos recommandations avec les orientations SysML.™

Nos recommandations	Représentation équivalente SysML™ (UML2)
Niveau 0	Contexte
Niveaux 1, 2, 3…	Cas d'utilisation et représentations hiérarchiques
Représentation des séquences du système à base de réseaux de Petri.	Diagrammes d'activité et de séquence
Fiche des informations complémentaires	Diagramme de besoins et matrice de traçabilité.

L'intérêt de cette standardisation est qu'elle va apporter l'interopérabilité des représentations avec des opérateurs ergonomiques apportant au concepteur des vues de l'ensemble et des « zooms » fonctionnels. L'intérêt de notre représentation par réseaux de Petri reste entier pour réaliser les simulations « haut-niveau » et la vérification de la conformité avec les spécifications.

A long terme, il sera possible de proposer, sur la base de cette procédure standard, un véritable guide de rédaction de spécifications et donc d'envisager des automatismes de capture et d'interprétation, telles que celle que nous avons proposé dans le deuxième chapitre de ce mémoire (§2.6).

3.4 Passerelle HiLeS Designer – TINA

Nous partons ici de l'hypothèse que les spécifications sont interprétées par le concepteur et que grâce à son expérience, il peut définir les principales fonctions du système pour les représenter en utilisant l'interface graphique HiLeS Designer. Telle que nous l'avons présentée préalablement, cette représentation comporte un ensemble de blocs structuraux et fonctionnels coordonnés par un réseau de contrôle. C'est justement ce réseau de contrôle qui fait l'objet de cette passerelle vers une procédure et un outil de vérification.

Nous avons besoin tout d'abord d'extraire le réseau de Petri complet de la représentation HiLeS pour ensuite le « faire analyser » formellement. Pour ce faire, nous avons donc développé les outils et les méthodes adéquates. Initialement, nous parcourions les projets HiLeS pour identifier les arcs, les places et les transitions, ainsi que les niveaux hiérarchiques contenus dans la structure.

Une fois le réseau extrait, l'objectif est de pouvoir lancer le simulateur TINA en arrière plan à partir de HiLeS Designer, puis de récupérer les résultats et de les interpréter afin de fournir au concepteur les premiers éléments de vérification du projet. Pour ceci, nous accédons sur TINA directement, sans faire appel à son interface d'utilisation. Ce mode d'opération avait déjà été prévu par les concepteurs de TINA, ainsi que la possibilité d'utiliser comme entrée des représentations textuelles des réseaux. Nous présentons à la fin de ce document, une description synthétique complète de l'outil TINA réalisé par nos collègues du groupe OLC [Ber02].

Dans la Figure 3-6 ci dessous, nous illustrons de manière générale, la procédure que nous avons implémenté pour coupler les deux outils. Il s'agit d'une stratégie d'échange de fichiers.

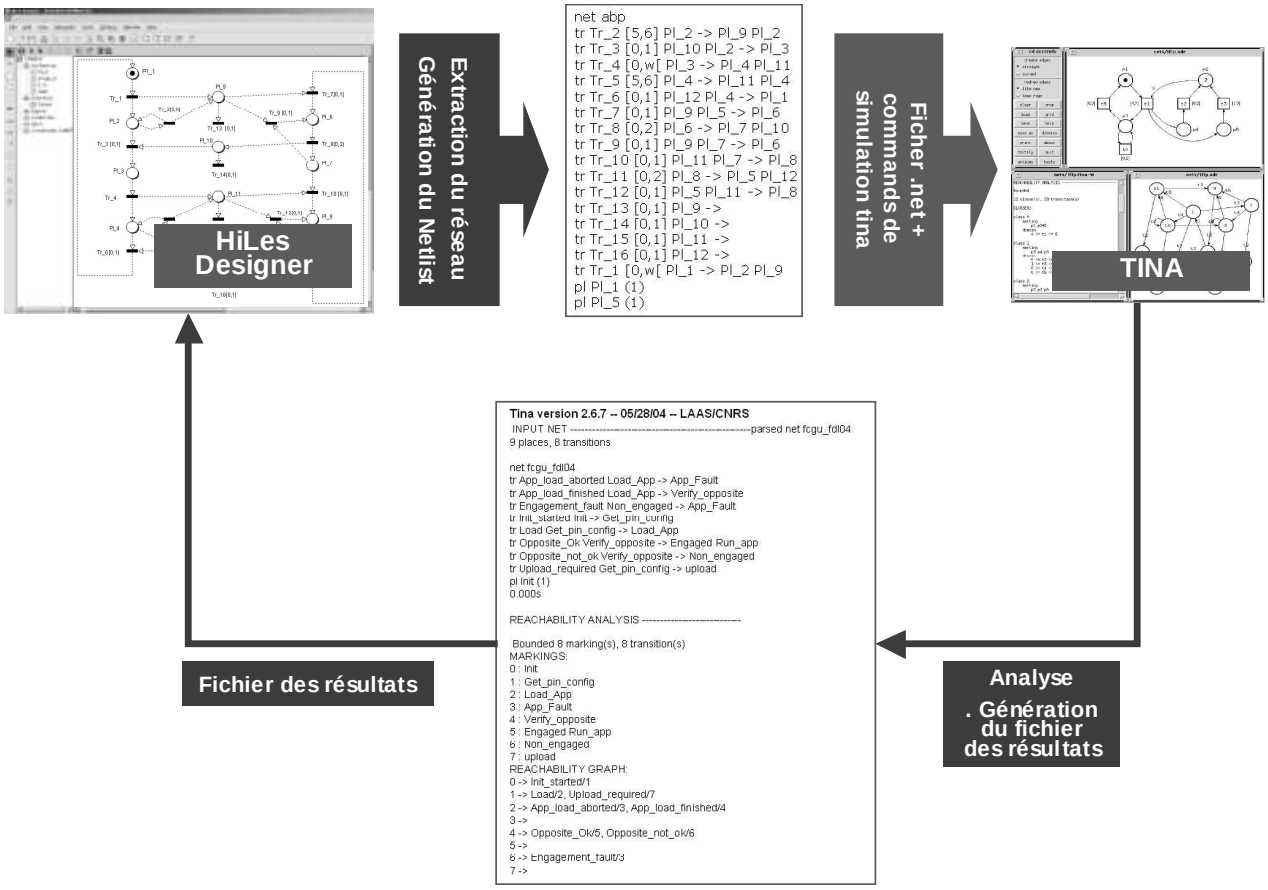


Figure 3-6. Diagramme général de l'échange de fichiers entre HiLeS Designer et TINA.

Cet échange de fichiers décrit par la Figure 3-6 est réalisé par les outils HiLeS Designer et TINA, dans le dossier crée automatiquement lors du premier enregistrement de chaque projet (c.f. §2.9.1.8), plus précisément dans le répertoire PN_Folder. C'est à cet endroit que seront sauvegardés tous les fichiers échangés entre TINA et HiLeS Designer.

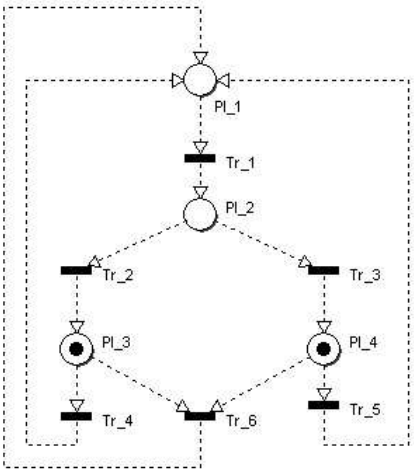
3.4.1 Extraction du réseau de contrôle

La première phase de la procédure est l'inspection du projet pour récupérer les réseaux de Petri. Il faut préciser techniquement que, à cause du caractère hiérarchique de la modélisation HiLeS, il est nécessaire de parcourir tous les niveaux de représentation du projet et d'identifier

automatiquement toutes les places et transitions parmi les autres éléments du projet. Cette étape d'exploration permet de construire un fichier texte équivalent à la topologie du réseau : **La netlist**.

3.4.2 Génération du fichier Netlist

La netlist est une description textuelle du réseau. Pour la générer, les transitions, les places et les arcs doivent être traduites dans une syntaxe très précise. Dans notre cas, cette syntaxe est définie par le format imposé en entrée de TINA. Nous gênerons le fichier texte et le sauvegardons dans le PN_FOLDER (c.f. §2.9.1.8) du projet.




```
net exempl_netlist
tr Tr_1 [0,w[ Pl_1 -> Pl_2
tr Tr_2 [0,w[ Pl_2 -> Pl_3
tr Tr_3 [0,w[ Pl_2 -> Pl_4
tr Tr_4 [0,w[ Pl_3 -> Pl_1
tr Tr_5 [0,w[ Pl_4 -> Pl_1
tr Tr_6 [0,w[ Pl_3 Pl_4 -> Pl_1
pl Pl_3 (1)
pl Pl_4 (1)
```

Figure 3-7. Exemple de génération du netlist TINA à partir de HiLeS Designer.

A titre d'exemple, sur la Figure 3-7, nous présentons un réseau de Petri dessiné avec HiLeS Designer et transformé automatiquement en fichier texte par le même outil. Considérons à droite de la figure la netlist correspondant au réseau de gauche : la première ligne « en tête » indique le nom du réseau. Les six lignes suivantes indiquent les transitions du réseau. Elles sont étiquetées avec le préfixe tr. Sur chacune des lignes de transitions, nous trouvons leurs intervalles de tir. Dans notre exemple, les valeurs des limites par défaut sont ([0,w[) zéro et infini. Après l'intervalle de tir, sont désignées les places d'entrée à la transition suivies d'une flèche et des places de sortie. Les deux lignes de la fin du fichier indiquent les places marquées avec des jetons. Entre parenthèses est indiqué le nombre de jetons par place. Tout réseau de Petri faisant partie d'un projet HiLeS Designer pourra être ainsi traduit en fichiers texte en utilisant le format que nous venons de détailler.

Pendant le processus d'extraction du réseau, nous avons prévu que l'outil réalise une vérification syntaxique sur les noms des places et des transitions. TINA accepte tous les caractères alphanumériques pour nommer les éléments des réseaux de Petri. Néanmoins, nous évitons d'utiliser des caractères non acceptés par la syntaxe du langage VHDL-AMS puisque nous devons garder la même nomenclature tout au long du projet, y compris pour la génération du modèle final. En cas de détection de caractères non autorisés ou de noms répétés dans le réseau, HiLeS Designer affiche un message d'erreur et donne à l'utilisateur l'option de visualiser les erreurs détectées.

3.4.3 Exécution de TINA à partir de HiLeS Designer

Une fois les réseaux extraits et exportés sous la forme d'un fichier type texte, nous pouvons réaliser la simulation sous TINA en utilisant ces données. L'outil TINA est indépendant de HiLeS Designer. Il doit donc être exécuté comme un programme externe. Nous avons pu incorporer facilement cette fonctionnalité, car TINA permet d'être exécuté depuis une ligne de commande, sans utiliser son interface graphique. C'est ainsi que nous pouvons spécifier le fichier à analyser, la destination du fichier des résultats et toutes les options d'analyse. Cette deuxième étape correspond à la flèche bleue dans le schéma de la Figure 3-6. Apres avoir lancé la vérification sur TINA, HiLeS Designer affiche un message d'attente jusqu'à la finalisation du processus de calcul.

En l'état, nous n'avons pas de contrôle sur l'exécution de TINA et nous n'avons pas la possibilité de l'arrêter par exemple si elle demande trop de temps. Ce cas est tout à fait possible car le temps de simulation des réseaux de Pétri peut être infini à cause du nombre d'états possibles à vérifier. Etant donnée cette caractéristique, nous avons mis en place des temps morts pendant lesquels HiLeS Designer attend la création du fichier résultats. Si au terme de ce délai, TINA n'a pas encore fini, HiLeS Designer demande à l'utilisateur s'il veut poursuivre son attente.

3.4.4 Les observateurs de propriétés.

Une des principales et plus grandes difficultés rencontrées dans notre travail a été l'analyse et l'exploitation des résultats de vérifications des bonnes propriétés des réseaux de Petri. En particulier, dans des contextes d'utilisateurs non-spécialistes en modélisation, selon ce type de formalisme, il convient de réaliser une interface facilitant l'interprétation des résultats. D'un autre côté, TINA est un outil très spécialisé qui contient beaucoup de fonctionnalités, que le concepteur – non-spécialiste - ne saura pas exploiter complètement.

Considérons l'analyse des « bonnes propriétés » des réseaux de Petri [Jim00] qui peuvent révéler dans quelques cas des caractéristiques associées au système qu'ils décrivent :

- « **Un réseau de Petri est dit vivant** pour un marquage donné si à partir d'un marquage accessible quelconque, toute transition peut être sensibilisée après une séquence de tir appropriée. Pour le système physique modélisé, la vérification de cette propriété du modèle indique la non-existence de situations de blocage.

- Quelle que soit l'évolution du **réseau de Petri borné**, il existe une limite finie au nombre de jetons dans le réseau. Cette propriété peut être liée à la capacité limité de calcul ou de traitement d'un système.

- Dans un réseau **réinitialisable**, il est toujours possible de trouver une séquence de tir qui ramène au marquage initial. Cette propriété est très utile lorsqu'on modélise (et c'est très souvent le cas) des systèmes à caractère cyclique ou répétitif. ».

- A un niveau général, un marquage du réseau de Petri peut être interprété comme un état global du système et un changement du marquage correspond à une transition d'état.

La vérification des bonnes propriétés pourrait être un premier élément de vérification pour le concepteur. L'outil va attirer son attention sur les places ou transitions identifiées comme problématiques. Nous avons fait en sorte que HiLeS Designer affiche des alertes en indiquant les

possibles sources de problèmes, mais la correction de ces anomalies est une tâche réservée au concepteur.

Malheureusement, la tâche de récupération et d'interprétation des résultats s'avère compliquée, et, lorsque l'on n'est pas un spécialiste, il est très facile de se « noyer » dans la complexité des fichiers de sortie générés par TINA. Voyons ci-dessous les résultats de l'analyse du réseau de la Figure 3-7a.

```
Tina version 2.6.7 -- 05/28/04 -- LAAS/CNRS
mode -R
INPUT NET ----------------------------------------------------

parsed net exemple_TINA
4 places, 6 transitions

net exemple_TINA
tr Tr_1 Pl_1 -> Pl_2
tr Tr_2 Pl_2 -> Pl_3
tr Tr_3 Pl_2 -> Pl_4
tr Tr_4 Pl_3 -> Pl_1
tr Tr_5 Pl_4 -> Pl_1
tr Tr_6 Pl_3 Pl_4 -> Pl_1
pl Pl_3 (1)
pl Pl_4 (1)

0.000s
REACHABILITY ANALYSIS ----------------------------------------

bounded

14 marking(s), 26 transition(s)

MARKINGS:
0 : Pl_3 Pl_4
1 : Pl_1 Pl_4
2 : Pl_2 Pl_4
3 : Pl_4*2
4 : Pl_1 Pl_2
5 : Pl_2*2
6 : Pl_2 Pl_3
7 : Pl_3*2
8 : Pl_1 Pl_3
9 : Pl_1*2
10 : Pl_1
11 : Pl_2
12 : Pl_3
13 : Pl_4

REACHABILITY GRAPH:

0 -> Tr_4/1, Tr_5/8, Tr_6/10
1 -> Tr_1/2, Tr_5/9
2 -> Tr_2/0, Tr_3/3, Tr_5/4
3 -> Tr_5/1
4 -> Tr_1/5, Tr_2/8, Tr_3/1
5 -> Tr_2/6, Tr_3/2
6 -> Tr_2/7, Tr_3/0, Tr_4/4
7 -> Tr_4/8
```

```
8 -> Tr_1/6, Tr_4/9
9 -> Tr_1/4
10 -> Tr_1/11
11 -> Tr_2/12, Tr_3/13
12 -> Tr_4/10
13 -> Tr_5/10

0.000s

LIVENESS ANALYSIS ----------------------------------------------

not live

0 dead marking(s), 4 live marking(s)
0 dead transition(s), 5 live transition(s)

STRONG CONNECTED COMPONENTS:
1 : 0 1 2 3 4 5 6 7 8 9
0 : 10 11 12 13

SCC GRAPH:

1 -> Tr_4/1, Tr_5/1, Tr_6/0, Tr_1/1, Tr_2/1, Tr_3/1
0 -> Tr_1/0, Tr_2/0, Tr_3/0, Tr_4/0, Tr_5/0

0.000s

ANALYSIS COMPLETED ---------------------------------------------
```

Il s'agit encore d'un exemple très simple dans lequel la lisibilité reste accessible pour identifier facilement les bonnes propriétés. Cependant, dans des modèles plus grands, les états à évaluer peuvent générer des fichiers très longs ou même faire diverger la simulation. Cette complexité, ainsi que la difficulté de traduire les « bonnes propriétés » en « propriétés métier » nous ont amené à chercher des moyens plus conviviaux d'utilisation. De plus, la seule analyse des bonnes propriétés de réseau de Petri ne garantit pas une correspondance directe avec les caractéristiques opérationnelles du système. C'est pour cela que le besoin de réaliser d'autres analyses s'impose. En gros, l'objectif, au-delà du fait que le réseau soit borné, vivant ou reinitialisable, est de vérifier qu'il représente bien le comportement attendu du système.

Nous avons donc réfléchi à ce problème et nous avons conçu et développé une première version d'un mécanisme qui vise à rendre plus aisé l'usage de cet outil. L'idée est de pouvoir vérifier des propriétés concrètes à un ou plusieurs endroits du projet. Sur cette base le concepteur doit pouvoir « poser des questions » liées à son domaine de connaissances et associées au comportement du système. Ces questions doivent pouvoir s'associer aux éléments de description fournis par HiLeS, notamment les places et transitions des réseaux de Petri et les blocs contenant d'autres sous réseaux à l'intérieur. Notre proposition est de mettre en place des « observateurs » permettant de vérifier des conditions de fonctionnement très précises du système en bénéficiant de la représentation système de HiLeS. En pratique, au niveau de l'interface graphique de HiLeS Designer, les objets ayant un observateur associé sont étiquetés avec des « lunettes bleues » tel que l'illustre la Figure 3-8.

La mise en œuvre des observateurs n'a pas que des effets sur l'interface graphique. Au niveau de la passerelle HiLeS Designer – TINA, elle implique des modifications de structure des netlist. Il a fallu rajouter des lignes à la fin du fichier correspondantes aux observateurs et aux propriétés à vérifier.

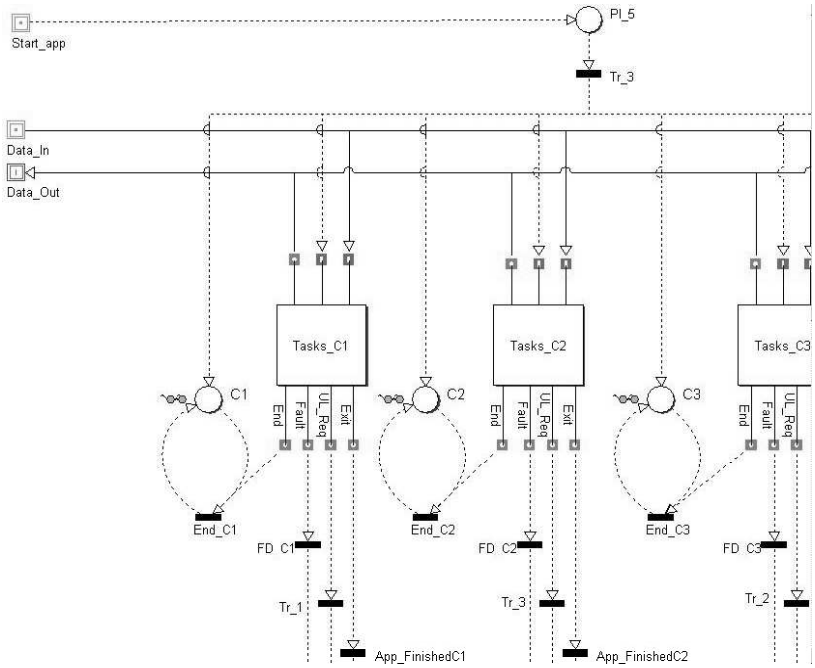
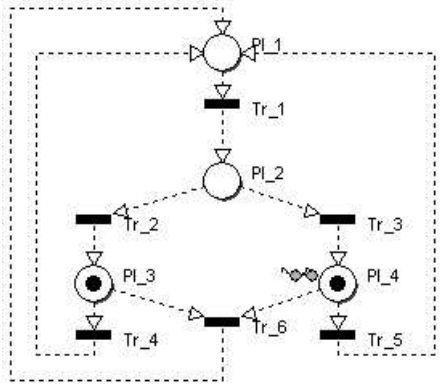


Figure 3-8. Illustration des observateurs représentés par des lunettes à coté des places dans un projet HiLeS Designer 0.

Afin d'illustrer de manière générale cette implémentation, reprenons l'exemple de la Figure 3-7a : Nous définissons un observateur sur la place Pl_4. Ensuite, en utilisant l'interface HiLeS Designer, nous y associons la propriété (PR_A) à vérifier. Notre outil génère donc, une netlist légèrement modifiée incluant ces observateurs. Sur la Figure 3-9a, nous montrons une capture d'écran de HiLeS Designer avec un observateur sur la place Pl_4. A droite (Figure 3-9b) de l'image, la netlist incluant en dernière ligne, on peut lire la propriété PR_A que l'on souhaite vérifier. TINA va pouvoir interpréter cette version de netlist dite « augmentée ».



a. Un observateur sur la place Pl_4

```
net Watchpoint_Analysis_exemple_tina2
tr Tr_1 [0,w[ Pl_1 -> Pl_2
tr Tr_2 [0,w[ Pl_2 -> Pl_3
tr Tr_3 [0,w[ Pl_2 -> Pl_4
tr Tr_4 [0,w[ Pl_3 -> Pl_1
tr Tr_5 [0,w[ Pl_4 -> Pl_1
tr Tr_6 [0,w[ Pl_3 Pl_4 -> Pl_1
pl Pl_3 (1)
pl Pl_4 (1)
pr PR_A Pl_4
```

b. Netlist équivalente du réseau.

Figure 3-9. Exemple de génération d'une netlist lorsqu'il y a des observateurs associés. Ici, l'observateur est placé sur Pl_4

Ainsi, en phase opérationnelle et grâce à ce principe des observateurs, l'utilisateur de HiLeS Designer pourra « poser » des questions simples et concrètes sur le comportement temporel général du système en cours de conception :

- des statistiques sur l'occurrence d'événements :
  - Les listes d'occurrences des événements
  - La liste des événements parallèles et concurrents
- la détermination de la nature cyclique du système,
- la détermination des blocages,
- l'estimation de la séquence globale du système en regroupant des événements consécutifs,
- la vérification de l'occurrence des conséquences suite à un événement particulier,
- événements non atteignables,
- ou bien la génération d'un graphe simplifié du comportement du système permettant de « lire » plus facilement ce que le réseau représente.

3.5 Passerelle HiLeS Designer vers VHDL-AMS

Une exigence forte de notre projet, est de permettre une approche pluridisciplinaire, générale et standardisée. Comme nous l'avons déjà précisé (§2.4.8) nous avons choisi VHDL-AMS comme langage support de représentation finale des modélisations HiLeS et comme élément terminal de notre plate-forme de conception système. Cette compatibilité est un choix technique et stratégique car le langage VHDL-AMS est bien adapté pour la description de systèmes pluridisciplinaires et car il s'agit d'un standard IEEE en pleine expansion scientifique et industrielle. Ceci assure la pérennité de nos modélisations et ouvre des grandes possibilités pour le partage de modèles et leur mise en opération sous la forme d'IP. Nous avons conçu HiLeS Designer pour faciliter cette transformation en nous appuyant sur la structure propre du langage VHDL-AMS.

Nous présentons dans cette section la manière dont les différents éléments du formalisme HiLeS sont traduits en VHDL-AMS ainsi que les mécanismes qui nous avons développé pour ce faire. Nous devons signaler qu'une grande partie de cette implémentation a été faite grâce aux travaux réalisés par nos collègues David GUIHAL et Julien BAYLAC pendant leurs stages [Gui03][BHE04].

3.5.1 Traduction des blocs HiLeS en VHDL-AMS

Une première étape fondamentale de la transformation d'un projet HiLeS Designer en VHDL-AMS est la traduction des blocs structuraux et fonctionnels. 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.

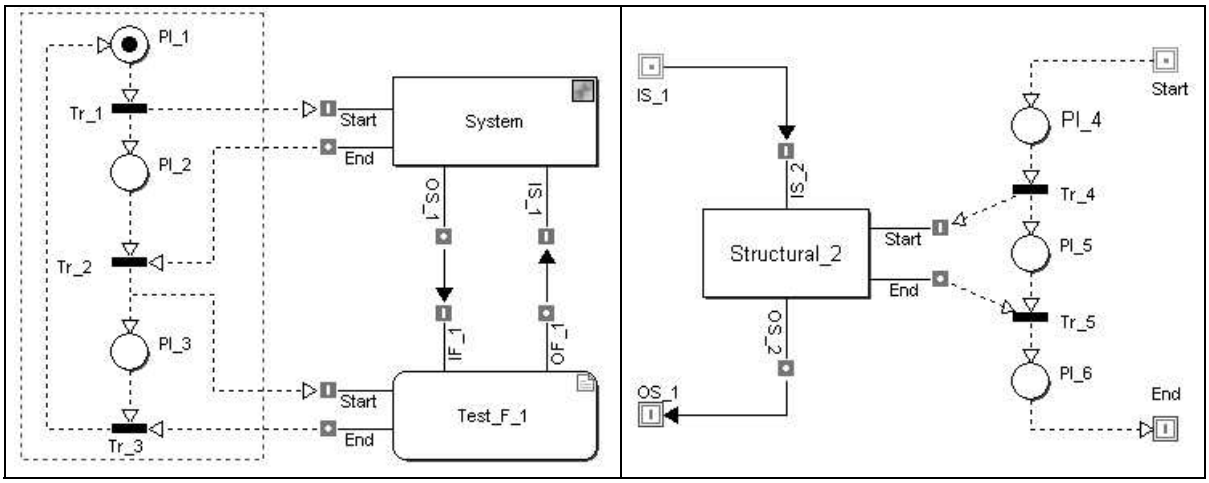
Ici, notre objectif est de montrer la manière dont l'on peut modéliser en VHDL-AMS les différents éléments du formalisme HiLeS. Cette étape est tout à fait nécessaire pour permettre à l'approche HiLeS de parvenir au stade souhaité du prototype virtuel. Le langage VHDL-AMS paraît le

mieux adapté à ce niveau général. En effet le langage de description matériel VHDL-AMS permet de décrire des modèles multi-abstractions, pluridisciplinaires, hiérarchiques continus et discrets.

3.5.1.1 L'interprétation du niveau zéro

Avant de rentrer dans les équivalences précises de chaque type de bloc, il est pertinent d'analyser le cas particulier de notre premier niveau de représentation : le niveau zéro. Celui ci va représenter le système dans son contexte. Nous pouvons considérer l'environnement du projet ou niveau zéro, proposé précédemment (§3.3.1), non seulement comme son environnement opérationnel mais aussi comme son environnement de test. Dans ces conditions, la représentation de plus haut niveau du projet créé sous HiLeS Designer peut être modélisée sous VHDL-AMS comme une entité **TestBench**. L'utilisateur pourra aussi le réutiliser avec d'autres configurations du système grâce à notre implémentation des architectures multiples (§2.9.1.3) tout à fait cohérente avec le langage VHDL-AMS. Le concept de TestBench sert à modéliser un système et son environnement afin d'en évaluer son fonctionnement. Nous pouvons établir ce rapprochement entre les concepts de niveau zéro HiLeS et le testbench de VHDL-AMS car l'un comme l'autre représentent le niveau d'abstraction le plus élevé de chaque modélisation, qui va permettre de modéliser le système et son environnement. L'architecture de ce niveau zéro 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 sont réalisés automatiquement lorsque l'utilisateur relie les différents blocs entre eux par le biais de l'interface graphique.

La **Figure 3-10 a, b**, montre un exemple de schématique type créé sous HiLeS Designer. A gauche un exemple de niveau zéro, le réseau de pétri cadence l'exécution des actions dans l'environnement du système. Pour simplifier la lisibilité de ce niveau, nous pourrions imaginer cette procédure construite avec des réseaux de Pétri regroupée en un seul bloc de commande, comme le suggère le pointillé rouge de la figure. Le bloc Test_F_1 et le réseau de Petri comportent l'environnement du « System ». A droite nous présentons l'architecture du bloc « System ».



a. Exemple de niveau zéro b. Détail du bloc Structural_1, niveau -1

Figure 3-10. A gauche, une possible représentation d'un TestBench sous HiLeS. A droite, le contenu du bloc « System ».

3.5.1.2 Interprétation des blocs structurels

En général, nous modélisons les blocs structurels comme des entités dont l'architecture est décrite de manière structurelle en instanciant et en reliant les différents blocs contenus à l'intérieur. Dans notre exemple, l'architecture de *« System »* est montré dans la Figure 3-10b. L'entité possède des entrées/sorties qui interviennent dans l'architecture qui correspondent à celles qui ont été définies au niveau zéro. Ainsi au niveau 0, nous réalisons la définition du nom de l'entité lors de la création du bloc, son « port map » et son « generic map ». Tel que nous l'avons déjà présenté (Figure 2-32), la section d'information de chaque bloc, l'outil HiLeS Designer permet de créer et de définir les paramètres génériques.

Concernant les ports, les interfaces développées permettent de spécifier le nom, l'objet (signal, terminal, quantité), le type (std_ulogic, bit, etc…) ou la nature (electrical, thermal, etc…). 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. Pour s'assurer une meilleure compatibilité avec les principaux outils existants, il est préférable de laisser à l'utilisateur le soin d'inscrire les bibliothèques qu'il souhaite utiliser pour son modèle. Initialement notre objectif était de ne pas associer HiLeS Designer à un simulateur en particulier, mais au fur et a mesure que nous progressons dans le projet, nous pensons à la possibilité de proposer des « templates » ou formats prédéfinis orientés vers le logiciel de simulation souhaité par l'utilisateur. Cette solution s'impose car les noms des bibliothèques varient suivant l'outil que l'on utilise pour faire la simulation.

Par rapport à la définition de l'architecture, l'utilisateur peut aller à l'intérieur du bloc en utilisant la fonction « Go Inside » (décrite en §2.9.1.1). Cette option permet de définir le nom de l'architecture ou de choisir parmi d'autres déjà définies pour l'entité ou bloc père du niveau supérieur. Les ports d'entrées\sorties sont répertoriés et transcrits automatiquement au niveau de l'architecture en cours de définition (Figure 3-10 b).

A terme, nous pourrons envisager l'utilisation d'un bloc déjà créé et disponible dans une bibliothèque, ayant la possibilité de choisir son architecture s'il en existe plusieurs et préciser ses paramètres génériques. Cela équivaut en VHDL-AMS à instancier une entité déjà compilée et stockée dans une bibliothèque.

3.5.1.3 Interprétation des blocs fonctionnels

Ainsi que les blocs structurels, les blocs fonctionnels représentent eux aussi une entité. Lors de leur création sous HiLeS Designer on peut nommer l'entité et définir le « generic map ». Le « port map » est généré automatiquement par HiLeS Designer en fonction des connexions du niveau supérieur. Concernant l'architecture de l'entité, elle est accessible directement par le biais de l'interface présentée en §2.9.1.6. L'utilisateur peut écrire directement l'architecture VHDL-AMS en utilisant l'éditeur inclus dans l'outil. Rappelons ici que ce type de bloc ne comporte pas, au niveau de HiLeS, des propriétés hiérarchiques. Toutefois, cela n'empêche pas le concepteur de définir des hiérarchies directement dans le code en instanciant des composants précédemment compilés.

A l'heure actuelle, HiLeS Designer ne permet qu'une architecture par bloc fonctionnel, mais on peut envisager de créer plusieurs architectures pour une même entité (bloc). Le choix de l'architecture pourrait se faire parmi une liste d'architectures disponibles, de la même façon dont nous pouvons définir l'architecture par défaut (§2.9.1.3) d'un bloc structurel. La traduction de cette possibilité en

VHDL-AMS consiste à la création d'une configuration ou d'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, telle architecture ne pourra être compilée que si l'entité à laquelle elle est associée à 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.

3.5.1.4 Interprétation des canaux

Sous HiLeS Designer, 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 choses compatibles.

Selon le sens des canaux HiLeS, les sens des ports sont forcés en mode in ou en mode out, sauf dans le cas des terminaux qui, par définition de la norme, n'ont pas de mode. Dans ce dernier cas, la direction des canaux a du sens uniquement au niveau de la représentation graphique comme une guide pour le concepteur. En spécifiant le champ type d'un port, on force le type des canaux reliés à ce port et inversement. Il est très souhaitable d'élargir la vérification de cohérence des connexions sous HiLeS. Pour l'instant, nous vérifions uniquement le type dit HiLeS (continu, discret ou arche de réseau de Petri) et non leur contenu proprement dit, leur définition VHDL-AMS. Ceci est nécessaire pour garantir la cohérence entre niveaux hiérarchiques et dans les connexions entre éléments de même niveau. Dans la version actuelle de HiLeS Designer, nous avons inclu la possibilité d'accéder aux caractéristiques des ports. Lors de la création d'un port, l'utilisateur peut le configurer en utilisant l'interface suivante :

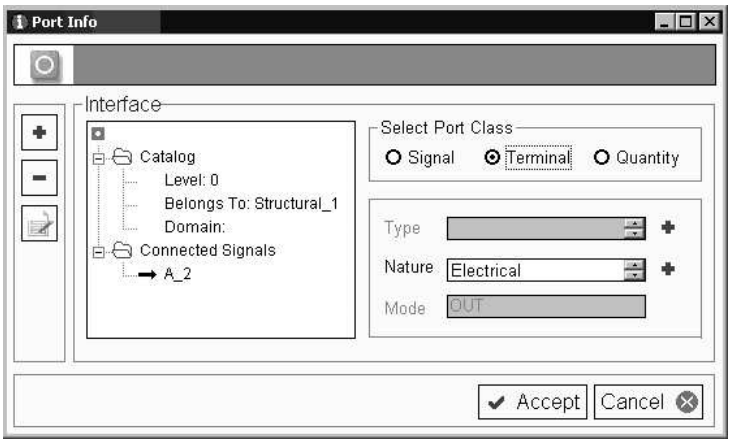


Figure 3-11. Fenêtre de configuration des ports.

3.5.1.5 Problème : Les bibliothèques du projet

Une des difficultés importantes lors de l'implémentation d'une passerelle avec le langage VHDL-AMS est l'utilisation des bibliothèques dans les projets. Le concepteur doit pouvoir choisir les différentes bibliothèques utiles à son entité. Cependant ce choix est fortement lié à l'outil final de simulation. En l'état, l'utilisateur doit définir lui-même les bibliothèques qu'il utilise dans chaque bloc. Cette méthode permet à HiLeS de ne pas être dépendant d'un outil de simulation et permet aussi à l'utilisateur d'appeler ses propres bibliothèques. En revanche, elle exige un effort supplémentaire de sa part au moment de compiler le projet en utilisant un logiciel en particulier car les noms de ces bibliothèques et de ces packages peuvent varier d'un outil de simulation à un autre. En conséquence, une adaptation spécifique du projet est impérative. Tel que nous l'avons mentionné précédemment (§3.5.1.2) nous envisageons la définition de « templates » ou formats spécifiques pour rendre possible la pré-configuration d'un projet HiLeS Designer selon l'outil ciblé.

3.5.2 Traduction des réseaux de Petri

Pour la génération complète du modèle, nous devons réaliser la traduction du réseau de contrôle en VHDL-AMS. Tel que nous l'avons déjà présenté, dans notre démarche HiLeS le modèle de commande d'un système est basé, sur les réseaux de Petri. Ils représentent les modes d'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 formalisme 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. Notre objectif est de décrire le réseau de Petri en langage VHDL-AMS.

Nous pouvons tout d'abord transformer le réseau de Petri en une FSM (machine d'états finis) qui est facilement modèlisable en VHDL-AMS, ils existent des outils commerciaux qui réalisent cette opération : [Syn99], ou bien [CDS04b]. 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. Mais les FSM ne sont pas adaptées pour exprimer la concurrence. Ainsi, lorsque la complexité de la spécification augmente, on va constater une croissance exponentielle du nombre d'états. Même si les FSM sont bien adaptées pour modéliser des processus séquentiels, les réseaux de Petri décrivent naturellement des processus concurrents, ainsi un simple réseau de Petri peut devenir une FSM complexe.

Une autre possibilité consiste à créer des entités place et transition directement en VHDL-AMS et de les connecter de façon structurelle. C'est cette deuxième option qui nous avons retenu par la suite.

3.5.2.1 Une première tentative : le langage CONPAR

Une méthodologie déjà appliquée dans l'industrie a été proposée au Portugal par [FAP97]. Cette méthode permet la génération automatique de code VHDL et supporte les spécifications hiérarchiques. Pour garder l'isomorphisme entre les spécifications initiales du réseau de Petri et les instructions VHDL, la méthode implique d'utiliser un langage intermédiaire : CONPAR.

Dans ce 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. Une transition est décrite comme une règle conditionnelle :

<label> : <Preconditions> |- <PostConditions> ;

Les préconditions et les post conditions sont respectivement formées par les symboles des places d'entrée et de sortie. Quand la précondition d'une règle est satisfaite, les postconditions sont déclarées vraies. Par exemple pour la transition 1 (t1) de la Figure 3-12, on a :

t1 : p1 * x1 |- p2 * p3 * y1 ;

La traduction en VHDL est alors :

t1 <= x1 AND p1 AND NOT p2 AND NOT p3 ;
t2 <=p2 and not p4

Les préconditions d'une transition sont directement traduites via une expression booléenne en VHDL. Les postconditions sont distribuées parmi les expressions VHDL Npi qui sont des signaux VHDL qui modélisent l'entrée des bascules. Pour chaque signal de sortie, un signal concurrent est utilisé. Par exemple :

Np1 <= t5 OR (p1 AND NOT t1) ;
y1 <= p4 OR t1 ;

Ainsi la description d'un SIPN (réseaux de Petri interprété synchrone) utilise l'affectation de signaux concurrents, incluant une liste de transitions, de signaux Np (marquage d'une place), et de signaux de contrôle de sortie.

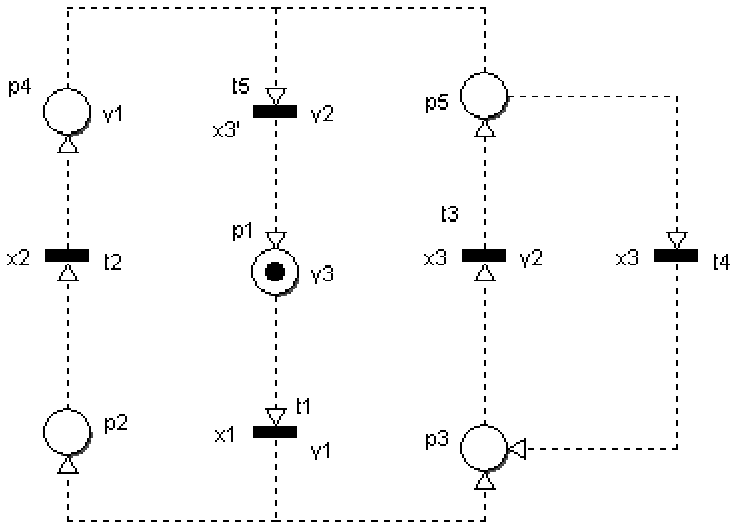


Figure 3-12. SIPN spécification d'un contrôleur (SIPN : Réseau de Petri Interprété Synchrone)

La modélisation d'un SIPN nécessite l'ajout d'une horloge globale qui permet d'actualiser le marquage des places. Il semble judicieux de choisir l'horloge du réseau de Petri au moins deux fois supérieure à l'horloge du système. En effet il est nécessaire que dès qu'un signal « acknowledge » s'active et que les conditions de passage d'une transition sont réunies, le passage de jeton entre deux places s'exécutent instantanément. Ainsi il est nécessaire que la période de l'horloge CONPAR soit assez faible pour qu'il y ait un front montant au moment de la réunion des conditions.

Nonobstant, il est tout à fait souhaitable de rester le plus cohérent possible avec la représentation de base des réseaux de Petri, donc d'avoir un franchissement des transitions asynchrone. Pour ce faire, nous proposons de modifier la méthodologie CONPAR en rendant les *process*, en affectant les signaux Npi aux signaux Pi, sensibles aux signaux d'entrée (Init, etc …) et aux signaux Npi, au lieu des fronts montants de l'horloge. On utilise l'instruction *wait on* en début de « *process* » pour réaliser cela.

Les essais réalisés avec la méthode CONPAR ont été satisfaisants. Mais nous trouvons gênant le besoin de passer par une représentation intermédiaire, particulièrement à mesure que la complexité des structures augmente. Ce passage complique sérieusement l'implémentation d'un mécanisme de génération automatique de code à partir de HiLeS Designer. De plus, ceci implique une transformation supplémentaire dans notre démarche. En conséquence, nous estimons plus convenable d'utiliser une approche dans laquelle le passage vers une représentation VHDL est immédiat, c'est à dire avec une correspondance directe entre places et transitions et leurs modèles équivalents.

3.5.2.2 L'approche par composants

Nous avons donc, cherché une méthode plus directe permettant de construire un modèle équivalent en VHDL de manière directe avec les mêmes éléments de base des réseaux de Petri, c'est à dire en utilisant des places et des transitions. Pour ce faire, nous avons pris, sur une suggestion de notre collègue du LAAS, A. NKETSA [CEN04], une approche par composants [ABG04] présentée par l'équipe de monsieur David ANDREU du LIRMM. Dans cette approche, deux composants de base sont utilisés pour générer la totalité de l'ensemble.

Dans le schéma de la Figure 3-13, nous présentons la méthode que nous avons implémenté dans HiLeS Designer pour convertir la représentation graphique des réseaux de Petri en une représentation écrite en langage VHDL-AMS. De la même manière que nous l'avons fait pour générer la netlist de TINA, nous parcourons d'abord la totalité du projet à la recherche des places, des transitions et des arcs. Avec ces éléments et en utilisant un composant VHDL pour les *Places* et un autre pour les *transitions* nous construisons un fichier .vhd équivalent. Nous reproduisons la topologie entièrement sur ce fichier. Il est nécessaire de préciser que le modèle utilise des signaux supplémentaires pour assurer la gestion des jetons dans le réseau : Ajouter et/ou enlever. Nous avons réalisé cette deuxième implémentation en utilisant le logiciel SystemVision de Mentor Graphics.

Le fonctionnement des composants représente bien celui des réseaux de Pétri. Les *transitions* attendent que toutes leurs *places* amont soient marquées pour ensuite être franchies et, ainsi, placer un jeton sur chacun des jetons en aval. Pour réaliser cette gestion des jetons, le modèle dispose des signaux qui « remontent » dans le réseau afin d'enlever les jetons de leurs places en amont une fois que les transitions son franchies. Le modèle prévoit aussi une indication permanente du marquage du réseau.

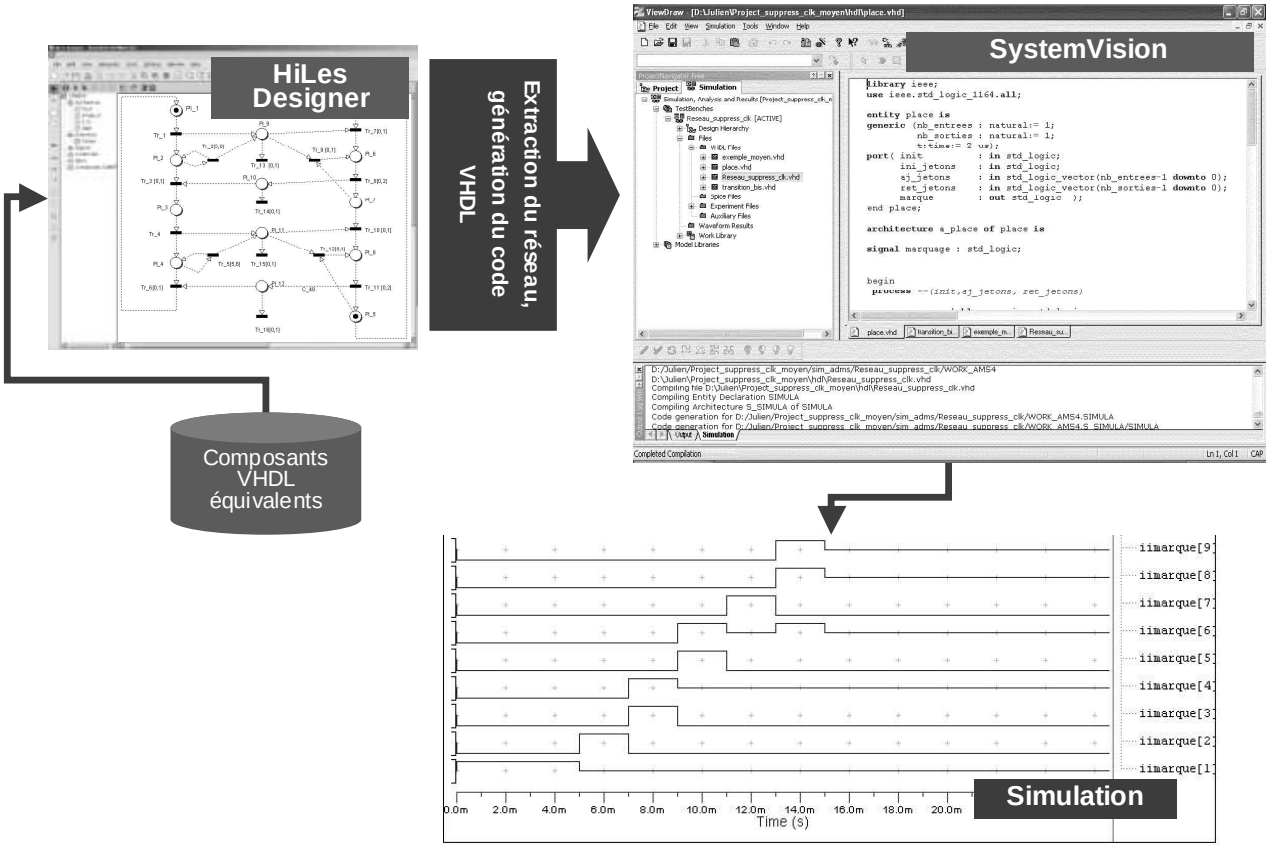


Figure 3-13. Schéma de fonctionnement du mécanisme d'extraction des réseaux de Petri vers VHDL-AMS.

Par rapport aux interconnexions entre objets, du modèle de composants de nos collègues du LAAS et du LIRMM [ABG04], les places et les transitions comportent des entrées et des sorties de taille variable, elles sont construites avec des vecteurs de taille n qu'il faut initialiser selon la topologie du réseau. Par exemple, une transition avec deux arcs d'entrée et trois arcs de sortie aura un vecteur d'entrée de deux positions et un vecteur de sortie de trois, ce principe est aussi appliqué pour les places. Cette caractéristique donne une bonne capacité de configuration et de la flexibilité à l'approche. En revanche, elle la rend difficile au moment de l'utiliser directement avec des outils de saisie schématique.

Suivant le principe considéré précédemment (§3.5.2.1), en modifiant la méthode CoNPAR, nous avons tenté de modéliser le fonctionnement de base des réseaux. De cette manière, il est possible de concevoir un modèle asynchrone, dans la mesure où c'est la présence ou non d'un jeton et la possibilité ou non de franchissement de transition qui fait fonctionner le modèle et non le cadencement d'une horloge extérieure. Malgré tout, il est évident que rares sont les systèmes en électronique qui ne disposent pas d'horloge. Donc, selon les besoins de l'utilisateur, il est intéressant d'avoir le choix des deux modèles.

Quel que soit le modèle choisi, la démarche a été la même, c'est-à-dire l'encapsulation. De fait, après avoir créé les deux composants, on appelle ces derniers depuis un autre fichier, que nous pouvons considérer comme un macro composant qui a pour rôle la description complète de la topologie du réseau à modéliser. Puis on utilise un testbench afin de pouvoir gérer les paramètres du réseau (les conditions de franchissement des transitions, la génération de l'horloge, l'initialisation du

réseau...). Nous présentons ci-dessous les détails des composants utilisés, leur version asynchrone et synchrone :

Composant place asynchrone

Le signal *init* permet d'imposer ou non un marquage initial de la place. Si *init* est à '1' alors la marque de la place prend la valeur de *ini_jetons*. Sinon on rentre dans une boucle de calcul de la marque en tenant compte du nombre de jetons qui arrivent sur la place (*aj_jetons*) et ceux qu'on enlève (*ret_jetons*). Ensuite la valeur logique du résultat est chargée dans *marque.*

Figure 3-14 Composant place asynchrone

```
generic (nb_entrees : natural:= 1;
         nb_sorties  : natural: = 1;
         t:time:=2 us);
port(init    : in std_logic;
     ini_jetons: in std_logic;
     aj_jetons : in std_logic_vector(nb_entrees-1 downto 0);
     ret_jetons: in std_logic_vector(nb_sorties-1 downto 0);
     marque    : out std_logic );
```

Composant transition asynchrone

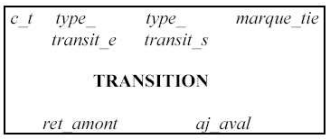


Figure 3-15. Composant transition asynchrone

```
generic (   nb_entrees    natural:=2;
        nb_sorties  : natural:=2
        t:time:=2 us);

port(   c_t             : in  std_logic;
    type_transit_e  : in std_logic_vector((nb_entrees-1) downto 0);
    type_transit_s  : in std_logic_vector((nb_entrees-1) downto 0);
    marque_tie      : in std_logic_vector((nb_entrees-1) downto 0);
    ret_amont       : out std_logic_vector((nb_entrees-1) downto 0);
    aj_aval         : out std_logic_vector(nb_sorties-1 downto 0));
```

c_t : condition associée à la transition ; *type_transit_e & type-transit_s* : précise le type d'arc d'entrée de la transition (arc normal, inhibiteur, test) ; *marque_tie* : marquage des places d'entrées ; *ret_amont* : marque à retirer des places amont de la transition ; *aj_aval* : marque a ajouter dans les places aval de la transition

Pour le modèle asynchrone, nous rajoutons le paramètre générique temporel (t) sur les composants *place* et *transition*. En fait celui ci correspond au temps minimum pour calculer la marque de la place. Pour ce faire, il faut imposer ce temps soit sur le composant place soit sur le composant transition. C'est à nous de l'utiliser sur l'un ou sur l'autre en fonction du besoin.

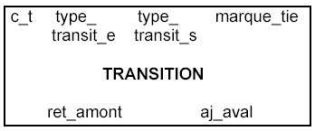
Composant place synchrone

Figure 3-16. Composant place synchrone

```
generic(nb_entrees : natural:= 1;
        nb_sorties  : natural: = 1);

port(   clk         : in std_logic;
        init        : in std_logic;
        ini_jetons  : in std_logic;
        aj_jetons   : in std_logic_vector((nb_entrees-1) downto 0);
        ret_jetons  : in std_logic_vector((nb_sorties-1) downto 0);
        marque      : out std_logic )
```

Composant transition synchrone




```
generic(nb_entrees     : natural:=2;
        nb_sorties  : natural:=2);

port(   c_t             : in  std_logic;
        type_transit_e  : in std_logic_vector((nb_entrees-1) downto 0);
        type_transit_s  : in std_logic_vector((nb_entrees-1) downto 0);
        marque_tie      : in std_logic_vector((nb_entrees-1) downto 0);
        ret_amont       : out std_logic_vector((nb_entrees-1) downto 0);
        aj_aval         : out std_logic_vector((nb_sorties-1) downto 0);
```

Figure 3-17 Composant transition synchrone

Pour le modèle synchrone nous supprimons le paramètre générique temporel (t). En revanche, nous rajoutons une entrée supplémentaire pour l'horloge sur le composant place et nous modifions légèrement les fichiers.

Pour l'implémentation du modèle sous SystemVision, nous avons dû faire une modification sur les composants de base. Dans le modèle initial un package avait été créé pour pouvoir gérer facilement le type d'arc attaquant les transitions. Le package déclarait un tableau de vecteur de deux bits et de longueur le nombre d'arcs entrant (voir ci-dessous).

```
library ieee;
use ieee.std_logic_1164.all;
package rdp is
type typ_arc_vector is array(natural range <>) of std_logic_vector(1 downto 0);
component place
…..
end component;
component transition
……
end component;
end rdp;
```

Les trois types d'arcs existants sont définis de la manière suivante :

```
constant arc_classique      :std_logic_vector(1 downto 0) :="00";
constant arc_test           :std_logic_vector(1 downto 0) :="10";
constant arc_inhibiteur     :std_logic_vector(1 downto 0) :="11";
```

Parmi eux, uniquement l'arc classique est utilisé dans HiLeS. Nous avons gardé les autres en tant que constituants du modèle de base.

Pour décrire l'arc qui arrive sur la transition, on utilise donc les deux vecteurs :

```
type_transit_e      : in  std_logic_vector((nb_entrees-1) downto 0);
type_transit_s      : in  std_logic_vector((nb_entrees-1) downto 0);
```

Le premier bit des signaux *arc_classique, arc_test, arc_inhibiteur* correspond au vecteur type_transit_e et le deuxième bit correspond à type_transit_s.

Aussi pour mettre en œuvre le modèle asynchrone, nous avons utilisé l'instruction « wait on » sur le composant *place.* En fait cette l'utilisation des *« wait on »*' permet d'attendre un évènement sur le signal déclaré. Ainsi le calcul de marquage ne se fait pas avant un changement de celui-ci.

Cependant l'insertion d'un délai (after) a été nécessaire pour assurer la prise en compte de tous les événements.

Ainsi donc le composant place a un paramètre générique temporel qui doit être absolument en adéquation avec le temps imposé pour l'initialisation du réseau via le signal "*init*". Cependant ce facteur possède un large domaine de validité, il peut varier entre une dizaine de microsecondes et une centaine de picosecondes. La seule obligation est la cohérence entre les deux paramètres temporels, celui du composant place et celui du testbench. L'annexe B illustre un exemple de traduction d'un reseau de Petri en VHDL simulé sous un outil VHDL-AMS et généré automatiquement par HiLeS Designer0.

3.5.3 Connexion des modèles équivalents des blocs et des réseaux de Petri

Avec les procédures de traduction présentes dans les deux sections précédentes, nous pouvons construire un modèle complet équivalent VHDL-AMS. Pour illustrer le principe de connexion reprenons l'unité fondamentale présentée dans la section 3.3.1. Dans la Figure 3-18 nous faisons un « Zoom » sur les *transitions* et leurs connexions au bloc. Le principe de fonctionnement est illustré par le chronogramme à l'intérieur du bloc : Le franchissement de la *transition* Tr1 marque un jeton sur la *place* P1, il est interprété comme le signal de démarrage pour les activités du bloc. Ce signal est pour le réseau de Petri, un jeton qui est transféré à une *place* connectée en aval de la transition Tr1. Une fois que les processus internes sont achevés ou lorsqu'un état « interne » donné est attendu, le bloc renvoi un signal à la transition Tr2. Ce signal est interprété par la transition comme un jeton qui est placé sur une de ces places d'entrée. La transition Tr2 sera franchie lorsque toutes ses *places* sont marquées.

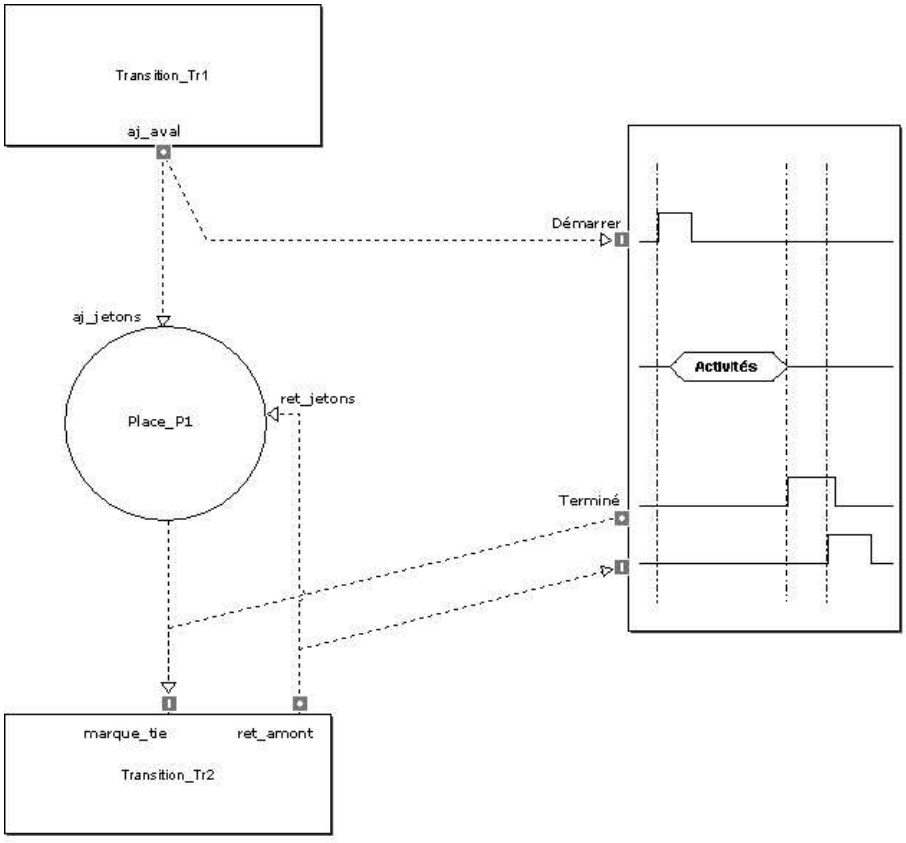


Figure 3-18 Connexion fondamentale des modèles VHDL des réseaux de Petri et des blocs HiLeS

Les signaux qui « remontent » de la *transition* Tr2 vers la *place* P1 et vers le bloc ne sont pas visibles sous HiLeS Designer. Ils servent à contrôler le passage des jetons mais restent cachés à l'utilisateur. Ces signaux appartiennent au modèle des réseaux de Petri en composants VHDL (§3.5.2.2) présenté précédemment. La gestion de ces signaux implique la mise en place d'une logique générale et nécessaire pour la connexion entre les réseaux et les blocs. Ce comportement logique de base est inclus dans les blocs et permet de les connecter avec les *transitions* en reproduisant le comportement d'une place. Nous pouvons alors considérer les blocs en général comme des *macro places* dont le marquage apparaît une fois que sa fonction interne a été réalisée.

En guise d'exemple, la Figure 3-19 illustre un bloc fonctionnel cadencé par un réseau de Petri simple. Le franchissement de la transition Tr_0 initialise les activités du bloc *Genb1*. Ce bloc fonctionnel comporte une fonction de comparaison permettant de détecter le dépassement d'un seuil par rapport au signal d'entrée *Input1.* Le bloc habilitera le franchissement de la transition Tr2 lorsque le signal d'entrée attendre la valeur du seuil. La comparaison aura lieu uniquement à partir du moment ou la place PL_1 soit marquée.

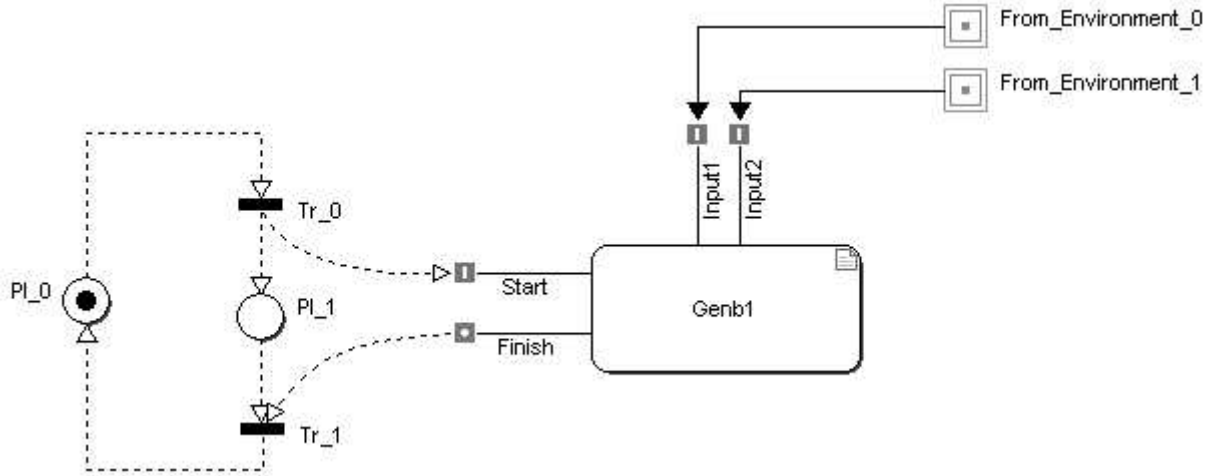


Figure 3-19. Exemple du couplage du réseau de Petri et un bloc fonctionnel.

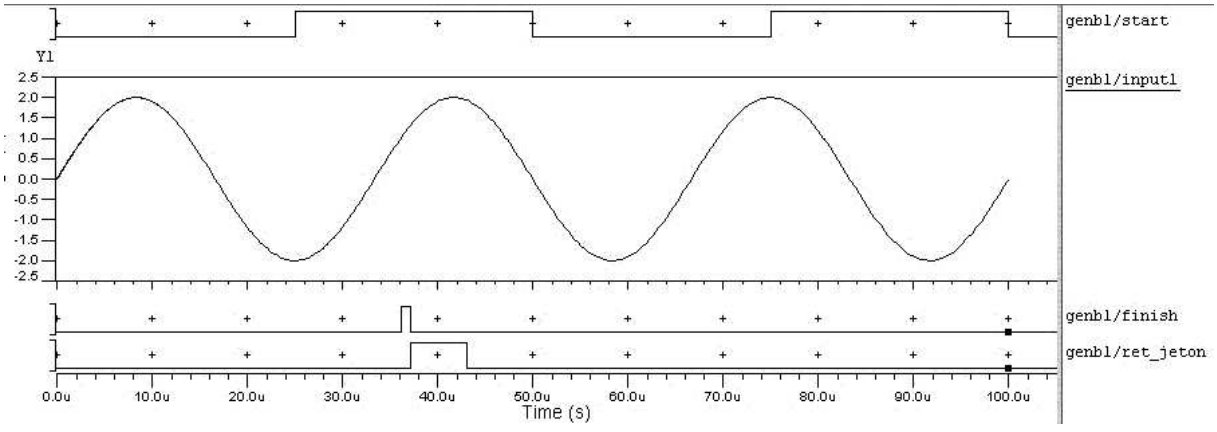


Figure 3-20. Simulation de l'exemple de détection de seuil dans un modèle HiLeS.

Le bloc *Genb* comporte la description détaillée de la fonction en VHDL-AMS. Nous avons réalisé la simulation de cet exemple sur System Vision de Mentor Graphics. La Figure 3-20 montre d'abord le signal *start (genbl1/start)* qui « pilote » l'exécution de la comparaison. Le deuxième signal

est l'entrée de *input1 (genbl1/start).* Le troisième est le signal finish qu'indique l'occurrence de l'événement attendu pour sortir du bloc. Ce signal restera actif en attendant la confirmation du franchissement de la transition aval (Tr_2 dans la Figure 3-19).

3.6 Validation, vérification et certification

Notre objectif ici, est de revenir sur le rôle que pourrait jouer notre démarche HiLeS et la conception descendante telle que nous l'appréhendons, pour aider à des procédures de validation, vérification, certification dont nous venons de préciser la terminologie.

HiLeS Designer est un outil graphique travaillant en blocs fonctionnels interconnectés et hiérarchisés.

L'écriture de ces blocs et les modèles HiLeS associés sont compatibles VHDL-AMS de telle sorte que l'on peut simuler la fonctionnalité de chaque bloc et la fonctionnalité globale (les modèles utilisés sont des modèles d'ingénieur ; équations différentielles, fonctionnes logiques…).

La gestion temporelle de ces blocs est décrite par des Réseaux de Petri simulables sur TINA. Le simulateur TINA vérifie la bonne exécution des modes opératoires déduite des objectifs généraux du produit et des considérations générales d'utilisation.

Les blocs sont créés par le concepteur mais une bibliothèque de blocs et de modèles de l'ingénieur facilite la description du système.

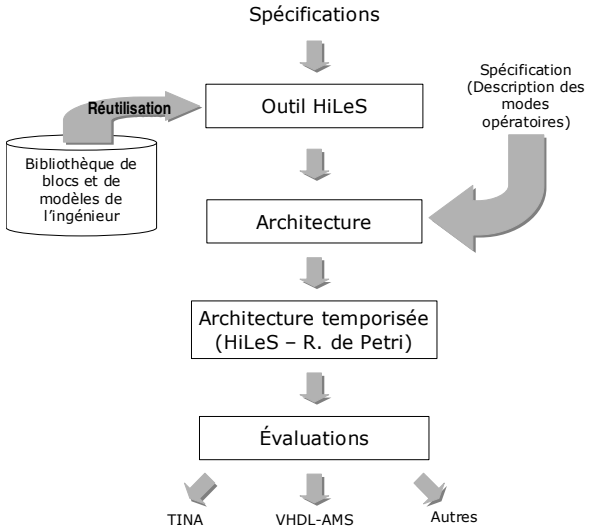


Figure 3-21. Etapes préliminaires de la démarche.

Pour illustrer les enjeux, voici quelques questions générales intéressant le triptyque : validation, vérification, certification :

- Les spécifications :
  - complètes et correctes ?
  - réalisables ?
- Conception « sans faute » ? (bon par construction)

- Vérification des fonctions attendues du système
- Détection de fonctions inattendues ?
  - Détection de modes opératoires de défaillance
- Evaluation de modes de fonctionnement dégradé ?
  - Modes de fonctionnement impossibles
- Estimation de marges de fonctionnement : Optimisations et tolérances ?
- Autres ?

3.6.1 Le niveau « validation »

Il se pose à l'ingénieur lorsqu'on lui amène :

- Des spécifications pour un produit nouveau,
- Des spécifications de modification sur un produit déjà existant.

La question qui lui est posée est : est-ce que l'on va savoir faire ?
Cette question est en fait, multiple :

- Est-ce que l'on sait concevoir ?
- Est-ce que l'on sait réaliser ?
- Est-ce que l'on sait produire ?
- Est-ce que l'on sait vérifier chacune des étapes ?

Les réponses ne peuvent être qu'estimations d'autant plus imprécises que la part de « re-use » est faible…

L'expérience de l'ingénieur et de l'équipe va jouer un rôle essentiel : Il va puiser dans son expérience et dans son expertise pour donner cette estimation qu'il argumentera par des calculs, si possible.

On voit bien que HiLeS peut jouer un rôle très positif en validation,:

- Parce qu'il propose une **modélisation simple, générique, que l'on doit pouvoir nourrir de l 'expérience cumulée des ingénieurs (base de données experte).**
- Parce qu'il peut associer dans une représentation unique les modélisations précises des parts de « re-use » qui seront retenues et des modélisations « expertes » telle que nous venons de les présenter.
- Parce qu'il servira de base à la construction de scénarios techniques multiples que l'on pourra comparer par simulation.

3.6.2 Le niveau « Vérification »

Vérifier, c'est apporter la preuve que l'on peut satisfaire une exigence prévue, dans un environnement précis. Cette preuve sera préférablement démontrée formellement, et, le cas échéant, démontrée par des simulations multiples.

De ce point de vue TINA est l'opérateur de vérification des représentations HiLeS. Le mettre en œuvre suppose :

- De formuler les questions pour TINA, en regard de chacune des exigences.

- D'interpréter le résultat en terme de preuve.

Cette procédure de test, doit, comme il est indique dans notre chapitre 2 (Figure 2-8), s'appliquer aux deux étapes :

- En amont, au stade de la définition fonctionnelle du système et de sa mise en architecture.
- Au niveau du prototypage virtuel, lorsque le modèle des composants réels du système ont été choisis

On notera que la représentation HiLeS amont est en complète continuité avec la modélisation VHDL-AMS du prototype virtuel. Ainsi, les questions prises en amont restent valables pour la vérification finale du prototype virtuel.

3.6.3 Le niveau de certification

Ce niveau ne peut pas être traité par la seule représentation fonctionnelle, quelle soit au niveau architectural, HiLeS ou au stade du prototypage virtuel VHDL-AMS, **car on ne peut vérifier que les fonctions attendues du système.**

Or, il y a de très nombreux mécanismes de défaillance qui peuvent induire un disfonctionnement de manière indirecte : par exemple la destruction d'une connexion par contrainte mécanique n'est pas incluse dans les descriptions de HiLeS ou VHDL-AMS.

HiLeS ou VHDL-AMS ne proposent qu'une modélisation fonctionnelle, comportementale, **qu'il faut donc compléter pour des objectifs de fiabilité prédictive ou de certification, d'identification et de modélisation des fonctions inattendues**. On peut ainsi imaginer d'étendre le rôle du prototypage virtuel vers d'autres intérêts de conception, grâce à un prototypage plus complet et plus spécialisé sur le type d'analyse visée. Pour l'instant nous apercevons un rôle actif uniquement dans une éventuelle modélisation du mode de fonctionnement dégradé des systèmes si c'est décrit explicitement dans les spécifications.

3.6.4 Perspectives dans ce domaine

En résumé, la conception système pose les problèmes de validation, de vérification et de certification. La démarche « HiLeS », bien documentée des modélisations expertes et des modélisations de « reuse », peut être un outil très intéressant d'estimation de la faisabilité d'un système ou d'aménagement d'un système existant.

Le couplage HiLeS-TINA introduit une procédure possible de vérification intégrée sur les propriétés fonctionnelles et comportementales au niveau amont et au niveau du prototypage virtuel.

La certification implique que soient associés aux fonctions et performances attendues du système, les fonctions inattendues, parasites, qui peuvent induire des erreurs ou des défaillances… La méthode est à découvrir et à implanter pour traiter cette question.

Il y a beaucoup de travail à faire pour compléter les outils et les méthodes dans les applications : Validation et vérification.

Dans cette perspective, que peut apporter une démarche telle que HiLeS à la validation et à la vérification d'une application électronique embarquée ?

L'inventaire des possibilités suivant est à compléter et à approfondir :

- Recherche et exploration d'architectures Hardware-firmware : définition des meilleurs choix possibles.
- Modélisation comportementale de la spécification (avec un haut niveau d'abstraction).
- Démonstration que l'architecture fonctionnelle est correcte par construction.
- Recherche et exploration d'architectures sur des paramètres non fonctionnels (consommation, nombre de connections, masse, taille…) : ces paramètres dits industriels sont dimensionnants pour l'électronique.
- Obtention d'un ensemble de vecteurs de tests pour vérifier à tous les niveaux d'implémentation que la conception est bonne par construction.
- Obtention d'un modèle fonctionnel décrit dans un langage standardisé, donc portable.
- Diminution du temps de génération des vecteurs de test à chaque étape, puisque l'on disposerait d'un ensemble de référence (lié à la modélisation haut niveau). Dans les étapes descendantes de la conception, l'effort serait porté sur les vecteurs complémentaires, pour tester des cas aux limites…
- Facilitation de la compréhension du besoin par la modélisation haut niveau.

En ce qui concerne les aspects outils, on ne doit pas rechercher à qualifier HiLeS Designer (au sens DO178B). Une démonstration que les sorties de l'outil sont évaluées indépendamment est possible mais reste à faire.

Avec ces perspectives, les étapes à continuer sont, entre autres :

- Lien avec des langages de modélisation haut niveau (AADL…).
- Lien avec des langages de génération de testbench (PSL…).
- Lien avec la modélisation du « firmware » (logiciels de bas niveau).
- Analyse des paramètres industriels.
- Formalisation de la méthode de conception supportée par HILES.

3.7 Les insuffisances de la démarche en l'état

Pour l'utilisateur averti, HiLeS Designer est un outil efficace pour construire une représentation graphique « amont » d'un système. Il faut s'interroger, à l'usage, si la représentation du composant élémentaire est suffisamment riche pour illustrer tous les cas possibles de comportement et de fonctionnement. Cependant, la représentation HiLeS est complexe, peu lisible et rigide pour le concepteur qui souhaiterait voir l'état d'avancement de son travail sous différents angles : représentation architecturale, représentation métier ou bien, simplement le réseau de Pretri décrivant la commande associée au système. Cet aspect devra être développé dans l'avenir en conformité avec les recommandations de SysML™.

La relation au logiciel n'a pas été traitée au fond. Les exemples du chapitre quatre nous aideront à formuler des recommandations dans ce sens.

Notre approche est conçue comme le cœur d'une plate-forme de conception, dans ce contexte d'autres insuffisances sont apparues. Ci-dessous, le récapitulatif des déficiences que nous avons pu constater et qui certainement vont représenter des points à développer ou à améliorer dans des travaux ultérieurs.

- **Capture de spécifications :** Malgré nos recommandations de « bonne pratique » il manque encore une procédure formelle de capture des spécifications. Pour l'instant, nous nous sommes fortement appuyés sur le savoir-faire du concepteur pour identifier les états principaux des systèmes et pour construire son environnement de fonctionnement à partir du cahier des charges. Eventuellement l'utilisation des cas d'utilisation et des diagrammes de séquence UML pourra définir un outillage complémentaire facilitant l'entrée aux représentations HiLeS. Cette aide à la capture devrait être complétée par des critères de classification des spécifications permettant d'établir des rapports hiérarchiques dès le départ de la modélisation.
- **La traçabilité :** Pour pouvoir répondre aux besoins de certification, il est nécessaire que soit définie une structure de données permettant de suivre la prise en compte et la déclinaison des besoins dans un projet HiLeS. Pour cela, nous devons pouvoir développer encore plus le concept des CDF proposés précédemment (§2.9.1.7). Nous pensons que les CDF vont servir de base pour nourrir un **système de documentation partagée** du projet permettant de suivre la prise en compte pas à pas des besoins dans la démarche descendante. Cela s'inscrit dans une démarche d'écriture de fiches documentaires compatibles avec le standard XML, pour pouvoir les partager avec d'autres outils.

- **Lisibilité :** La sémantique de HiLeS devient très vite compliquée. En outre, nous n'avons que la possibilité de « voir » un niveau de représentation par bloc à la fois. L'utilisateur peut facilement naviguer dans les projets HiLeS en utilisant la fonction « go inside » et l'explorateur de projets (§2.9.1.1). Mais la disponibilité d'un aperçu général du projet est tout à fait urgente. Nous pouvons envisager par la suite des « points de vue » complémentaires permettant de donner à l'utilisateur plusieurs aperçus simplifiés du projet :
  - Une première option est l'élaboration d'un arbre simplifié du projet. (A terme il faudra prendre en compte les conclusions des initiatives comme SysML pour élaborer des représentations compatibles et standardisées).
  - Une deuxième option, pour faciliter la visualisation des séquences fondamentales du système ainsi que son comportement de base, est une représentation « sans blocs ». De cette façon le concepteur aura une visualisation construite uniquement par des réseaux de Petri, simplification qui peut avoir lieu niveau par niveau.

- **Simulation pas à pas avec TINA**. Ce sera souhaitable de pouvoir exécuter des simulations pas à pas sur TINA permettant de suivre état par état les évolutions du système. Pour l'instant nous n'avons pas le mécanisme de dialogue entre HiLeS Designer et TINA différents du transfert de fichiers décrit dans §3.4 qui reste à développer.

- **Génération systématique et automatique de code VHDL-AMS pour des outils commerciaux.** La génération du code 100% utilisable dépend de l'outil de simulation ciblé. Ceci implique le besoin de définir des « Templates » pour que le code soit généré en adéquation aux particularités de chaque outil. Pour l'instant nous avons implémenté une première version compatible avec SystemVision® de chez Mentor Graphics afin de pouvoir tester notre proposition. Un aspect très important, non seulement pour la génération automatique de code, mais aussi pour la structure générale des modèles HiLeS en VHDL-AMS peut être la définition de règles ou de méthodes d'écriture formelles. Parmi les travaux actuels, la méthode a-FSM inventée par Y Hervé [Her03] semble pouvoir se rapprocher conceptuellement de notre démarche. Le principe des a-FSM est la définition

des états fonctionnels d'un système et l'identifications des conditions de transition entre les états. Ces éléments constituent la base du modèle VHDL-AMS : les équations du modèle sont écrites pour chaque état. Le rapprochement avec notre démarche est clair car notre proposition à base de réseaux de Petri et blocs peut être interprétée en termes des a-FSM.

- **Vérification formelle du code VHDL-AMS.** Plus important que la génération de code pour un outil spécifique, la qualité des modèles produits est cruciale. Vu que la tâche d'écriture des fonctions élémentaires (description interne des blocs fonctionnels) est réservée au concepteur, le code VHDL-AMS est de son entière responsabilité. HiLeS Designer ne comporte pas de fonctionnalités tournées vers la vérification du code, ni au niveau syntaxique ni au niveau de la structure du code.

3.8 Une nouvelle phase de développement : HiLeS 1

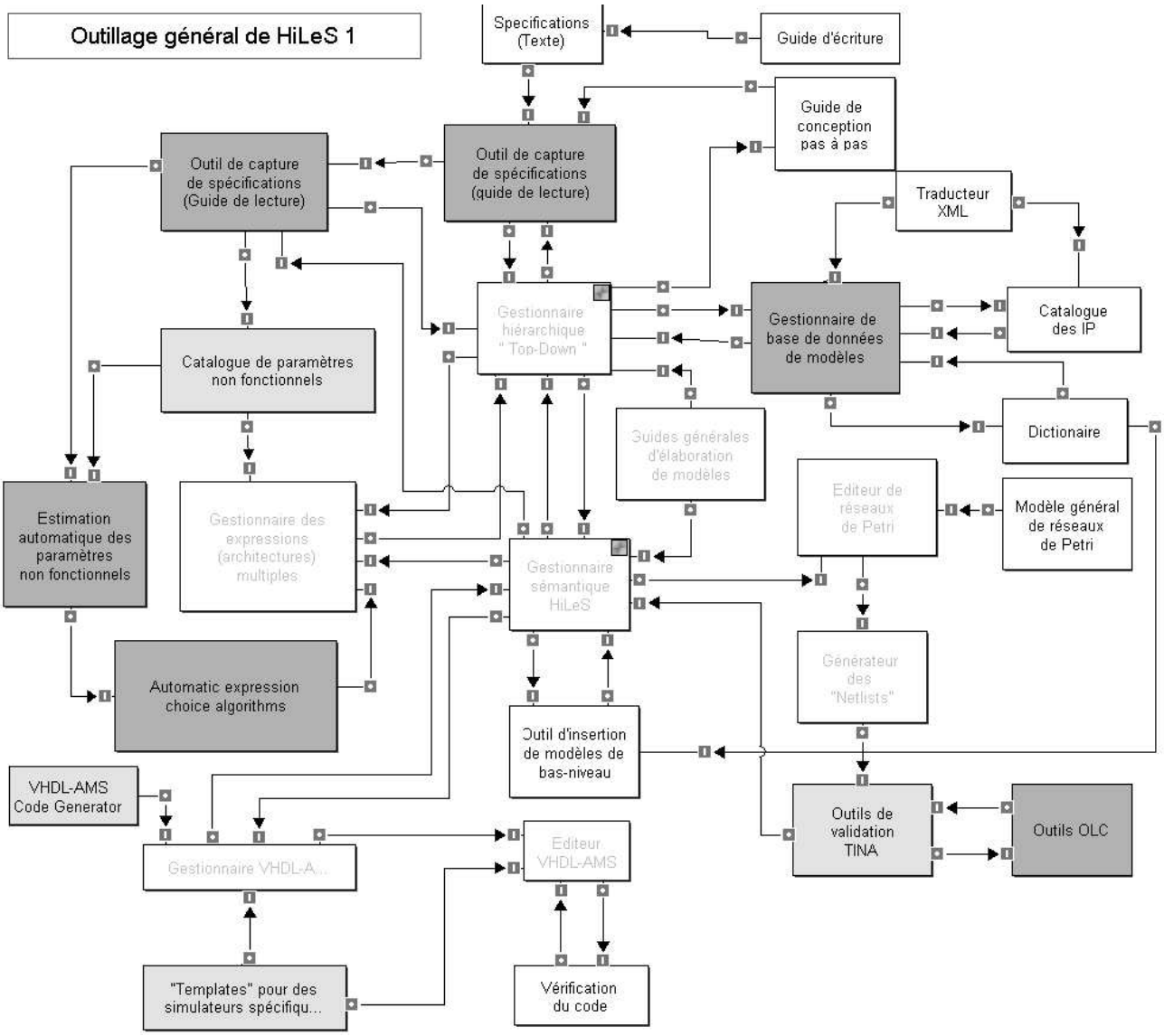


Figure 3-22. Outillage général d'une nouvelle génération HiLeS

Suite à cette première phase de développement et d'intégration d'outils nous constatons, en considérant les insuffisances ici mentionnées, la nécessité de lancer un nouveau programme, peut être, sous le nom de HiLeS 1. Ce projet devra tenir compte des conclusions de nos travaux de thèse ainsi que des innovations les plus récentes en matière de conception système qui ont été développées en parallèle. Nous réfléchissons notamment aux approches générales de type UML-SysML et aux outils d'écriture de spécifications, ainsi qu'aux mécanismes d'exploitation de la modélisation par des réseaux de Petri combinée avec le langage VHDL-AMS.

Dans la Figure 3-22, nous présentons l'ensemble des constituants que nous envisageons d'intégrer pour cette nouvelle phase. Il faut mentionner qu'ici, nous avons pris comme point de départ les fonctionnalités actuellement implémentées sur HiLeS Designer 0 (**Erreur ! Source du renvoi introuvable.**), ces modules sont ici présentés avec leurs noms en gris. Les blocs jeunes indiquent des modules en cours de développement dans le cadre du projet transversal MOCAS. Les modules bleu cyan font partie de nos réflexions courantes avec nos collègues du LAAS et du LESIA. Les blocs blancs avec des titres noirs pourront être inclus dans les travaux correspondants à des collaborations qui commencent à se designer avec nos les équipes de conception de Airbus, notamment dans la mise en place d'un pôle régional de compétitivité. Pour pouvoir lancer ce nouveau programme, nous devons continuer l'exploitation de l'outil actuel afin de rassembler la plus grande quantité possible de retours d'expérience sur la démarche de conception en général ainsi que sur l'ergonomie de l'outil. Ces données seront de grande utilité au moment d'engager la spécification d'une nouvelle génération d'outils.

Finalement, et pour synthétiser les idées exposées, dans le contexte d'un effort coopératif pour le développement d'une nouvelle génération HiLeS 1, nous avons identifié les 10 points clés d'intérêt suivants pour la recherche et l'industrie :

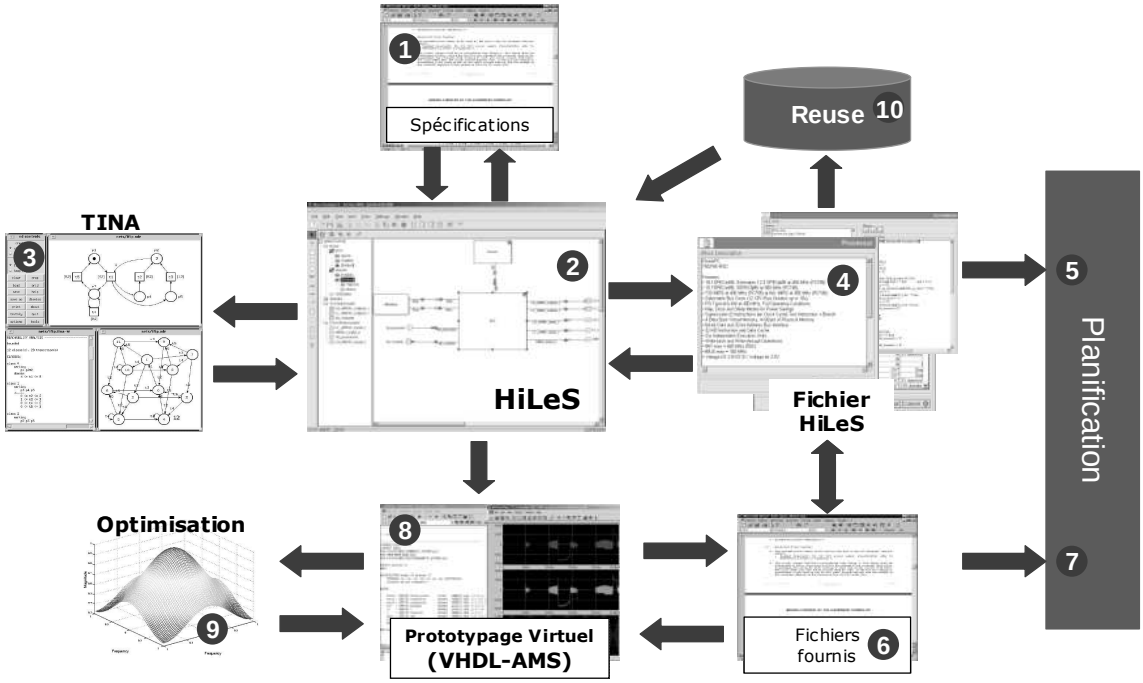


Figure 3-23. Une démarche de conception complète.

1. Guide de rédaction des spécifications,
2. Coupler les acquis à la conception,

3. Vérifier la conformité de la conception avec les spécifications,
4. Générer et spécifier des options de tâches de réalisation,
5. Sélectionner les meilleurs scénarios,
6. Exploiter les réponses des fournisseurs,
7. Sélectionner le scénario,
8. Continuer le prototypage virtuel des scénarios retenus,
9. Appliquer des outils d'optimisation,
10. Gérer la réutilisation.

La Figure 3-23 illustre les différents éléments identifies dans notre proposition d'une démarche générale de conception.

3.9 Conclusion

Nous avons rappelé tout d'abord ce qu'était une plate-forme en nous appuyant sur des secteurs où la CAO est particulièrement développée et en analysant les besoins nouveaux de la conception système, notamment, l'hétérogénéité, la pluridisciplinarité et la modélisation sous un modèle formel.

Nous avons, dans cette vision générale, présenté notre ambition pour la mise en place d'une plate-forme autour de l'outil HiLeS Designer comportant :

- La transition des spécifications vers un modèle formel
- La vérification des structures des modèles en utilisant TINA
- La traduction du modèle en VHDL-AMS

Pour l'interprétation et capture des spécifications nous avons fait des recommandations sans proposer d'outil. Pour l'instant, nous nous appuyons sur les connaissances et savoir-faire du concepteur, mais nous avons identifie des points communs entre nos recommandations et le langage UML, notamment, nous avons identifié que **pour une nouvelle génération de plate-forme nous pourrons interfacer HiLeS avec SysML**.

Pour la vérification des réseaux de Petri nous avons mis en place un système de dialogue entre les outils HiLeS Designer et TINA. Pour ce faire nous avons développé une méthode d'exploration du projet HiLeS permettant d'identifier automatiquement les places, les arcs et transitions à tous les niveaux du projet pour générer une netlist équivalente du réseau. Cette netlist est une représentation textuelle écrite selon le format défini par TINA. Nous exécutons ensuite TINA à partir d'une ligne de commande en indiquant le fichier à analyser. De cette façon nous n'utilisons pas l'interface graphique de TINA et restons dans l'environnement de travail HiLeS Designer. Les résultats de la vérification sont présentés sur l'écran. Nous avons implémenté deux types d'analyse : une analyse de base des propriétés des réseaux identique à celle proposé par TINA et une analyse par observateurs de propriétés. Avec cette dernière, nous avons l'objectif de faciliter l'utilisation des réseaux de Petri en tant que formalisme pour la conception système. Nous essayons de les rapprocher des questions métier que peuvent avoir les concepteurs sur certains points particuliers de leurs modèles.

Pour la partie VHDL-AMS, nous avons construit la structure de HiLeS Designer compatible avec le langage, c'est-à-dire, que les éléments de l'outil ont tous une traduction possible en VHDL-AMS. Pour la partie réseaux de Petri, nous utilisons une approche par composants. Les transitions et les places sont des composants type que nous assemblons selon la topologie du réseau. Ceci nous a permis de systématiser la génération automatique du code. Par rapport aux blocs HiLeS, ils sont

traduits comme des entités, et leur description interne comme leurs architectures. HiLeS Designer comporte aussi des facilités opérationnelles pour écrire les fonctions des blocs. C'est le cas de l'éditeur VHDL-AMS qui capable d'identifier les mots clefs du langage. Au départ de notre projet, nous avions l'intention de rester indépendants des outils de simulation VHDL-AMS, mais pour pouvoir tester notre approche, nous avons choisi de générer le code pour être simulé sous SystemVision de chez Mentor Graphics.

Nous avons, exploré les perspectives de la méthode en terme de :

- Evaluation de performance,
- Validation, vérification, certification.

Différentes possibilités ont été identifiées qu'il convient d'implémenter et de valider pas à pas.

CHAPITRE 4

4. ILLUSTRATION SUR DEUX EXEMPLES D'APPLICATION

4.1 Introduction

Dans ce chapitre, nous voulons illustrer le potentiel d'application de notre approche sur deux exemples traités pendant notre recherche. Ces études de cas ont été réalisées grâce à la collaboration de la société Airbus France qui nous a donné accès aux informations nécessaires comme les spécifications et le cahier de charges des systèmes avioniques réels. Nous avons bénéficié aussi d'un contact permanent avec les experts des différentes équipes de conception de l'Entreprise.

Nous présentons d'abord un premier exemple qui a servi de véhicule de test dans les premières étapes du développement de l'outil HiLeS Designer. Cette première étude nous a aussi permis de préciser certains objectifs de notre démarche générale de conception système. Pendant l'exécution de cet exemple, nous avons travaillé avec nos collègues David GUIHAL et Fernando GARCIA DE LOS RIOS, qui ont réalisé leur stage d'Ecole d'Ingénieurs et de DEA respectivement, au sein de l'équipe Airbus *Méthodes et Outils du Service Ingénierie Electronique*, en collaboration avec le LAAS-CNRS. Au niveau modélisation, l'objectif de ce projet a été la représentation d'une partie du calculateur ECP (ECAM Control Panel), notamment la modélisation de sa source interne.

La seconde partie de ce chapitre sera consacrée à la présentation d'un « projet pilote » sur lequel nous avons appliqué complètement notre approche de conception descendante. Nous y avons utilisé une version de HiLeS Designer dans laquelle les conclusions tirées du premier projet ont été considérées et des améliorations complémentaires ont été réalisées. Le choix de ce projet a été décidé d'un commun accord avec les équipes de conception Airbus afin de répondre à des besoins réels de la conception système aéronautique. Ce projet nous a permis aussi d'améliorer notre outil et de parvenir à une version opérationnelle, non seulement de l'interface graphique HiLeS Designer, mais aussi d'une première tentative de plate-forme de prototypage virtuel telle que nous l'avons présente dans notre chapitre 3. Ces améliorations ont été réalisées grâce aux travaux qui nous avons conduits avec nos collègues Hernan DUARTE [Dua04] et Julien BAYLAC [BHE04]. Ce projet a représenté pour nous, l'occasion d'appliquer notre démarche en essayant de nous détacher d'une approche « trop architecturale » et en privilégiant une solution générale précédente à des choix

d'implémentation. En outre, nous avons tente de tirer parti de ces exemples pour enrichir notre réflexion sur la problématique générale de la conception système.

4.2 L'exemple du calculateur ECP, « ECAM Control Panel »

Cette première étude de cas nous a été fournie par l'équipe « *Méthodes et Outils du Service Ingénierie Electronique »* d'Airbus. Il s'agit d'un calculateur ECP (ECAM Control Panel) dont une solution architecturale avait déjà été retenue par les concepteurs d'un des plus récents avions commerciaux. Nous avons développé cette étude en considérant l'architecture initiale et sa représentation sous la forme proposée par notre approche HiLeS.

Malgré le fait de ne pas utiliser complètement la démarche de conception, cet exemple nous a permis d'évaluer les fonctionnalités d'une première version de l'interface graphique HiLeS Designer, ainsi que de concevoir des améliorations sur l'outil. Pour ce faire, nous avons utilisé les spécifications originales du système [Air01] ainsi q'une description de la solution retenue [Rai01] afin d'identifier les différentes fonctions fondamentales. A partir de ces spécifications et de l'architecture finale, nous sommes arrivés à une première représentation graphique de ces fonctions et de leurs interactions. Nous avons ensuite élaboré un prototype virtuel. Voici donc, une rapide description générale du système :

Actuellement, les avions sont équipés de systèmes d'alarmes complexes qui permettent aux équipages de surveiller le comportement de tous les systèmes de bord et de détecter les défaillances et les situations anormales. Dans de tels systèmes d'alarmes de vol (Flight Warning Systems ou FWS), les fonctions d'interface homme-machine (HMI) sont centralisées. Un panel de contrôle permet de manipuler les écrans du système, de reconnaître les messages d'alarme, et de chercher des informations particulières sur l'appareil. Ce panneau est appelé « Electronic Centralized Aircraft Monitoring » (ECAM) Control Panel. A partir d'ici, nous nous référerons à ce système sous l'acronyme ECP.

4.2.1 Description spécifique de l'ECP

Le calculateur ECP est un sous-système du "Flight Warning System" (FWS) qui fournit à l'équipage des fonctions de IHM pour agir sur le FWS, avec une sortie visuelle vers le "Control and Display System" (CDS). Le FWS est un élément avionique centralisé destiné à réaliser :

- la supervision continue des systèmes essentiels de l'avion :
  - l'affichage automatique ou manuel des messages et pages de données du système pour attirer l'attention des pilotes dans une configuration normale de vol,
  - la génération immédiate des alertes : Dans le cas d'une défaillance des systèmes de l'avion ou dans le cas de configurations dangereuses de vol,
  - la génération d'alertes retardées (Alertes type 4 [Air01])
- l'affichage des listes de vérification normales et des informations complémentaires correspondant a la séquence de vol ou venant en réponse aux demandes de l'équipage.
- L'émission d'alertes sonores et visuelles :
  - sonores, par l'émission de sons et de voix synthétisées à travers le système de haut-parleurs de cabine,
  - visuels, par LEDS clignotants et avec messages sur les écrans du système CDS.

4.2.2 Inventaire des fonctions

Pour exécuter les actions précédemment indiquées, le calculateur ECP comporte les fonctionnalités suivantes :

- entrée des données : sur demande de l'équipage de l'avion, typiquement en utilisant un clavier.
- redondance : certaines entrées sont câblées directement aux calculateurs FWC1 et FWC2.
- retour d'ordres : le pilote doit être averti sur la bonne réception des commandes par le calculateur. La méthode typiquement appliquée couramment est celle des touches de clavier retro éclairés.
- communications : le calculateur dispose des moyens de communication capables de gérer le protocole ARINC429 [CE00] .
- exécution des commandes : le calculateur doit répondre aux demandes de l'équipage et permettre la « navigation » sur les écrans du système.
- auto vérification : un système de diagnostic automatique doit être intégré au calculateur afin de pouvoir vérifier la normalité de l'opération, dès l'allumage de l'unité.

4.2.3 Description descendante du système sous HiLeS Designer 0

Sur la base des documents disponibles, nous avons commencé à établir la modélisation. Tel que nous l'avons mentionné précédemment, une première tentative a été réalisée en représentant l'architecture retenue par une équipe de conception avionique. Dans cet exemple [Gui03], HiLeS Designer a été utilisé pour décrire l'architecture générale du système : comprenant une description simplifiée de son environnement local, pour élaborer un modèle détaillé de ses interfaces de communication (développé complètement en VHDL par D GUIHAL et non présenté dans ce travail), pour réaliser un exemple d'exploration architecturale appliqué au système interne d'alimentation d'énergie et une exécution de sa logique de commande dans la routine de mise en route du système. Aucun modèle de logiciel n'est fait à cette étape du projet.

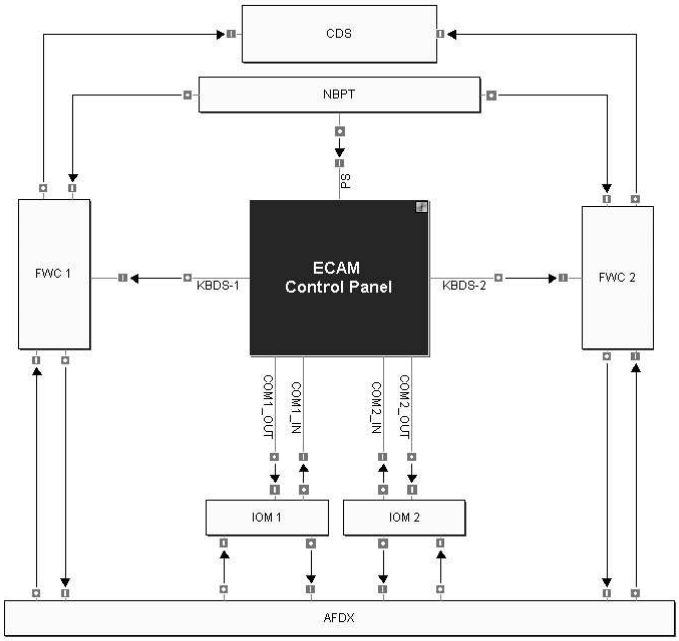


Figure 4-1. Niveau 0, le ECP et dans son environnement.

Nous avons considéré, comme nous le proposons auparavant (§3.5.1.1), ce premier niveau (0) de description, comme l'environnement de travail du calculateur, c'est à dire le FWS. Sur la Figure 4-1 nous présentons ce premier niveau : Chacun des blocs représente un des éléments structurels du FWS, parmi eux le ECP. Les blocs marqués avec des icônes colorés dans le coin supérieur droit contiennent des niveaux inférieurs. Une fois l'environnement du système créé, le panneau de contrôle du bloc ECAM est décomposé et son premier niveau interne est établi; les fonctions principales sont déployées et des raccordements parmi elles sont définis. Ici nous montrons une de ces fonctions : Le sous système de surveillance (Figure 4-2). Selon le cahier de charges, une fonction de surveillance est nécessaire pour vérifier l'opération normale du système, cette fonction est reliée à toutes les autres fonctions et exécute des algorithmes de contrôle à l'intérieur d'eux. Parmi les tâches de surveillance, la séquence de démarrage est exécutée en relation étroite avec le "Power_Block", le système de surveillance détecte les valeurs anormales et produit un rapport de défaillance si nécessaire.

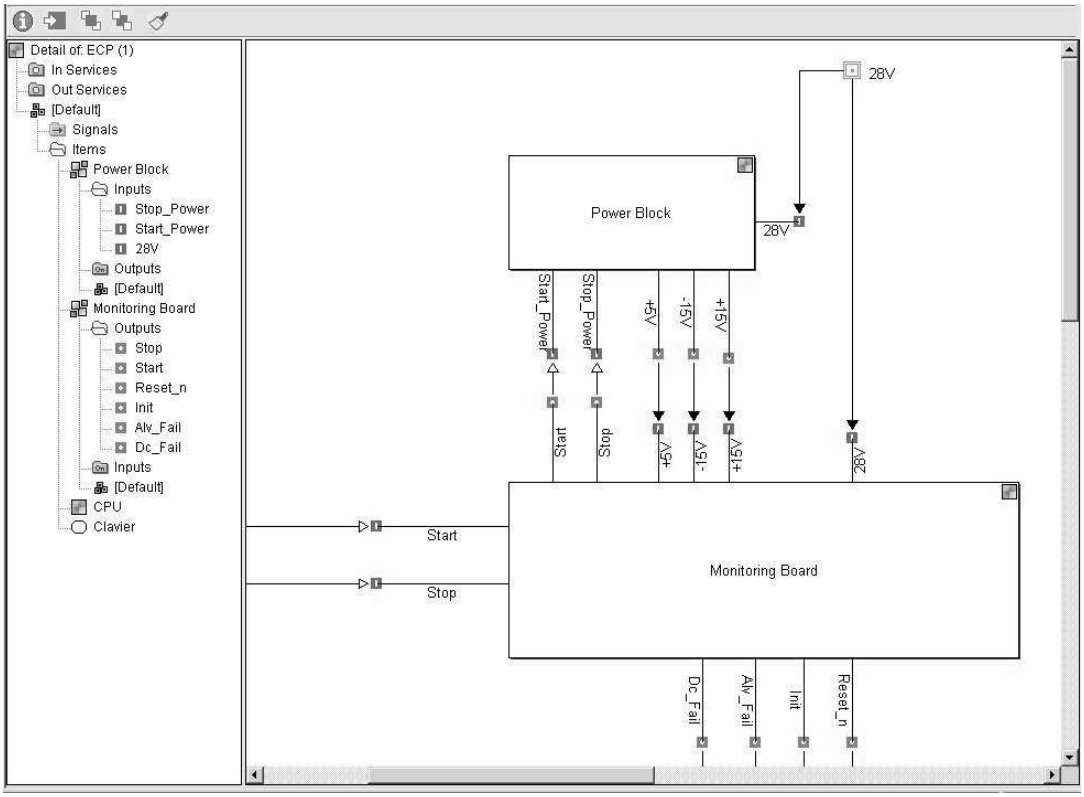


Figure 4-2. Aperçu de la description interne de la fonction de surveillance, leur connections vers l'unité d'alimentation [Gui03]

La définition interne de la fonction surveillance de la séquence de démarrage de l'unité d'alimentation est montrée dans la Figure 4-3. Elle se compose d'un réseau de Petri associé à la fonction "Check_Power", cette fonction est intérieurement implémentée en utilisant VHDL-AMS et vérifie les valeurs de sortie de la source d'alimentation. Lorsque la vérification est faite la fonction renvoie un signal d'acquittement.

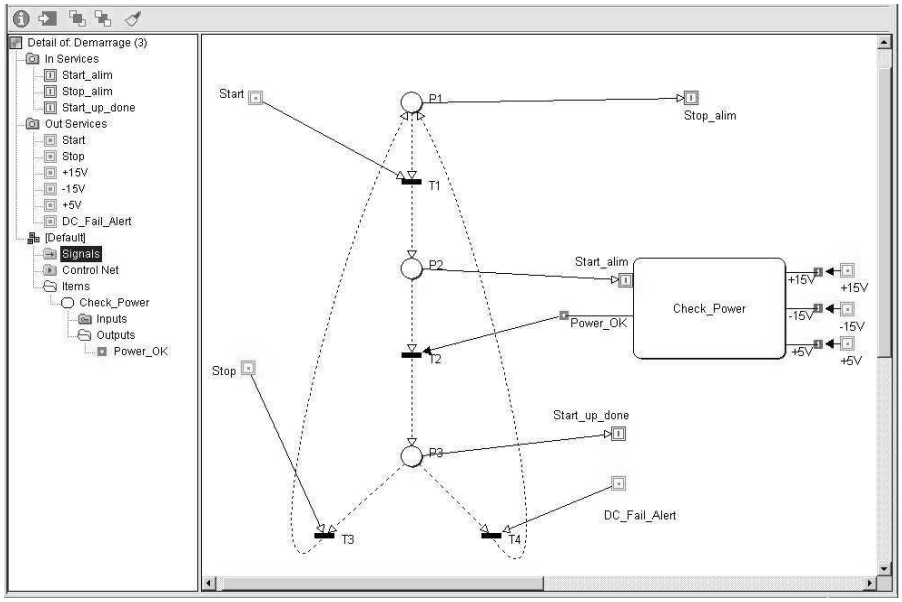


Figure 4-3. Implémentation de la fonction de surveillance de la séquence de démarrage de la source d'alimentation du calculateur ECP.

Pour la fonction d'alimentation d'énergie, nous avons employé le principe des expressions multiples de HiLeS Designer. Le premier modèle était juste une description comportementale des prestations prévues. Une deuxième option a été considérée : un modèle plus détaillé d'une source d'alimentation à commutation de type « step-down » avec les mêmes prestations et la même interface définie pour le modèle simple. Les simulations de la source d'alimentation et de l'interface de communications ont été faites [Gui03] sous Simplorer 6.0. Ici, nous montrons la mise en route de l'alimentation, voir Figure 4-4.

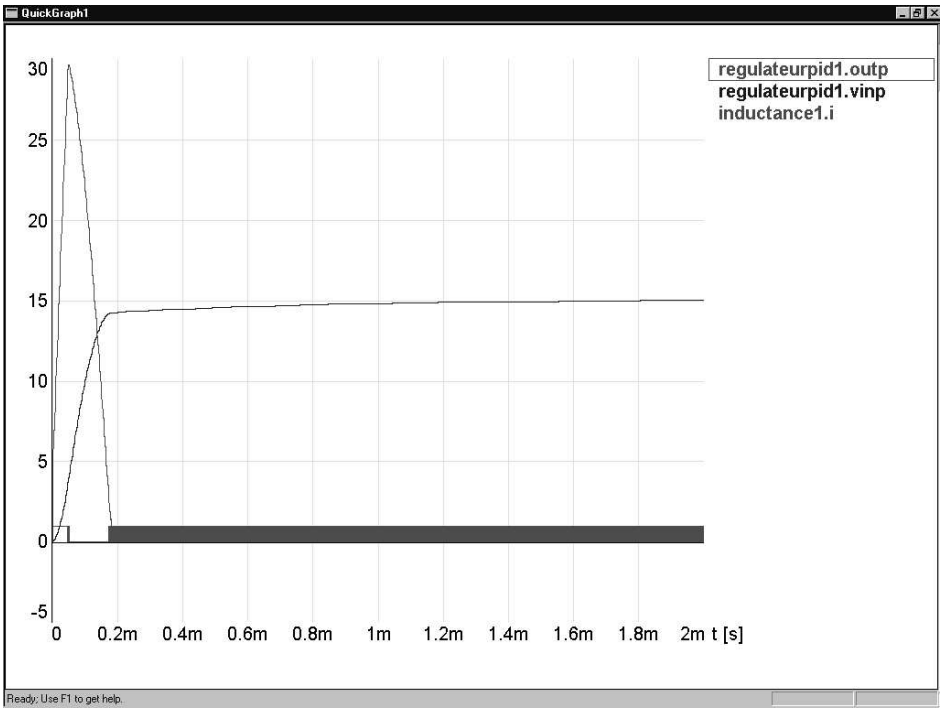


Figure 4-4. Simulation de la source **«** Step Down » [Gui03]

L'exemple du système ECP nous a permis de tester notre première interface de HiLeS - VHDL-AMS, aussi bien que, l'interprétation des réseaux de Petri en utilisant une version révisée [Gui03] de CONPAR (§3.5.2.1). L'équivalence structurale de HiLeS et de VHDL-AMS doit rendre plus facile la construction du modèle, même si le code des fonctions doit être écrit par l'utilisateur et la traduction structurelle a été réalisée ici sans l'aide d'une méthode automatique.

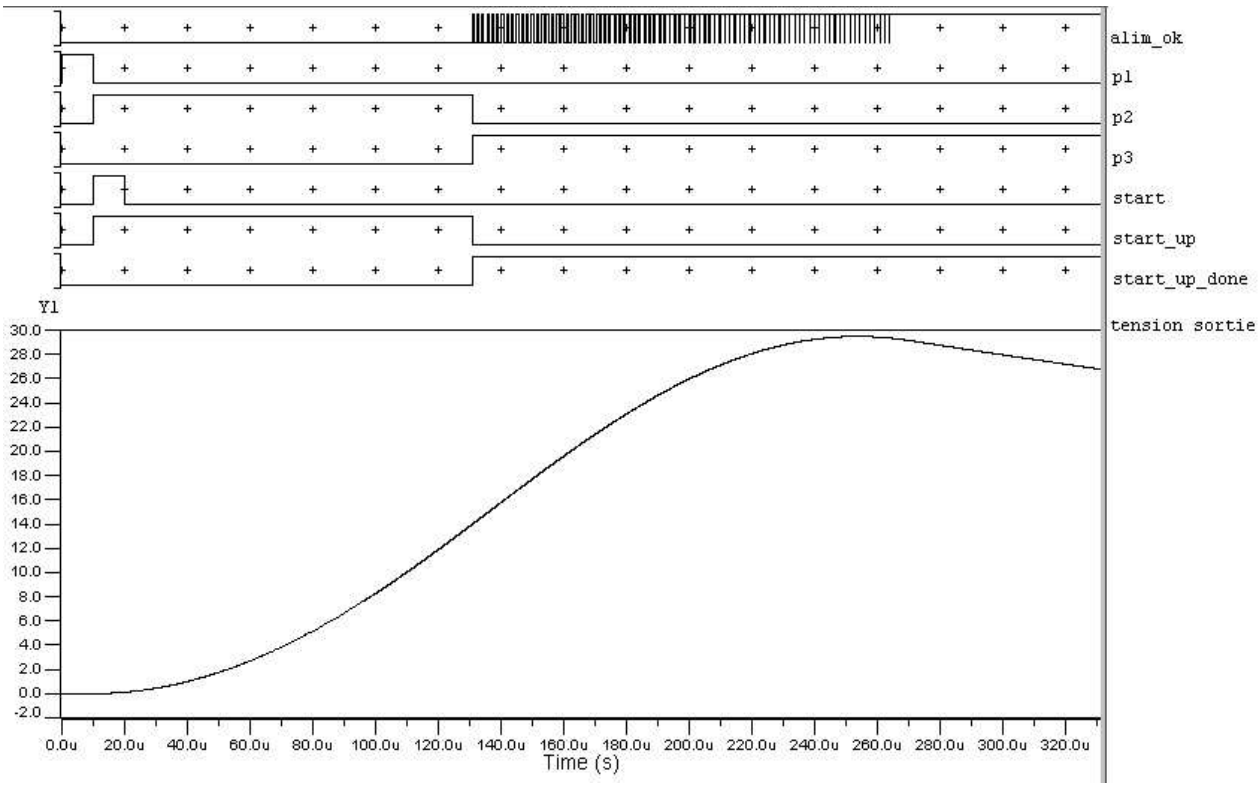


Figure 4-5. Simulation de la surveillance au démarrage de la source d'alimentation.

La Figure 4-5 illustre le comportement du système lors de l'exécution de surveillance de la séquence de démarrage dans la source de tension. Les chronogrammes indiquent le séquencement du sous système de la Figure 4-3. Nous apercevons le marquage des places P1, P2 et P3 et l'initiation du système de vérification de la source (alim_ok). Le système indique la fin du démarrage lors que la tension de sortie atteigne la valeur nominale.

4.2.4 Conclusions de la première étude

Avant de réaliser un bilan sur cette première étude il est tout à fait pertinent de signaler qu'au-delà des résultats des simulations ou des caractéristiques du modèle réalisé, ce qui nous a intéressé, est la partie opérationnelle et ergonomique de la mise en œuvre de la démarche.

Malgré notre intention initiale de rester assez général par rapport aux outils de simulation, nous avons constaté le besoin d'implémenter des mécanismes de traduction automatique des modèles selon l'outil ciblé. Les simulations réalisés pour cet exemple ont été faites sous Simplorer 6, les

traductions des blocs ont été effectuées manuellement. Une partie importante du projet a permis aussi de tester trois outils de simulation VHDL-AMS, par la suite, nous utiliserons SystemVision de Mentor Graphics. Ce choix est justifié dans le cadre d'une collaboration existante entre Mentor Graphics et le LAAS-CNRS.

Un des aspects qui a été clairement sous utilisé dans cette première application est la modélisation à base de réseau de Petri. L'exploitation de cet aspect de notre approche est cruciale et suite à ce projet nous avons mis plus d'efforts sur ce point, notamment sur l'interopérabilité de HiLeS designer et TINA. Au moment de réaliser ce projet la passerelle entre HiLeS Designer et TINA n'était pas automatique, nous étions arrivés à lancer l'outil mais l'extraction du réseau et la configuration des paramètres des analyses n'était pas opérationnelle. Cependant, ce projet nous a permis d'identifier une possible source d'erreur et de manque de cohérence au niveau de la construction des modèles. La version utilisée pour le développement de cet exemple permettait la connexion directe des places aux blocs, par exemple, P2 dans la Figure 4-3. Ceci implique, dans le cas d'une connexion à un bloc structurel, que du point de vue du fonctionnement du réseau de Petri, le jeton qui marque la place soit « transporté » à l'intérieur du bloc en laissant vide la place associée. A la fin des activités du bloc la place était vide et en conséquence la transition en aval ne pouvait être franchie. Ce comportement empêchait la construction hiérarchique des réseaux de Petri et affectait le comportement décrit. **Dans la version actuelle, la connexion des places aux blocs n'est pas autorisée et l'initialisation des événements des blocs est associée aux transitions**. Ce comportement est illustré de manière détaillée dans §3.5.3. Ce fait facilite l'analyse car à ce moment là, les blocs peuvent être considérés comme des « macro-places ».

Par rapport à leur transformation en VHDL-AMS nous avons utilisé la méthode CONPAR. Celle ci nécessite une étape intermédiaire de transformation, nous avons donc, cherché un mécanisme plus direct de traduction. Par la suite nous utiliserons l'approche par composants présentée dans notre chapitre 3 (§3.5.2.2). L'utilisation directe de places et transitions écrites en VHDL permet de reproduire exactement la topologie exprimée avec des réseaux de Petri, notre objectif est d'éliminer de cette façon une éventuelle source d'erreurs supplémentaires dans notre plate-forme.

Ce projet a été très limité par une approche « trop » architecturale. Nous avons constaté une forte tendance à représenter des composants et des architectures. Cette tendance naturelle est dérivée de l'expérience du concepteur, de l'influence de solutions précédentes ou bien de certaines contraintes imposées par le cahier de charges. La démarche HiLeS privilégie une approche par les états fonctionnels qui n'est pas fortement liée à la réalisation finale du système, cette particularité peut constituer une difficulté au moment d'aborder le problème tout au départ de la modélisation. Il est nécessaire d'aborder la conception visant le fonctionnement du système et non les à priori d'architectures que le concepteur tire de sa propre expérience ou des contraintes imposées par le cahier des charges.

Le développement de cette étude de cas, avec la participation de notre collègue David GUIHAL dans le cadre d'une collaboration avec Airbus France nous a permis de tester HiLeS Designer dans un environnement industriel et d'identifier plusieurs améliorations possibles dans la mise en œuvre de notre démarche. Nous abordons maintenant un deuxième exemple vraiment focalisé sur l'application de notre démarche et non sur des aspects opérationnels de notre outil.

4.3 Projet pilote : Le cœur de calcul, une structure d'accueil d'algorithmes pour des applications avioniques

Apres la réalisation de l'exemple précédent et d'autres projets [HSE+03], [HEP03], [MCEB04], [HEP+03], [PHM+04] qui nous ont permis de préciser notre approche et de tester le fonctionnement de l'outil HiLeS Designer 0, nous avons lancé un « projet pilote ». L'objectif de ce projet pilote était de mettre en œuvre tous les principes de la démarche de conception système exposés au long des chapitres 2 et 3. En conséquence, l'exemple choisi ne nous amène pas à la réalisation d'un prototype matériel mais au stade du prototypage virtuel à partir duquel nous pouvons réfléchir à la détection des états fonctionnels critiques du système et à l'exploration de diverses hypothèses sur l'architecture finale du système.

Dans une application réelle comme celle-ci, les concepteurs se retrouvent rarement devant une feuille blanche demandant une implémentation complètement innovatrice. Dans la plupart des cas un certain nombre de choix ou des décisions concernent la mise en œuvre finale du produit ont déjà été établis. Nous devons tenir compte de ces fortes contraintes dans la conduite de notre approche générale.

Il s'agit d'un système numérique avec des possibles implémentations matérielles et logicielles, cet exemple ne comporte pas de composantes analogiques. Nous ne disposons pas des spécifications propres au projet. En conséquence, une première étape du travail a donc consisté à rassembler les informations concernant le système et les besoins exprimés par les utilisateurs pour ensuite formuler le cahier de charges du système. Cette étude a été faite à partir de la documentation existante du système complet auquel le cœur de calcul appartient : Le calculateur de commandes de vol (CCV). Ce CCV est un système destiné à réaliser des fonctions critiques telles que le pilotage automatique, l'exécution des lois de pilotage manuel et les asservissements au vol et au sol.

4.3.1 Description générale du système : ces spécifications

Le cœur de calcul [HEP04] est d'abord, le noyau du calculateur de commandes de vol CCV. Cependant il peut être utilisé dans d'autres applications car il doit être conçu de manière générique comme une « structure d'accueil » pour des algorithmes de plusieurs types destinés à des applications avioniques diverses. Cette structure doit permettre d'exécuter les fonctions décrites par l'algorithme dans un contexte précisé. L'objectif de notre système **est « l'exécution, dans un contexte mono ou multi-applications des logiciels fonctionnels »**. Le système comporte les moyens de communication interagissant avec les autres équipements de l'avion. Pour l'application du CCV, deux configurations sont possibles : soit deux modes de commande, car la sécurité d'opération demande l'existence de deux dispositifs identiques, un pour exécuter les lois primaires de vol, l'autre pour être prêt à prendre le relève en cas de défaillance de l'unité principale. Le choix du mode d'opération se réalise par le biais de « pin programming », la configuration est détectée par l'unité au moment d'exécuter son initialisation. Une fois en route, les deux unités doivent se surveiller simultanément et seront capables de passer au mode CCV_FAULT en cas de défaillance. En cas de reset, les unités échangeront des signaux afin de synchroniser la sortie du fonctionnement du module principal : PRINC.

4.3.1.1 Exigences explicites dans les spécifications

Nous limitons notre étude à la structure d'accueil en isolant ses spécifications de celles des autres sous-systèmes du calculateur CCV. Nous avons pu relever les exigences suivantes à partir de la documentation disponible [Air02] :

- ***Exigences physiques, encombrement* :** La surface totale allouée au cœur numérique est de 340 mm², ce qui représente le 85% du total de la carte CCV. Nous garderons cette information dans le projet en utilisant les CDF (§2.9.1.7). Dans une phase d'agrégation architecturale, cette exigence peut être déclinée et documentée dans chaque composant du projet.

- ***Exigences architecturales* :** Le micro processeur du système doit être conforme à la certification « avionnable » (airworthiness), le Power PC AMTEL MPC755 a été choisi avant de lancer la démarche de conception du système. Celui-ci est un choix imposé lie à des considérations sur la sécurité de l'architecture avion, car le calculateur doit être différent d'autres calculateurs, notamment des calculateurs de commande de vol secondaire. En conséquence ce premier choix imposé est incontournable. L'utilisation de ce microprocesseur impose aussi une forte contrainte architecturale : il ne permet que deux abonnés donc, la gestion des périphériques devienne critique. Afin de privilégier la performance du système, une liaison directe sera réalisée avec la mémoire, les autres périphériques seront branchés sur un bus PCI. Un gestionnaire de bus PCI est nécessaire afin de pouvoir gérer touts les ports USB ainsi que la mémoire. Cette exigence définie une grande partie de l'implémentation finale du système. Notre approche demande de faire abstraction du choix du calculateur, pour cela nous ne considérerons la présence du calculateur pendant cette phase de conception amont. Cependant, c'est une condition incontournable lors de l'agrégation de modules lors de l'élaboration d'une architecture à partir du modèle HiLeS. En revanche, le partage de ressources proposé dans la spécification pourra être modélisé sans proposer une solution d'implémentation.

- ***Exigences de performance :*** L'exigence imposé est d'obtenir 3 fois plus de puissance de calcul d'une carte CPU d'un autre calculateur de référence (CREF). Par définition des spécifications les différentes instructions sont exécutées sur 4 cycles distincts : Le cycle C1 est à 10 ms, C2 à 20 ms, C3 à 40 ms, C4 à 120 ms. La puissance de calcul est estimée en fonction du nombre de planches SCADE par cycle de calcul. Les planches SCADE sont une méthode à base de flots de données, développées par l'Aérospatiale (Aujourd'hui EADS) afin de décrire les systèmes avions dans leur globalité. Les planches SCADE sont devenues un standard interne à Airbus. Elles sont systématiquement utilisées pour la description des logiciels embarqués. Dans une large mesure, cette réussite est due [SDC99] à leur ressemblance avec des techniques de description graphique des circuits électriques, qui sont bien comprises tant par les ingénieurs du bureau d'étude que par les équipementiers. Nous avons ici un **indicateur de performance** que nous allons respecter lors de l'application de notre démarche. Voici (Tableau 4-1) la distribution des tâches par cycle de calcul dans le calculateur de référence (CREF) :

Tableau 4-1. Performance du calculateur de référence (CREF)

	C1	C2	C3	C4
Période.	10 ms	20 ms	40 ms	120 ms
Numéro de planches SCADE	121	46	478	378
% d'utilisation sur 120mS	41.4%	7.9%	41.2%	9.4%
Puissance de calcul intrinsèque*	33%	8.7%	46.7%	10.9%
DGO traitées par cycle	27	12	100	297
DGI traitées par cycle	86	0	143	158
DSI traitées par cycle	63	0	63	0
DSO traites par cycle	10	0	10	0
Sorties de type relais	28	-	-	-
Sorties analogiques	7	-	-	-
Entrées analogiques	50	-	-	-

* Maximum théorique.

- ***Exigences de fonctionnement sous conditions extrêmes* :** Le système doit fonctionner normalement sous agressions neutroniques à très haute altitude. La solution pour garantir l'intégrité des données est la mise en place d'un code et correction d'erreurs EDC. Une partie de la mémoire est destinée à constituer un vecteur des CRCs de chaque position de mémoire. **Pendant l'initialisation du système**, il est nécessaire de parcourir la totalité de la mémoire afin d'attribuer à chaque position du vecteur une valeur de CRC initial.

4.3.1.2 Les fonctions principales du cœur de calcul

Grâce aux informations disponibles, nous avons identifié les fonctionnalités principales du système. Ces fonctions permettent d'établir les procédures opérationnelles qui nous amèneront à élaborer des diagrammes définissant les cas d'utilisation et les séquences principales du système.

Mise sous tension (Démarrage)

- Lecture des configurations définies par « hardware » (Pin programming)
- Exécuter le Boot en mémoire flash : Lecture de la mémoire et initialisation du vecteur des CRCs.
- Transférer le reste du code vers la RAM

Mise hors tension

- Sauvegarder les informations de la RAM dans la mémoire non volatile (NVM),
- Positionner le cœur de calcul en RESET 1mS avant la chute d'alimentation,
- Les sorties numériques sont inhibées en t<10 mS après la confirmation du Reset. Les sorties de type DSO restent ouvertes,
- Les sorties analogiques sont forcées à courant zéro par des moyens matériels.

Reset

- Sauvegarder la cause de reset dans un registre,
- Aucune fonction sur la mémoire,
- Le reset sera exécuté suite à l'activation confirmée du « Pushbutton » plus 20mS,
- Après ON/OFF,
- Synchronisation avec l'autre module CCV.

Téléchargement : Le téléchargement doit être considéré comme un transfert de données entre le bus PCI et la SDRAM (transfert AFDX), puis par un transfert des mêmes données entre la SDRAM et la mémoire flash (écriture des données). Cette fonctionnalité doit respecter les conditions suivantes :

- Le téléchargement ne se réalisera jamais en vol,
- La fonction téléchargement peut être réalisée aussi par le biais du port RS232 pendant la fabrication ou test hors avion du calculateur, ce port n'est pas câblé lors de l'installation du calculateur dans l'avion.

Exécution de la fonction embarquée

Cette activité doit être indépendante du fonctionnement du système. Le système doit permettre d'accueillir toute fonction ou algorithme respectant ses limitations matérielles : Puissance de calcul, mémoire, capacité d 'entrée/sortie, etc. Normalement l'expert chargé de la conception du calculateur ne connaît pas la nature exacte de la fonction embarquée. Il doit la gérer uniquement en fonction de l'indicateur de performance. C'est à dire par rapport aux cycles et à une estimation des ressources (entrées/sorties ou variables à traiter par exemple) impliqués par cycle. Dans notre cas, l'indicateur de performance est donné de manière explicite dans les spécifications : 3 fois la performance du calculateur de référence CREF (Tableau 4-1).

Gestion des interruptions

Prise en charge par un calculateur dédié : Les interruptions sont détectées et mémorisées jusqu'à l'acquittement par le microprocesseur. Cette spécification est directement liée à l'architecture finale du système que nous ne prenons pas en compte dans notre démarche de conception amont.

4.3.2 Entrées et sorties du système

Entrées/sorties : Le cœur de calcul n'est pas relié directement avec les entrées/ sorties du FGCU, sauf un « push button » de Reset installé dans le cockpit et quelques autres signaux considères comme critiques et donc, connectés directement sans passer pour les ports de communication. Toutes les informations qui concernent l'environnement du système arrivent par le biais des trois ports USB ou par une interface AFDX. Un port de service est disponible.

Tableau 4-2. Récapitulatif des entrées et sorties du système

Nom	Type	Description
USB1	I/O Port de communication	Port de qui communique avec le reste des dispositifs avion
USB2	I/O Port de communication	Port de qui communique avec le reste des dispositifs avion
USB3	I/O Port de communication	Port de qui communique avec le reste des dispositifs avion
RS232	I/O Port de communication	Port de serial de service. Disponible uniquement hors avion
DSIN	Entrées discrètes	Signaux discrets entrant directement au système
DSON	Sorties discrètes	Signaux discrets sortant directement au système

Les entrées discrètes

Un total de 73 signaux d'entrée discrets sont reliés au CCV, la plupart d'entre eux son acquis par le biais du port USB. Nous détaillerons ici uniquement ceux qui sont reliés directement au système faisant partie de la définition de son environnement : 4 signaux sont câblés directement de l'autre CCV, 10 signaux correspondent aux pins de configuration de l'unité.

Tableau 4-3. Liste de signaux discrets connectés directement au système.

Signal	Description
DSI1	Signaux indicateur de l'état de l'autre unité CCV
DSI11	«Dialogue entre unités »
DSI12	«Dialogue entre unités »
DSI13	Cockpit Reset
DSI14	Reset (GND)
DSI22	Identification de l'unité (A/B)
DSI23	Identification de position
DSI24	Identification de position
DSI25	Identification de position
DSI26	Pin de programmation de version A/C
DSI27	Pin de programmation de version A/C
DSI28	Pin de programmation de version A/C
DSI29	Pin de programmation de version A/C
DSI30	Pin de programmation de version A/C
DSI31	Pin de programmation de version A/C
DSI70	Coupure supérieure d'alimentation à 5s

Les sorties discrètes

La plupart des sorties discrètes sont pilotées par le cœur de calcul indirectement à travers du port USB. Les signaux directement activés par le système sont :

Tableau 4-4. Liste de signaux discrets qui sortent directement au système

Signal	Description
DSO1	Signaux d'état de l'unité
DSO5	«Dialogue entre unités »
DSO6	«Dialogue entre unités »
DSO40	Coupure d'alim. supérieur à 5s

DSO5 et DSO6 sont utilisés pour « dialoguer » avec d'autres unités CCV. Pendant l'initialisation du matériel ou après un reset les DS05 et DSI11 des deux unités sont synchronisées dans un intervalle inférieur à 1mS.

Communications : Le cœur de calcul est relié avec son environnement par le biais de :

1 interface AFDX, 3 ports USB et un port RS232. Le port RS232 est un port de maintenance, utilisé pour charger le logiciel pendant les étapes de fabrication du calculateur. Ce port n'est pas câblé lors que le calculateur est mis dans l'avion. *Le système accédera aux ports USB chaque 10ms*.

4.3.3 Interprétation des spécifications : fonctionnement du système

Dû à la difficulté d'interprétation des spécifications et plus particulièrement au fait que plusieurs choix architecturaux y sont imposés, nous avons trouvé nécessaire d'analyser les informations disponibles et de les interpréter pour une meilleure compréhension du fonctionnement du système. Pour cela, comme une première approche à des étapes d'extension à notre approche vers le traitement des spécifications, nous avons utilisé deux diagrammes UML : les cas d'utilisation et les diagrammes de séquences. A l'heure actuelle, ces diagrammes sont très utilisés dans le cadre de la

conception logicielle pour commencer la description. Néanmoins, ils ne sont qu'une représentation graphique qui ne conduit pas à un processus de vérification. L'intérêt d'utiliser des diagrammes UML comme un ressource complémentaire à notre démarche descendante a été évoqué précédemment dans ce mémoire (§3.3.2).

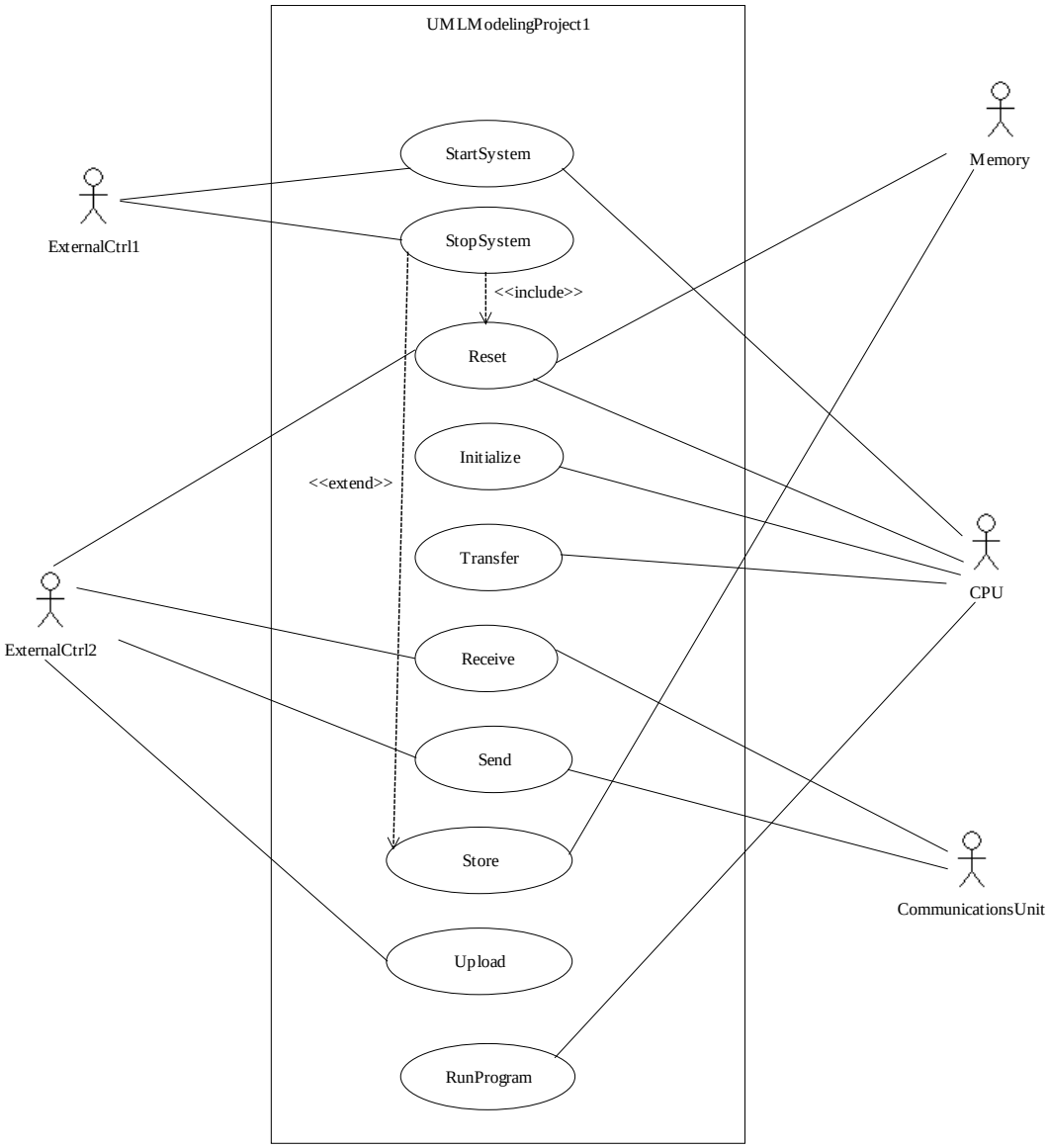


Figure 4-6. Diagramme des cas d'utilisation du système.

Nous avons d'abord construit le diagramme général des cas d'utilisation. Pour ce faire, nous avons identifié les acteurs et les services associées au fonctionnement du système. Ils sont montrés dans le diagramme de la Figure 4-6. Cette première étape a été utile car nous pouvons aussi synthétiser, au niveau des services, les fonctionnalités du système. En revanche, au niveau des acteurs, nous avons été influences par les choix imposés par les spécifications tout en restant assez général. C'est pour cela que nous trouvons explicitement « la CPU », la mémoire et l'unité de communications, dans la représentation. En conclusion, ce diagramme simple et statique du système permet d'identifier les fonctionnalités éventuelles telle que l'initialisation, le reset, le stockage de données et l'exécution du programme d'application parmi d'autres et par la même peut nourrir la création plus formelle de la description HiLeS.

Nous avons besoin aussi d'approcher le comportement dynamique général du système. Pour ce faire, nous avons construit un diagramme de séquence UML permettant d'illustrer les séquencements généraux du système d'une façon très visuelle, simple et riche à la fois.

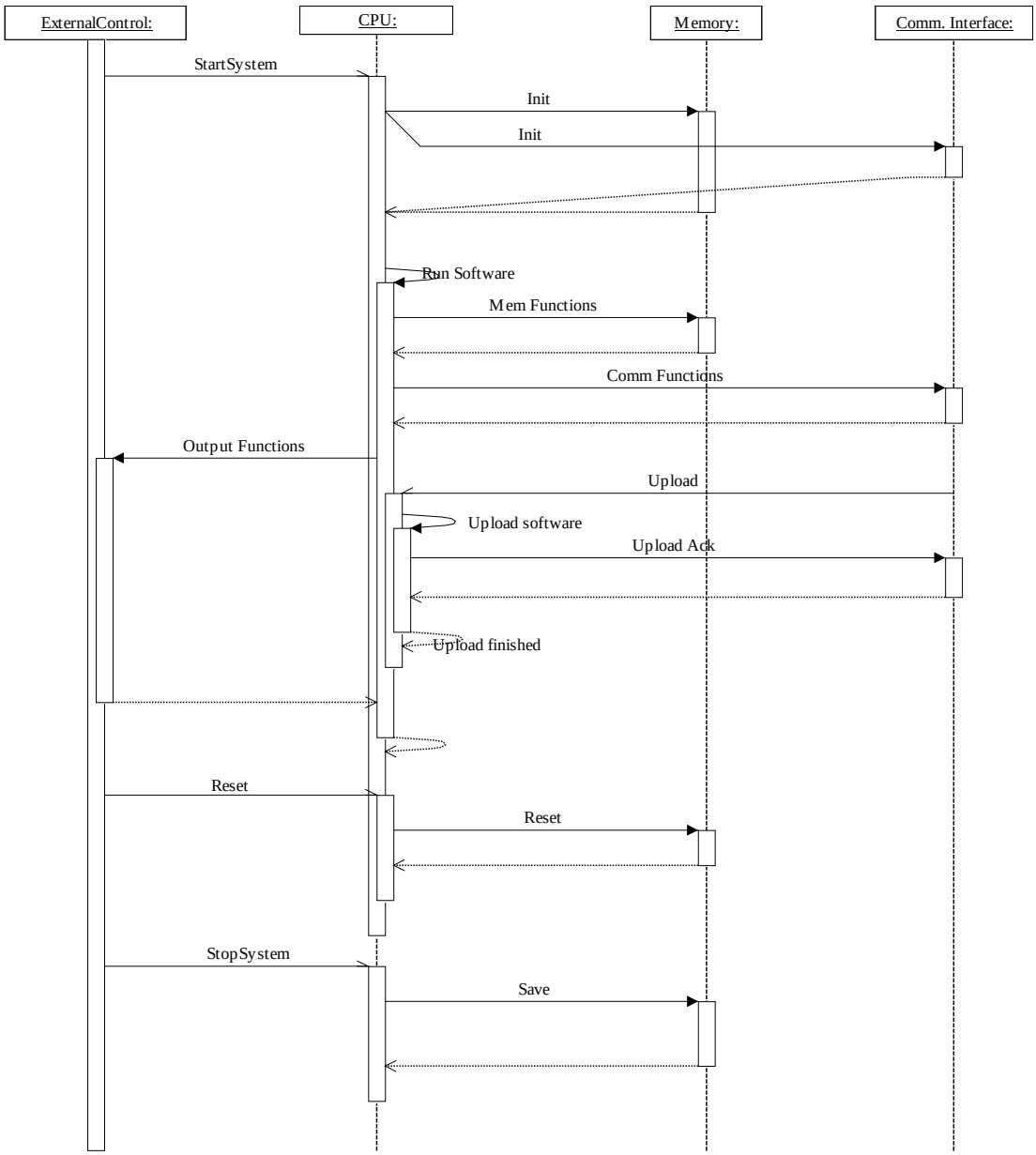


Figure 4-7. Diagramme de séquence du fonctionnement général du coeur de calcul

Le diagramme de séquences décrit les interactions entre les acteurs du système en rajoutant une dimension temporelle : la lecture verticale du diagramme donne une idée de la durée et de la succession des activités ; la lecture horizontale donne les rapports entre les acteurs. L'ensemble nous parait très utile pour identifier certains rapports hiérarchiques et les imbrications des actions du système. De cette représentation, nous pouvons extraire les rapports existants les fonctions de base et plusieurs informations sur le comportement du système. Dans nos études préliminaires, nous avions bien identifié la double nécessité de travailler sur les fonctions et sur les séquences de changement d'états du système pour construire une représentation HiLeS. En faisant l'exercice

d'utiliser les diagrammes UML pour franchir cette étape, nous voulons montrer que l'interconnexion UML2 (SysML™)-HiLeS est très facilement imaginable :

- SysML pourrait être le support à la rédaction d'un guide d'écriture et de lecture des spécifications facilitant la création d'une représentation formelle et vérifiable telle que HiLeS.
- L'étape de conception « amont » pourrait concentrer son action sur la formalisation de la représentation, sa vérification et la préparation des étapes ultérieures sur le partitionnement.

Cependant, les éléments que nous arrivons à dégager par ces diagrammes sont insuffisants. C'est le rôle de l'expert qui va finalement conduire la conception du système et construire les différents niveaux de représentation. C'est la raison pour laquelle, nous nous sommes adressé à un expert en avionique pour réaliser l'application que nous décrivons dans la prochaine section de notre mémoire.

Nous recommandons par la suite, d'explorer plus profondément les possibilités de UML 2 (SysML™) afin de créer des procédures d'interconnexion avec HiLeS et donc faciliter le passage des spécifications vers notre formalisme.

4.3.4 Description du projet sous HiLeS Designer 0

Nous abordons maintenant la modélisation sous HiLeS de la *structure d'accueil* que nous venons de décrire : c'est un cœur de calcul, dispositif principal du calculateur de commande de vol; il peut être configuré pour d'autres applications, puisqu'il permet de télécharger des algorithmes divers. Le système inclut plusieurs de ports de communication lui permettant de se relier et d'interagir avec d'autres dispositifs de l'appareil.

La première étape du processus de conception est l'interprétation et l'analyse des documents disponibles sur l'expression des besoins et les spécifications du système. Tel que nous l'avons mentionné initialement, dans cet exemple, nous n'avons pas disposé d'un document précis de spécifications car le cœur numérique fait partie d'un plus grand système : le calculateur CCV, par conséquent, les exigences du projet étaient incluses dans la documentation [Air02] du système principal. Nous avons extrait les informations nécessaires grâce à la collaboration de M Jean Michel MURAWSKI, un expert en matière de calculateurs avioniques de chez Airbus France. Tel que nous l'avons déjà mentionné, au long de cette première étape, nous avons trouvé, dans les documents de spécification, beaucoup de contraintes qui imposent un nombre significatif de choix architecturaux finals. La plupart des conditions ou des choix prédéfinis sont liées à des questions de conformité avec les normes de sécurité des composants utilisés et à la compatibilité du système avec d'autres équipements de l'avion. Cette remarque montre que, dans des applications réelles, les concepteurs partent rarement d'une feuille blanche pour la réalisation de leurs projets et parfois une partie de l'architecture des systèmes est prédéfinie. Afin d'illustrer notre démarche, nous avons décidé de conserver une vue générale du système et de remettre les questions de choix prédéfinis à la fin de la modélisation de haut-niveau, c'est-à-dire, une fois que la description fonctionnelle sera achevée et que la description architecturale sera entamée.

4.3.4.1 Niveau 0 : Définition de l'environnement du système

Nous commençons notre modélisation avec HileS Designer par la représentation de l'environnement du système. Tel que nous l'avons indiqué dans la section §4.3.2, cinq ports de

communication et une interface discrète définissent cet environnement. La majorité des signaux d'entrée et de sortie sont reliés au système par le biais de ports de communications. Un nombre restreint de signaux considères comme critiques et liés à la configuration ou à la sécurité opérationnelle sont directement reliés au système par l'intermédiaire de l'interface discrète. La Figure 4-8 montre, cette représentation de l'environnement : C'est le niveau 0 de représentation. A ce niveau, la représentation HiLeS reste assez générale.

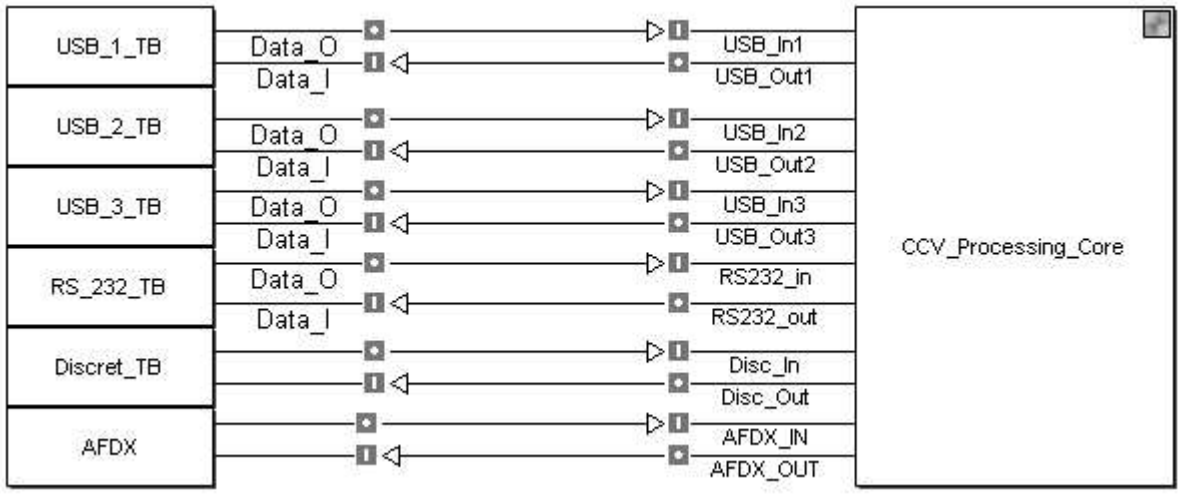


Figure 4-8. Niveau 0 (le système et son environnement) de la représentation HiLeS Designer de la structure d'accueil, cœur de calcul.

Le bloc « CCV_Processing_Core » (Figure 4-8) contient le cœur de calcul du calculateur CCV, il est étiqueté avec une icône carrée colorée au coin supérieur droit, ceci signifie dans la sémantique de HiLeS que le bloc va contenir d'autres niveaux de description à l'intérieur. Les autres blocs de ce premier niveau correspondent aux éléments identifiés à partir des spécifications. Ainsi, nous trouvons les trois ports USB, le port auxiliaire RS232, la rochelle AFDX et les signaux discrets qui entrent directement au système.

4.3.4.2 Niveau –1. Définition des états principaux

Nous avons construit le niveau -1 en utilisant les principaux états fonctionnels du système. Nous avons identifié ces états, d'un part en nous appuyant sur les diagrammes UML réalisés précédemment, et d'autre en tenant compte des avis de l'expert. Dans un premier temps nous avons construit ces diagrammes selon notre seule interprétation des spécifications. Puis, l'expert nous a aidé à classifier les fonctionnalités que nous proposons pour enfin obtenir quatre états fonctionnels :

- **L'exécution des fonctions de base** de la structure d'accueil. Nous avons regroupé sous un seul état fonctionnel l'exécution des fonctions de base du calculateur : initialisation, gestion d'accès aux communications, gestion de mémoire, etc..
- **L'exécution de la fonction embarquée** : Du point de vue fonctionnel ceci est le principal état du système car, théoriquement, il est un état « permanent » qui exécute l'algorithme qui a été téléchargé dans la structure d'accueil.
- **Traitement d'une faute**. Le système comporte un état de faute qui peut être atteint à partir de L'outil d'analyse des pannes,

- **Le téléchargement**. Lorsqu'une demande de téléchargement a été détectée au cours de l'initialisation ou a été requise par un agent extérieur, le système adoptera cet état. Le cahier des charges spécifie que le téléchargement ne se réalisera jamais pendant le vol de l'avion.

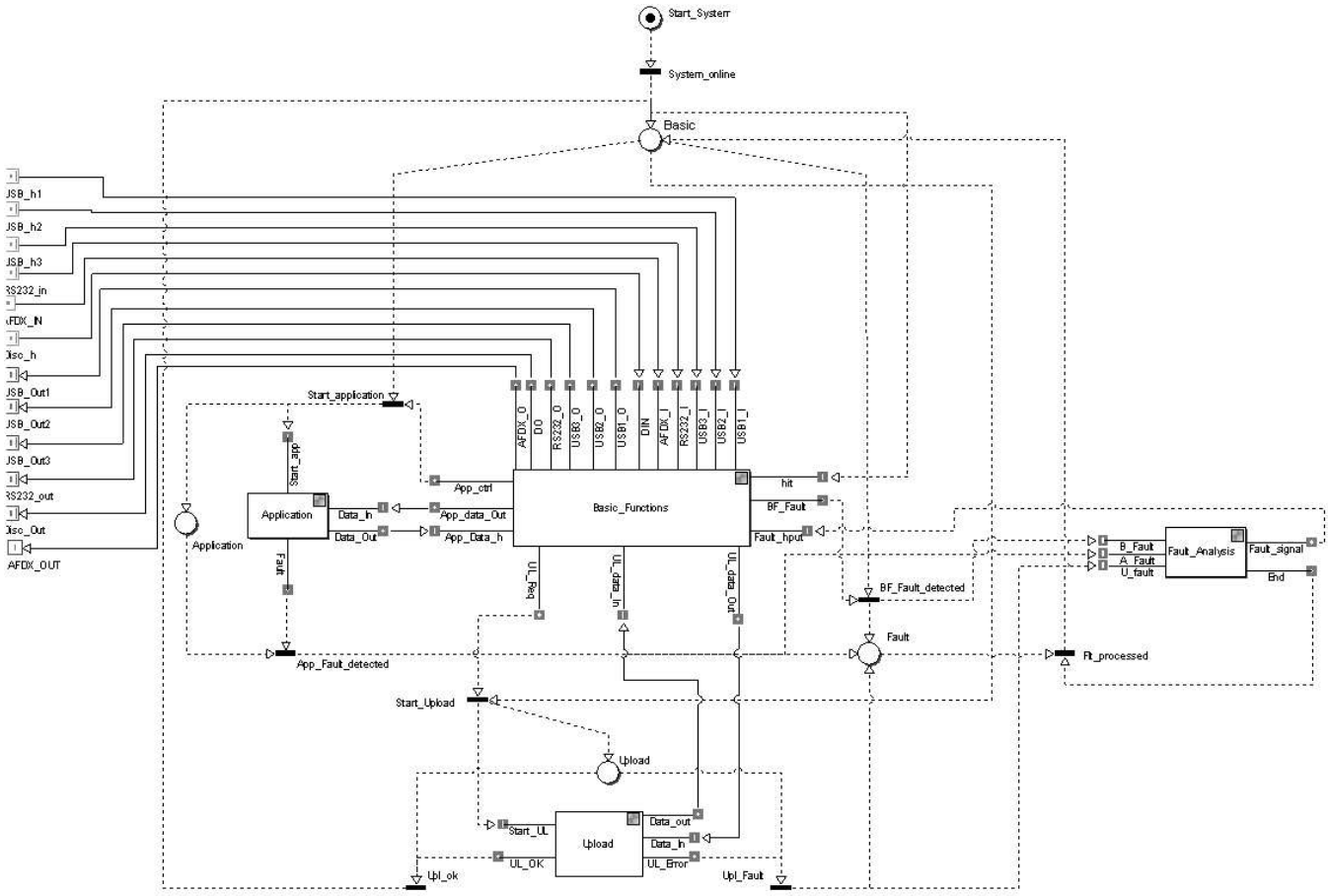


Figure 4-9. Le niveau –1. Représentation des états fondamentaux du système.

Nous avons construit ce premier niveau en prenant en compte les états principaux ici présentés et les interfaces avec l'environnement (Niveau 0). Afin de réaliser cette modélisation nous avons pris comme point de départ le module de base HiLeS présenté en §3.3.1. Chaque état est représenté par une place et les conditions de transition entre les états sont contrôles par les blocs qui les décrivent en détail. Le rôle des blocs, concernant la gestion des transitions, est crucial car c'est grâce à eux que nous arbitrons les conflits de fonctionnement du réseau. Par exemple (Figure 4-9), à partir de *Basic* le système peut aller vers deux états différents : *Fault* ou *Application*. La transition qui sera franchie (*start_application* ou BF_fault_detected) dépend du fonctionnement interne du bloc *Basic_Fonctions*. Nous pouvons, de toute manière, analyser le comportement du réseau en utilisant la procédure d'extraction présentée précédemment (§3.4.1).

Extraction et analyse du réseau de contrôle du niveau –1 : Tout d'abord nous avons généré la netlist correspondant à ce premier niveau : Nous avons fixé les intervalles de transition entre 0 et infini (valeurs par défaut de TINA) mais des études paramétriques peuvent être réalisées en modifiant ces intervalles selon des estimations des temps d'exécution des blocs associés. Nous présentons ci dessous la netlist équivalente de ce niveau :

```
net projet_pilote_CCV
tr Start_application [0,w[ Basic -> Application
tr BF_Fault_detected [0,w[ Basic -> Fault
tr App_Fault_detected [0,w[ Application -> Fault
tr Flt_processed [0,w[ Fault -> Basic
tr Upl_ok [0,w[ Upload -> Basic
tr Upl_Fault [0,w[ Upload -> Fault
tr System_online [0,w[ Start_System -> Basic
tr Start_Upload [0,w[ Basic -> Upload
pl Start_System (1)

REACHABILITY ANALYSIS ----------------------------------------
bounded

5 marking(s), 8 transition(s)

MARKINGS:
0 : Start_System
1 : Basic
2 : Fault
3 : Upload
4 : Application

REACHABILITY GRAPH:
0 -> System_online/1
1 -> BF_Fault_detected/2, Start_Upload/3, Start_application/4
2 -> Flt_processed/1
3 -> Upl_Fault/2, Upl_ok/1
4 -> App_Fault_detected/2
0.000s

LIVENESS ANALYSIS ----------------------------------------------
not live
0 dead marking(s), 4 live marking(s)
0 dead transition(s), 7 live transition(s)

STRONG CONNECTED COMPONENTS:
1 : 0
0 : 1 2 3 4

SCC GRAPH:
1 -> System_online/0
0 -> BF_Fault_detected/0, Start_Upload/0, Start_application/0, Flt_processed/0, Upl_Fault/0, Upl_ok/0,
App_Fault_detected/0
0.000s
```

La simulation par TINA indique que le système possède un nombre fini de jetons et qu'il ne présente pas de blocages. Cependant l'interprétation que nous faisons de cette analyse est insuffisante et demande l'utilisation des observateurs, en cours de développement au moment d'écrire ce mémoire. Une première alternative est une interprétation générale visant la génération du graphe équivalent du système.

4.3.4.3 Niveau –2. Modélisation de l'application embarquée

Nous continuons l'explication des différents niveaux de notre projet avec le bloc *Application.* Le système atteint cette fonctionnalité lorsque l'initialisation (contenue dans *Basic_Functions*) a été réalisée correctement. Lors que la transition en aval, *start_application*, est franchie (Figure 4-9), un jeton marque la place ***application*** et un deuxième « rentre » dans le bloc pour démarrer l'ensemble

des cycles de calcul. La Figure 4-10 montre l'intérieur du bloc application. Nous avons modélisé les quatre cycles de calcul comme des modules identiques comportant, chacun, un réseau simple de commande. L'ensemble de modules est activé suite au franchissement de la transition *Application_started.* Chaque réseau de contrôle cadence les tâches selon les périodes établies dans la spécification : C1=10 ms, C2=20 ms, C3=40 ms et C4=120 ms.

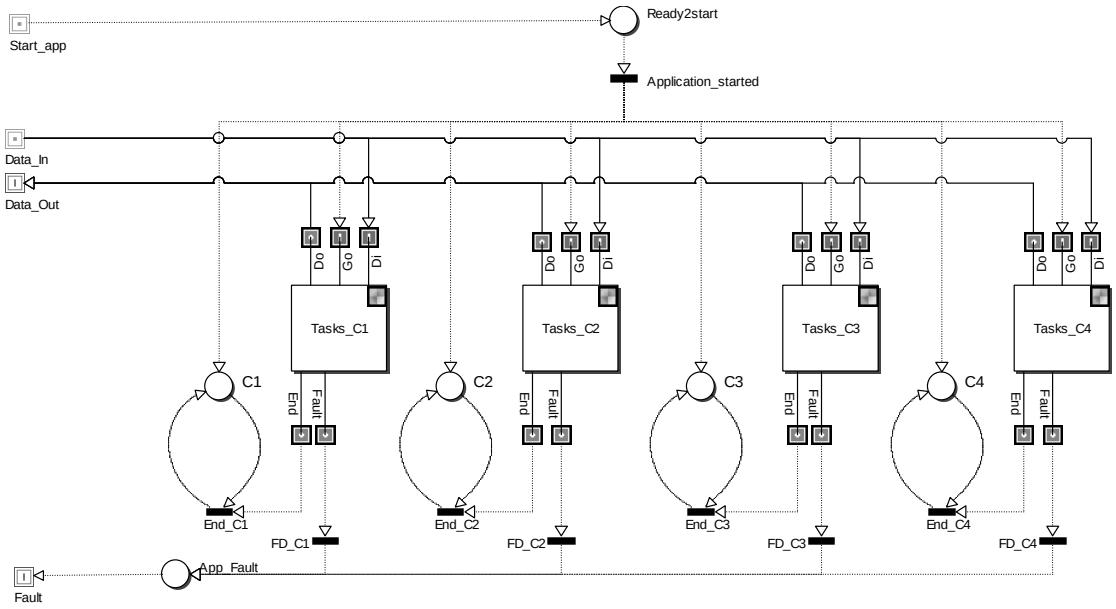


Figure 4-10. Modélisation des cycles de calcul du système.

Les cycles de calcul sont directement en relation avec la mesure de la performance du système. L'indicateur fourni dans les spécifications est le nombre de planches SCADE exécutées par cycle. Tel qu'il est spécifié dans le cahier de charges, chaque cycle comporte une quantité bien définie de planches à exécuter.

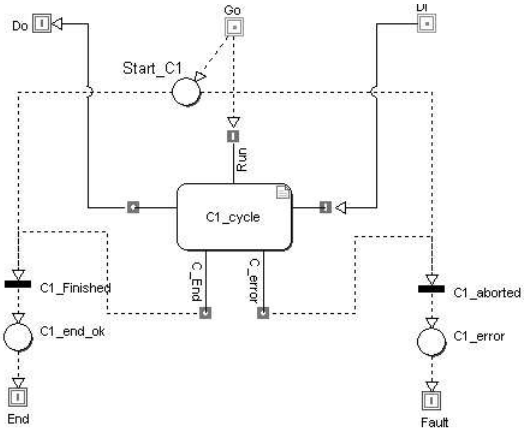


Figure 4-11. Représentation intérieure de chaque bloc *Tasks_CN*.

Nous avons modélisé chaque bloc (Figure 4-11) avec trois possibilités d'opération : Soit le système reste indéfiniment en exécutant les tâches programmées dans l'algorithme accueilli, soit une erreur est détectée et le système sort abruptement de l'exécution, soit l'application se termine. Normalement, une fois démarrées les activités du bloc, celui ci restera dans cet état : l'exécution de sa fonction à chaque « coup de jeton » du réseau de contrôle. Le module comporte aussi une entrée et

une sortie des données nécessaires pour son fonctionnement. A ce niveau, nous ne nous occupons pas des données requises ou émises pendant chaque cycle de calcul car le concepteur de la structure d'accueil (cœur de calcul) ne connaît pas la ou les fonctions qui seront réalisées. Nous pouvons établir une correspondance comportementale en fonction de la demande de ressources moyennes de chaque cycle (Tableau 4-1).

Par rapport aux aspects vérification formelle nous avons réalisé ici une première tentative d'application pratique de l'analyse par observateurs. Nous avons donc placé un observateur sur la transition amont du bloc application, c'est à dire, *« start_application »*. L'objectif est d'analyser le comportement interne du bloc application. Pour ce faire nous avons utilisé une procédure appelée projection par équivalences. Nous ne détaillerons pas les aspects formels de cette analyse qui est traité plus profondément dans [Fau90]. Avec cette procédure nous pouvons obtenir des automates simplifiés du comportement du système comme celui de la Figure 4-12.

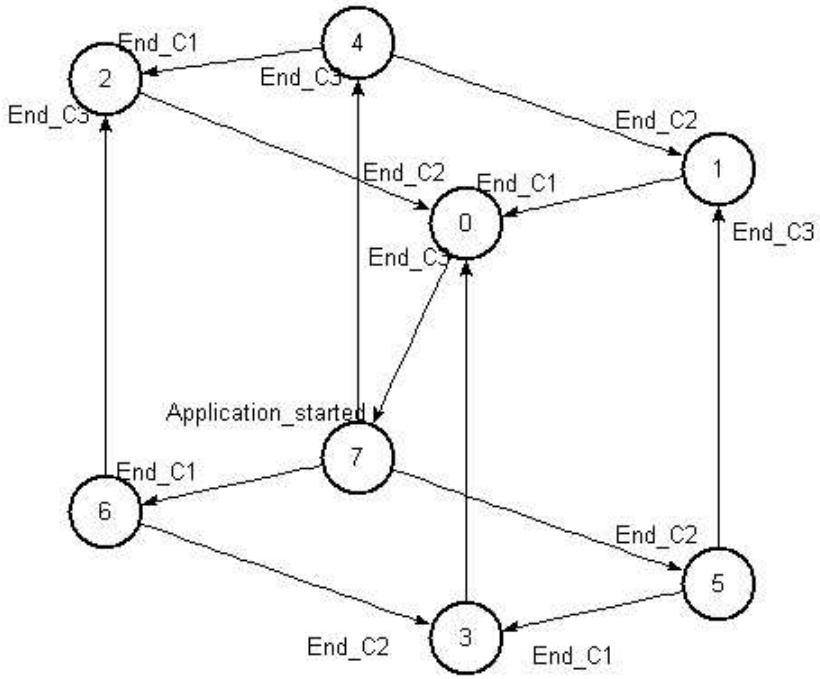


Figure 4-12. Automate équivalent du comportement interne du bloc « Application ».

La projection par équivalences est un mécanisme de vérification qui permet de réaliser des abstractions du comportement représenté afin de le visualiser plus facilement. Dans notre exemple nous constatons q'après le franchissement de la transition qui démarre les différents cycles de l'application, le système a la capacité d'arriver à la fin de chacun de ces cycles. Nous avons considéré ici volontairement uniquement trois cycles au lieu de quatre afin de faciliter la visualisation de l'automate. De point de vue pratique, cet observateur n'est pas encore implémenté dans la plate-forme. Pour obtenir l'automate nous utilisons la netlist généré par HiLeS Designer, utilisons une des options de TINA pour obtenir le graphe complet du système. A partir de ce graphe et en utilisant l'outil ALDEBARAN il est possible d'obtenir une nouvelle netlist qui représente le comportement spécifique d'une partie du système.

4.3.4.4 Niveau –2. Modélisation des fonctions de base

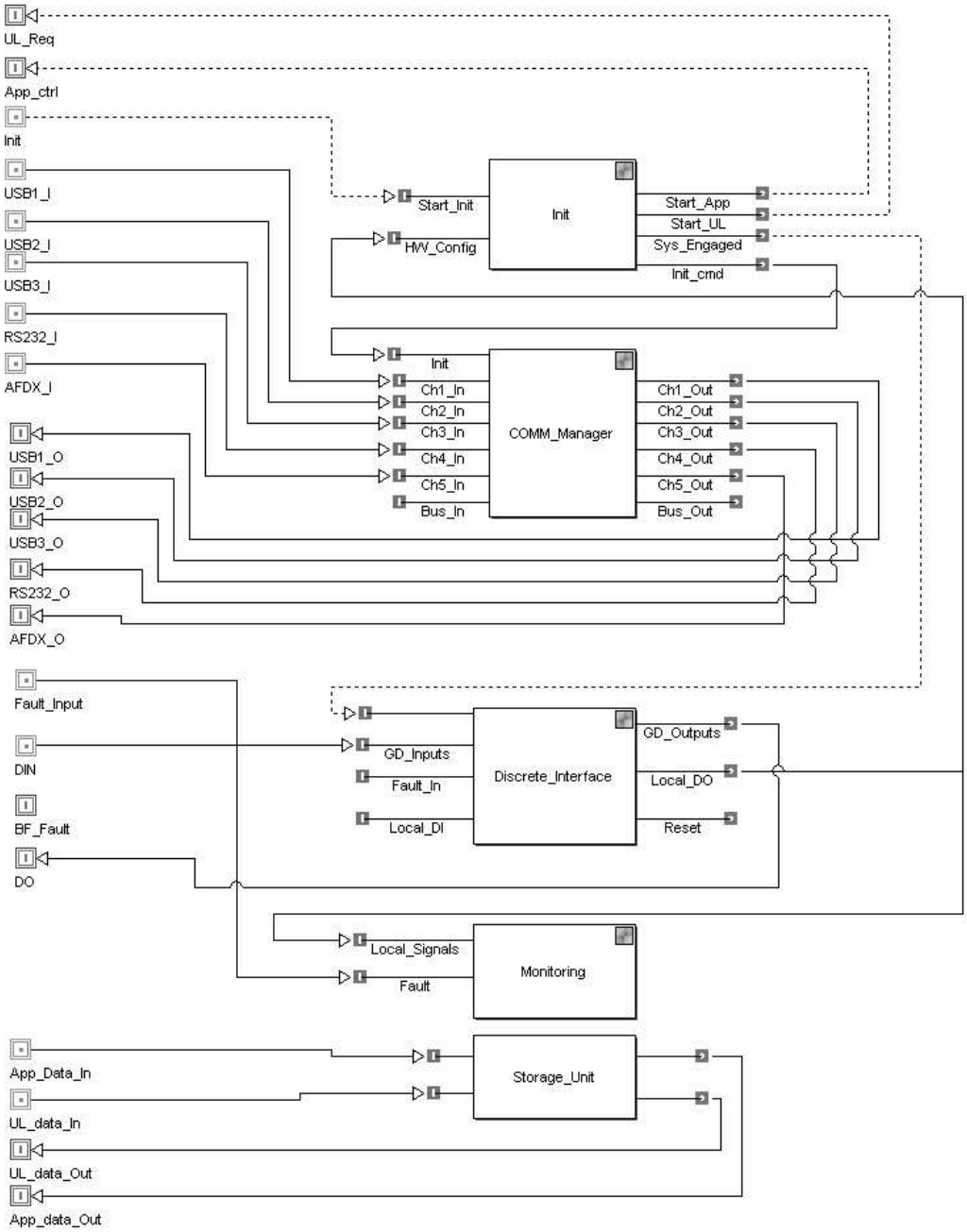


Figure 4-13. Description intérieure du bloc *Basic_functions*.

Pour le niveau –2 des fonctions de base du système, nous avons réalisé une agrégation préliminaire basée sur les fonctionnalités identifiées lors de la lecture des spécifications (§4.3.3) et sur l'avis de l'expert en conception de systèmes avioniques. Ce niveau regroupe la gestion des communications avec l'environnement, le stockage de données, l'initialisation du système…. Nous avons réalisé cette agrégation à cause de la diversité des fonctionnalités et pour faciliter la lecture de la représentation.

A ce niveau de la modélisation nous avons déjà constaté la manque de lisibilité « rapide » de notre approche. Tel que nous l'avons proposé (§2.4.7), des points de vue complémentaires facilitent la

compréhension d'une représentation HiLeS. La Figure 4-14 représente notre projet pilote sous un des possibles points de vue présentés précédemment, l'arbre du système.

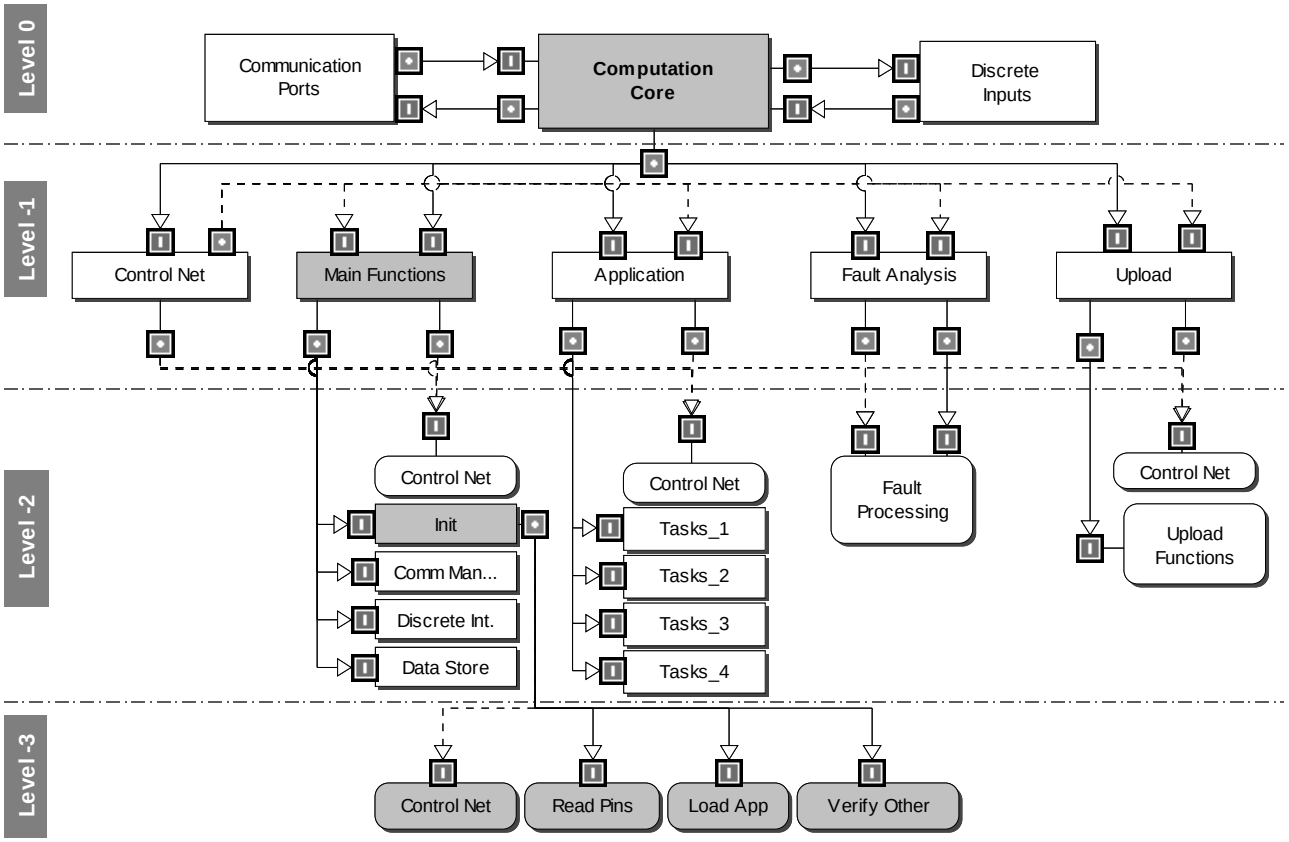


Figure 4-14. Un aperçu simplifié ou « mise à plat » de la structure hiérarchique du cœur de calcul sous la forme d'un arbre.

Pour développer notre exemple, nous allons décrire avec plus de précisions, une partie seulement du bloc des fonctions principales. Une vue simplifiée de l'ensemble de la structure décrite en HiLeS Designer est illustré dans le schéma de la Figure 4-14 ; il représente tous les niveaux de description (blocs gris) d'une des fonctions principales du système : ***Init***. Le schéma de la Figure 4-14 n'est pas un modèle architectural du système, c'est une décomposition fonctionnelle qui peut être réorganisée (fusion de blocs) pour devenir architecturale.

Modélisation de l'initialisation du système

Le bloc INIT est éclaté en 4 sous blocs qui conforment le niveau hiérarchique –3. La Figure 4-15 illustre la construction détaillée de l'intérieur du bloc. Ici, encore une fois, le réseau de contrôle coordonne les actions des blocs.

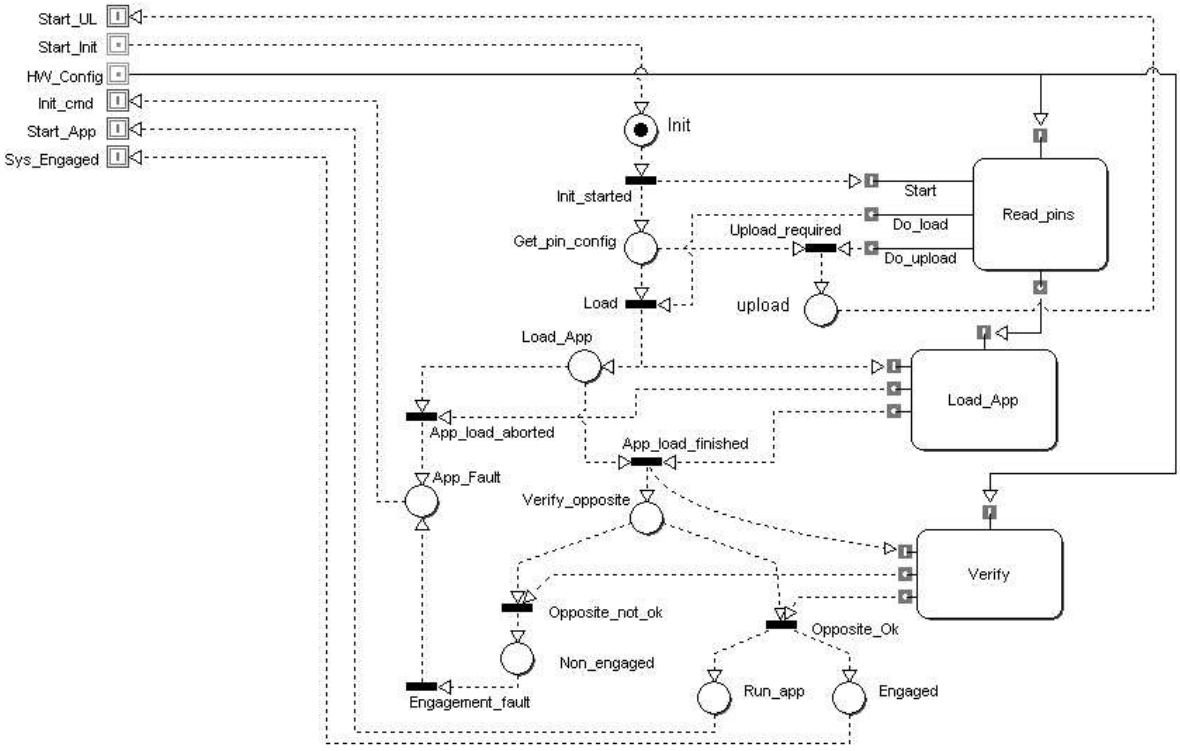


Figure 4-15. Vision détaillée de l'intérieur du block Init, la séquence d'initialisation du système est coordonnée par le réseau de Petri.

Après avoir défini la représentation détaillée représentée sur la Figure 4-15, nous avons lancé une analyse générale pour vérifier les propriétés du réseau de contrôle. Elle est lancée directement par une des fonctionnalités de HiLeS Designer. La version en cours du logiciel exécute en arrière plan les analyses de TINA. Les résultats sont présentés sous forme de texte, dans une fenêtre de l'interface graphique comme illustré sur la Figure 4-16.

Tina version 2.6.7 -- 05/28/04 – LAAS/CNRS mode -R INPUT NET ----------parsed net CCV_fdl04 9 places, 8 transitions net CCV_fdl04 tr App_load_aborted Load_App -> App_Fault tr App_load_finished Load_App -> Verify_opposite tr Engagement_fault Non_engaged -> App_Fault tr Init_started Init -> Get_pin_config tr Load Get_pin_config -> Load_App tr Opposite_Ok Verify_opposite -> Engaged Run_app tr Opposite_not_ok Verify_opposite -> Non_engaged tr Upload_required Get_pin_config -> upload pl Init (1) 0.000s REACHABILITY ANALYSIS ---------- Bounded 8 marking(s), 8 transition(s) MARKINGS: 0 : Init 1 : Get_pin_config 2 : Load_App 3 : App_Fault 4 : Verify_opposite 5 : Engaged Run_app	LIVENESS ANALYSIS ---------- Not live 3 dead marking(s), 3 live marking(s) 0 dead transition(s), 0 live transition(s) Dead marking(s): 7 5 3 STRONG CONNECTED COMPONENTS: 7 : 0 6 : 1 5 : 7 4 : 2 3 : 4 2 : 6 1 : 5 0 : 3 SCC GRAPH: 7 -> Init_started/6 6 -> Load/4, Upload_required/5 5 -> 4 -> App_load_aborted/0, App_load_finished/3 3 -> Opposite_Ok/1, Opposite_not_ok/2 2 -> Engagement_fault/0 1 -> 0 ->

6 : Non_engaged 7 : upload REACHABILITY GRAPH: 0 -> Init_started/1 1 -> Load/2, Upload_required/7 2 -> App_load_aborted/3, App_load_finished/4 3 -> 4 -> Opposite_Ok/5, Opposite_not_ok/6 5 -> 6 -> Engagement_fault/3 7 ->	0.000s ANALYSIS COMPLETED ----------

Figure 4-16. Résultats bruts de l'analyse du réseau de contrôle du bloc INIT.

Les résultats de l'analyse de cette partie du système prouvent que le réseau local est accoté et non vivant avec trois marquages morts. Ceci signifie que tous les états représentés peuvent être atteints à partir du marquage initial. Un nombre limité d'événements se produisent dans le système. Trois endroits représentent des impasses potentielles. Dans ce cas-ci, il est normal parce que les marquages morts rapportés correspondent aux sorties de ce niveau.

Ce genre d'analyse n'est pas très abordable pour les ingénieurs système : Un des objectifs courants est de développer une interface différente qui permet d'obtenir des propriétés spécifiques du système à partir de la vérification des réseaux de Petri.

La description fine des fonctions du système est réalisée en code VHDL-AMS contenu dans les blocs fonctionnels. HiLeS Designer fournit une interface spécifique qui permet d'écrire directement le code de chaque fonction. La fenêtre représentée de la Figure 4-17 est une capture d'écran de l'interface VHDL-AMS de l'outil, elle contient le code correspondant à un des blocs internes d'interface discrète, voient de niveau -2 sur la Figure 4-14.

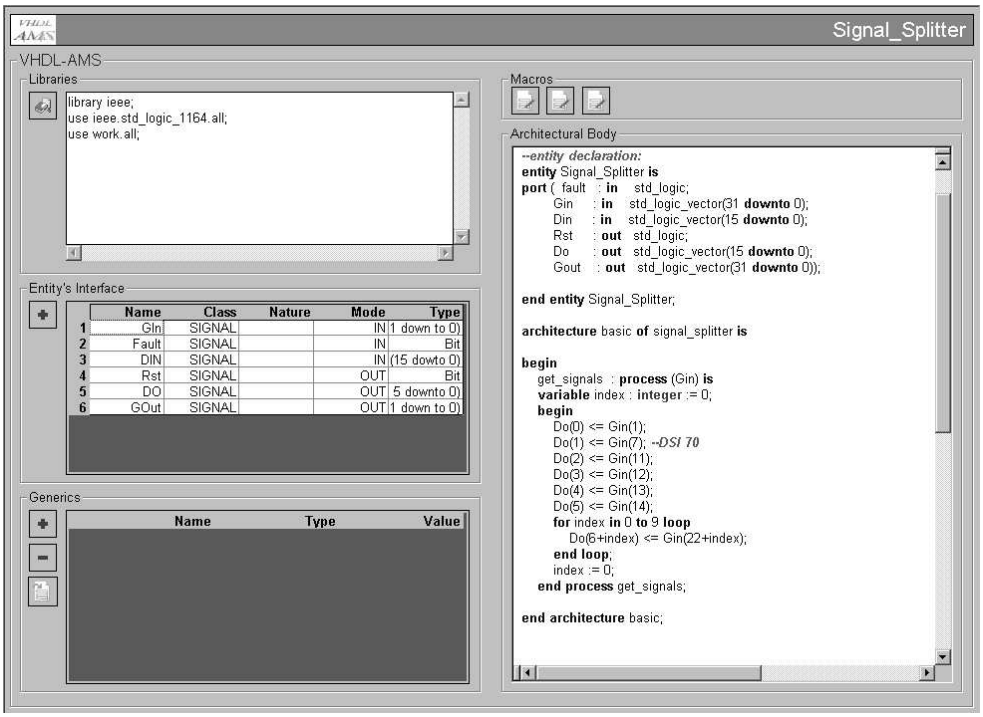


Figure 4-17. Fenêtre d'interface VHDL-AMS du logiciel HiLeS Designer. Nous pressentons ici le contenu d'un des blocs de "Discrete Interface" (Niveau –2).

Modélisation de l'accès aux communications

La gestion d'accès aux ressources de communications du système est une contrainte imposée par les spécifications. Pour la représenter, nous avons utilisé une configuration classique de partage de ressources par des réseaux de Petri. Nous illustrons dans Figure 4-18 le fonctionnement de base d'accès au ressource par des RdP et le contrôle de demande d'accès par le bloc fonctionnel *Ch_admin*.

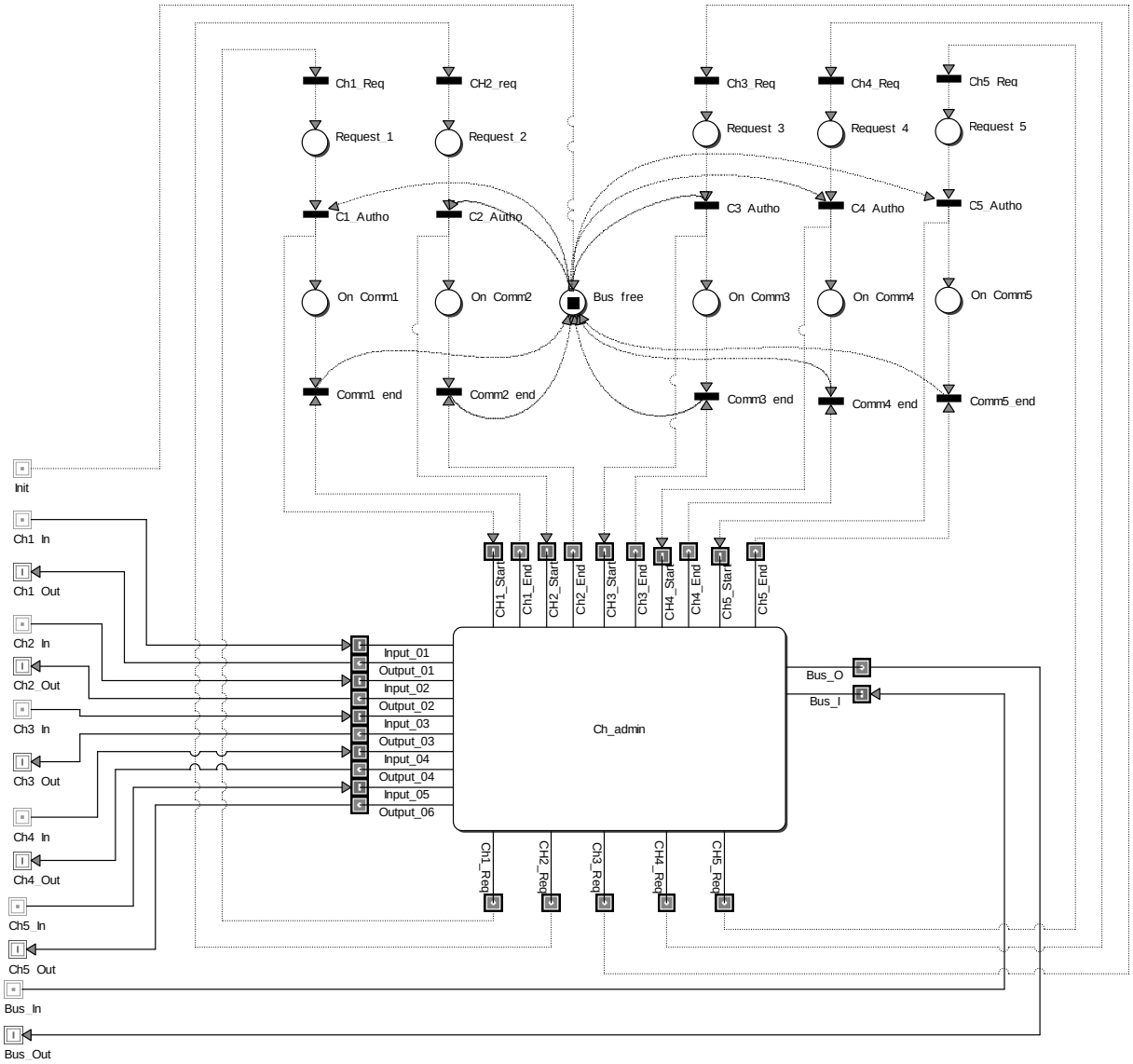


Figure 4-18. Représentation d'accès partagé aux ressources de communication.

Du point de vue du fonctionnement général du système, cette fonctionnalité reste « en veille » tant qu'elle n'est pas initialisée, ce sera la fonction d'initialisation, qui définira l'opération initiale de partage de ressource. Ceci se traduit par la disponibilité du bus, indiquée par le marquage de la place *Bus_free*. Dans un premier temps, cette place est vide. Elle sera marquée, par la première fois, uniquement sous contrôle de l'initialisation.

4.3.5 Perspectives

L'application projet pilote nous a précisé d'explorer les points suivants :

- la lecture et analyse des spécifications pour passer d'une représentation textuelle à une représentation graphique comme celle proposée par HiLeS,
- l'utilisation des premières interfaces automatiques opérationnelles entre HiLeS Designer et les autres éléments de notre plate-forme de conception système,
- la modélisation d'un système général d'accueil pour des algorithmes,
- la représentation du fonctionnement du système à partir des principaux états du système.

Cette étape est satisfaisante mais elle invite à des nouveaux développements :

Perspectives au court terme :

- l'urgence de faire évoluer nos procédés de validation formelle à base de réseaux de Petri. Nous n'avons pas développé assez le concept d'observateur.

- Un autre point et l'amélioration de l'ergonomie de la plate-forme par la génération de représentations : arbres de projet, séquences… La structure actuelle du logiciel, ainsi que les fonctionnalités implémentées permettent d'envisager la mise en œuvre de ces représentations complémentaires pour aider à la compréhension et la bonne lecture des projets HiLeS.

- A moyen terme, l'interprétation des projets HiLeS en XML permettra de communiquer avec d'autres domaines connexes de la conception strictement « technique », notamment avec la conduite de projet Les outils de documentation et représentation des paramètres non fonctionnels proposées dans nos CDF (§2.9.1.7) pourront être nourris et exploités par d'autres partenaires. Par ailleurs, la représentation HiLeS-XML est tout à fait nécessaire pour faciliter le « re-use » et la transportabilité des projets.

- Techniquement, la mise au jour de l'outil HiLeS Designer 0 sur Visual Studio .NET doit être mise en œuvre comme une dernière étape de HiLeS Designer 0. A plus long terme nous considérons nécessaire de revoir l'outil afin de proposer une plate-forme plus ouverte, facilitant les échanges d'information entre les partenaires d'un projet.

Perspectives au long terme :

- Ce lancement d'un programme de développement *HiLeS 1* est prévu avec l'utilisation de UML 2.0 (SysML) pour remplir le « gap » entre les spécifications et le formalisme HiLeS. La démarche sera de développer une méthode de transition entre les spécifications textuelles et les représentations semi-formelles UML (SysML™) conduisant au formalisme HiLeS. Les travaux comme [Del03] et [HF04] peuvent faciliter ces différentes transitions. Ce nouveau développement de HiLeS 1 se fera sur une plate-forme ouverte pour le rendre utilisable sous plusieurs systèmes d'exploitation. Les étapes principales de cette nouvelle étape pourront être :
  - L'interconnexion avec les approches standardisées issues de UML2, notamment SysML™.
  - La proposition d'un guide d'écriture et d'interprétation des spécifications et des besoins.
  - La mise en ouvre des outillages proposes dans notre travail (§3.8).

4.4 Conclusion

Ce chapitre visait à montrer l'exploitation de l'outil HiLeS Designer 0, sur deux exemples pour valider les concepts et identifier ses limites.

A ce jour, l'outil HiLeS Designer 0 est arrivé au stade opérationnel : il est en version 0V6. Il s'agit d'un outil de conception amont permettant de représenter les spécifications d'un produit en un modèle formel basé sur les Réseaux de Pétri et le langage VHDL-AMS. Ce modèle permet par une relation à l'outil TINA une certaine vérification des spécifications et la validation d'une architecture temporisée du système sous la forme de blocs fonctionnels interconnectés. Actuellement Il est en diffusion non commerciale pour susciter de l'intérêt et de donner accès à notre démarche sur le site www.laas.fr/toolsys/hiles. L'outil a été exploité sur plusieurs exemples de laboratoire, montrant qu'il était disponible à l'usage et utile à une description rapide, graphique d'un système et d'un microsystème.

Dans ce chapitre, nous l'avons, nous-mêmes, validé sur les points suivants :

- La richesse de la représentation graphique proposée. Cependant nous ne pouvons négliger la complexité des modélisations.
- La possibilité de réaliser des représentations amont faisant abstraction de l'architecture finale du système.
- Le fonctionnement correct de l'interface : HiLeS Designer – TINA
- La mise en fonctionnement d'une première version d'Interface semi-automatique avec le langage VHDL-AMS. Cependant, nous sommes encore loin de proposer la génération automatique de code.

L'outil est encore faible en matière de vérification. Le premier exemple, le calculateur ECP, a permis de tester la première version de notre outil dans un environnement industriel. Dans le second exemple, le cœur de calcul du CCV, nous nous sommes intéressés plus à l'application de notre démarche de conception de haut niveau. Un certain nombre de limites sont apparues pour lesquelles nous avons fait des propositions de nouveaux développements :

- Lecture de spécifications avec un guide d'interprétation. Ce dernier projet nous a permis d'envisager le rôle de UML dans notre démarche ainsi que de profiter de l'importante avis de l'expert sur la construction de la modélisation HiLeS.
- La mise en place des observateurs définis par l'utilisateur.
- L'amélioration des interfaces avec des outils de simulation VHDL-AMS. Pour incrémenter la flexibilité de HiLeS Designer nous pouvons envisager la création de « templates » spécifiques pour plusieurs outils de simulation VHDL-AMS. Par rapport à l'amélioration de la génération de code, un mécanisme de capture d'équations des fonctions est tout à fait envisageable. Pour l'étape que nous sommes entrain de finir nous avons contemplé cette possibilité mais nous l'avons considère hors des objectifs principaux.
- L'utilisation ou la proposition d'une méthode facilitant l'écriture du code VHDL-AMS associé aux blocs fonctionnels HiLeS.

5. CONCLUSIONS ET PERSPECTIVES GENERALES

Notre travail de recherche s'inscrit dans la préoccupation générale de **conception de systèmes complexes et pluridisciplinaires**. Il y a un besoin urgent de proposer des méthodes et des outils permettant au chercheur et à l'ingénieur de créer objets nouveaux au plus vite et « sans faute ».

Notre idée a été d'explorer la **conception amont** avec une exigence forte de produire une **modélisation formelle** sur laquelle puissent s'appuyer des étapes de vérification pour arbitrer des choix architecturaux et pour proposer des choix technologiques en améliorant les relations aux fournisseurs.

L'autre exigence forte de notre approche a été de se positionner en parfaite compatibilité avec le **langage VHDL-AMS** qui est un standard IEEE pour la description de systèmes pluridisciplinaires. En l'état actuel de notre analyse de l'existant, c'est un langage qui devrait s'imposer rapidement car il encourage les échanges de modèles et rend faisable l'objectif d'un **Prototypage Virtuel** des systèmes. Ce concept de prototypage virtuel est essentiel car il va constituer l'outil de dialogue entre tous les partenaires d'un projet qui permettra de prédire en amont, de plus en plus précisément les performances fonctionnelles et non fonctionnelles.

Sur ces principes, nous avons choisi de travailler en deux directions :

1. Développer un nouvel outil HiLeS Designer.
2. Placer cet outil dans une première étape de conception, en communication avec TINA, un outil de vérification de réseaux de Petri et avec les simulateurs VHDL-AMS.

Ces choix considérés, avec le recul, sont pertinents. Ils peuvent conduire vers un meilleur traitement des spécifications avec l'arrivée rapide des recommandations SysML™ et vers des démarches plus approfondies de vérification. Des nombreuses discussions ouvertes montrent que ce choix s'intègre bien dans les recommandations ENHANCE [Tho02] de traiter simultanément des questions de conception, de conduite de projet et de formation. Ils devraient bien s'intégrer dans la dynamique régionale autour de ces questions dans le cadre d'un pôle de compétitivité : aéronautique, espace, système embarqué.

Les résultats que nous avons obtenus ont été présentés en quatre chapitres :

Dans le **chapitre 1** nous avons réalisé un état de l'art des pratiques de conception de systèmes, à base d'électronique, en nous appuyant sur notre interprétation de la démarche générale de conception (Figure 1-3). Pour ce faire, nous avons parcouru les outils de conception électronique analogique, numérique et mixte. Puis nous avons révisé la conception des applications logicielles. Finalement nous avons considéré avec particulière attention les approches visant de répondre à la problématique de la conception de haut-niveau. Dans ce contexte nous identifions l'importance de démarches essayant d'être générales : d'un part, le langage VHDL-AMS, qui rapproche l'électronique analogique et numérique des autres domaines permettant de construire sous un seul standard des modèles pluridisciplinaires ; d'autre part UML qui depuis quelques années aborde la conception logicielle avec un regard système de haut-niveau. Des options « fédératrices » commencent à se mettre en place, cependant, au moment d'écrire ce mémoire, la communauté de l'ingénierie système ne propose pas encore une version consolidée ni opérationnelle. Nous avons conclu à la nécessité de

mettre en place une démarche réunissant trois aspects fondamentaux : l'utilisation d'une sémantique graphique de haut-niveau, la représentation des états fonctionnels du système avec un modèle formel tel que les réseaux de Petri et la compatibilité avec VHDL-AMS

Chapitre2. Nous proposons HiLeS. Il s'agit d'une approche généraliste pour la conception système de haut-niveau qui préconise l'utilisation d'une représentation graphique structurelle, inspiré de SA/RT. Le formalisme comporte aussi des réseaux de Petri pour représenter formellement le comportement général du système et le langage VHDL-AMS pour décrire les fonctions qui forment ce comportement. Nous avons réalisé une révision de la proposition originale présentée au LAAS en 2000. Nous avons étendu l'utilisation des réseaux de Petri à tout niveau de représentation, sauf l'environnement du système. Ce fait permet d'améliorer la lisibilité de la représentation en s'appuyant sur l'établissement de niveaux hiérarchiques, y compris pour les réseaux de Petri. Nous avons réalisé la mise en œuvre du formalisme cous la forme d'un outil logiciel avec plusieurs fonctionnalités. Parmi eux, nous avons introduit la possibilité d'exprimer plusieurs options chaque module HiLeS, ainsi que le principe d'agrégation nécessaire pour transformer une représentation fonctionnelle en architecturale. Nous préconisons également, l'extraction de vues complémentaires au formalisme, ces **représentations associées** permettront de regarder le système des plusieurs points de vue selon les nécessités immédiates des équipes de conception-conduite de projet.

Chapitre3. Nous proposons la construction d'une plate-forme de conception de haut-niveau autour de l'outil HiLeS Designer. De coté des spécifications nous ne proposons pas d'outil mais nous recommandons d'élaborer des interconnexions avec le langage UML (SysML™) pour établir une guide rédaction et d'interprétation.

Afin d'aboutir à une version utilisable de cette plate-forme nous avons crée des **« liens opérationnels » ou interfaces** avec d'autres outils. Pour les aspects liés à la vérification formelle nous réalisons l'extraction des réseaux de Petri pour les analyser avec TINA. L'exécution des analyses est lancée à partir de HiLeS Designer et réalisée en arrière plan par TINA. Toutefois, l'exploitation des résultats s'avère assez compliqué, en conséquence nous envisageons d'approfondir l'utilisation d'observateurs orientés à vérifier propriétés particulières du modèle, écrites en langage métier du concepteur. Par rapport à VHDL-AMS, nous avons réalisé une première interface permettant de générer automatiquement la structure blocs de la représentation ainsi que les réseaux de Petri associes. Le contenu des blocs, c'est à dire le code des fonctions, dépende entièrement de l'utilisateur et ne fait pas partie d'aucun traitement de la part de notre outil.

Chapitre4. Mise en application. Nous illustrons par deux exemples deux étapes de notre projet. Le premier a servi à tester par la première fois une version opérationnelle de HiLeS, il à mis en évidence certains insuffisances de cette première implémentation et a ouvert la voie pour la mise en œuvre d'une version plus stable et ergonomique. Le deuxième exemple nous a confrontés à un problème réel de conception dans laquelle nous avons expérimenté la difficulté de réaliser des représentations de haut-niveau sans nous rapprocher, très vite, de l'architecture. Nous avons utilisé comme complément à notre démarche deux diagrammes UML afin de comprendre le fonctionnement décrit par les spécifications. Cependant, ce qui est devenue cruciale est l'intervention de l'expert lors de la définition des états principaux du système. Lors de la modélisation HiLeS, la complexité et la richesse de celle-ci nous a conduit a nous appuyer sur des représentations associes permettant de visualiser plus aisément la globalité du système. Nous avons validé le fonctionnement des outils de la plate-forme et nous avons identifié des aspects importants à améliorer et corriger.

Pour l'avenir, notre programme de travail c'est de conclure le projet MOCAS, de proposer un espace commun avec GESOS et sûr tout, de lancer une nouvelle phase de développement HiLeS 1 dans le cadre d'une forte collaboration avec Airbus France et d'autres éventuelles partenaires européens.

Dans le cadre de la thèse, nous avons conduit d'autres activités liées à notre démarche méthodologique et à notre formation sur la conception système. Nous avons organisé ces activités autour du club **TOOLSYS** qui est devenu une cellule de réflexion et d'échanges sur la conception de systèmes complexes et pluridisciplinaires. Les membres de TOOLSYS sont des utilisateurs des outils CAO pour la conception système, des spécialistes de la conduite des projets de conception et des concepteurs d'outils de télé travail. Plusieurs séminaires et conférences ont été organisés et deux stages ouverts de formation en VHDL-AMS.

Nous avons également créé le site web: http://www.laas.fr/toolsys afin de faciliter:La diffusion de notre approche.

- La discussion ouverte sur la conception système.
- La proposition de rencontres internes et avec des spécialistes extérieurs.

Un projet transversal LAAS sur la conception système a été crée entre trois groupes recherche, puis ouvert aux membres du club TOOLSYS. Ce projet qui préfigure une action à plus long terme dans l'environnement Midi-Pyrènes, comporte les objectifs suivants :

- Promouvoir une démarche de **conception système et microsystème** au LAAS pour la conduite de projets internes et l'accueil de projets externes tels que IMPACT, RTB…
- Explorer plus particulièrement la conception amont en s'appuyant sur le développement d'un **outil nouveau: HiLeS Designer**, compatible VHDL-AMS, TINA et la gestion des IP's.
- Réaliser d'autres **exemples** d'application pour illustrer l'intérêt de la démarche.
- Multiplier les visites, **la formation**, l'accueil des spécialistes, les rencontres et séminaires pour renforcer la dynamique Conception Système et Microsystème.
- Préparer la mise en place d'une **plate-forme LAAS de prototypage virtuel**.

Une dernière facette de cet environnement favorable est la perspective de nouvelles collaborations entre le LAAS et Airbus France. Des discussions sont en cours sur le développement d'une nouvelle étape HiLeS 1. Le lancement d'un projet européen et l'élargissement de la collaboration actuelle dans le cadre du pole de compétitivité : Aéronautique, Espace, systèmes Embarqués, en préparation.

6. GLOSSAIRE

Browsers Outils informatiques de recherche, largement connus pour leur application dans les moteurs de recherche Internet.

CAN Convertisseur Analogique / Numérique.

CAO Conception Assistée par Ordinateur.

CDF Complementary Data Files

CNA Convertisseur Numérique / Analogique.

CRC Code de redondance cyclique (Cyclic redundancy code). Il s'agit d'un algorithme mathématique destiné à contrôler l'intégrité des données ou d'un fichier lors de transferts par un réseau ou via internet. Chaque fichier auquel on fait passer un CRC renvoie une valeur unique qui fait que même si un seul octet du fichier est modifié le CRC ne sera plus le même. C'est pour cela que le CRC est utilisé dans détection d'erreurs, pour contrôler l'éventuelle modification du contenu des mémoires ou des modifications dans les trames de données de certains protocoles de communications.

ECAM Electronic Centralized Alarm Module

ECP ECAM Control Panel

EDA Automatisation de la conception électronique (Electronic Design Automation).

Firmware Logiciel spécifique très proche du matériel et généralement embarqué dans une mémoire type ROM, EPROM, ou EEPROM. Ce logiciel permet de gérer les fonctions de base d'un système electronique.

FVT Formal Verification Tool

GDSII Abbreviation pour "Graphic Design Station II". GDSII est un format de données propriétaire de chez Cadence Design Systems pour l'échange de données graphiques bi-dimensionnel. Il est devenu un standard de fait pour les échanges des informations concernant le layout des circuits intégrés.

Hardware in the loop C'est une technique de simulation dans laquelle un système électronique simule le comportement d'un système physique tel q'un moteur, des freins ou un système d'injection de combustible.

HDL Hardware Description Language

MSHDLs Mixed-Signal Hardware Description Languages

PCB	Printed Circuit Board
PCI	'Peripheral component interconnect.' A type of 32-bit bus that has widely replaced the earlier 16-bit ISA bus. Having a 32-bit rather than a 16-bit bus gives the PC a speed improvement because a greater amount of data can be transferred in a given time. PCI and ISA peripherals are not compatible, but all except the most specialised are now available in PCI format as well as ISA.
RdP	Réseaux de Petri.
RTL	Register transfer level
PTS	Purchaser Technical Specification
STI	Spécification Technique Industrielle.
Testbench	Un « testbench » est la partie d'un modèle de VHDL qui lit ou produit d'un ensemble de vecteurs d'essai et les envoie au module étant testé. Le testbench compare les réponses faites par le module à l'essai contre des spécifications des résultats corrects. Ainsi le testbench est une partie intégrale de n'importe quel modèle de VHDL.
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

7. BIBLIOGRAPHIE

Dans ce document les références bibliographiques sont détaillées dans cette section, elles sont triées alphabétiquement. Le format utilisé permet d'identifier chaque référence par un acronyme entoure de crochets [] qui correspond à la structure suivante :

Pour les références d'un seul auteur : Les trois premières lettres du nom dont la première en majuscule suivie de l'année de publication, en cas de plusieurs publications dans la même année une nomenclature est rajoutée a la fin (a, b, c…). Par exemple [Aut98a].

Pour les références à plusieurs auteurs l'acronyme est composé par les initiales des auteurs en majuscules. En cas de plus de trois auteurs nous rajoutons un signe « + » entre les auteurs et l'année de la publication. Par exemple : [JHG97] ou [FGD+99].

Le format des références aux sections de ce document est le numéro de section précédé par le signe §, le tout entre parenthèses. Exemple : (§5.2) fait référence à la section 2 du chapitre 5.

[ABG04] D. ANDREU, N. BRUCHON, T. GIL. « Du modèle à l'exécution : Traduction automatique de réseau de Petri interprété en langage VHDL ». Rapport de recherche LIRMM n°04008 juillet 2004

[Acc03] Accellera. « Accellera Rosetta Standards Comitee Homepage ». [En ligne]. Adresse URL : http://www.accellera.org/. Accellera Organization, Inc 2003

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

[Air01] AIRBUS France. « Purchaser Technical Specification FWS A380 ». Airbus France. Décembre 2001.

[Air02] AIRBUS France. « Purchaser Technical Specification Flight Control And Guidance Unit. Issue 3.1». Airbus France. Mars 2002.

[Ana01a] Analogy, Inc. « Saber® / Verilog-XL® Co-Simulation Interface ». Analogy, Inc, Beaverton, Oregon, Etats Unis d'Amérique, 2001.

[Ana01b] Analogy, Inc. « Saber® / ModelSim ™ Co-Simulation Interface ». Analogy, Inc, Beaverton, Oregon, Etats Unis d'Amérique, 2001.

[Ana01c] Analogy, Inc. « Saber® / ViewSim ® Co-Simulation Interface ». URL : http://www.analogy.com/Products/simulation/simulation.htm#ViewSim. Analogy, Inc,

Beaverton, Oregon, Etats Unis d'Amérique, 2001.

[And96] C. ANDRE. « Representation and Analysis of Reactive Behaviors: A Synchronous Approach » CESA'96, IEEE-SMC, Lille, France, 9 au 12 juillet, 1996;

[Ans03] ANSOFT Corporation, « System Modeling » [En ligne]. Adresse URL : http://www.ansoft.com/products/em/simplorer/. ANSOFT Corporation. Etats Unis d'Amérique d'Amérique 2003.

[Ber02] B. BERTHOMIEU. Time Petri Net Analyzer ». [En ligne], Adresse URL : http://www.laas.fr/tina/. LAAS-CNRS, Toulouse, France. 2002.

[BEY04] C.BARON , D.ESTEVE , M.YACOUB. "Interests of genetic algorithms to select and optimize scenarios in a system design process". IEEE International Symposium on Industrial electronics, Ajaccio (France), 4-7 mai 2004.

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

[Bra02] H.J. BRAUDEL. « Introduction to Forum 3 ». ENHANCE Consortium. Toulouse France. Avril 23 et 24 2002.

[BRE04] C. BARON, S ROCHET, D. ESTEVE. "GESOS: a multi-objective genetic tool for a project management considering technical and non-technical constraints". IFIP 18th World Computer Congress. 1st IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI'2004), Toulouse (France), 22-27 Août 2004.

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

[BRV03] B. BERTHOMIEU, P.O. RIBET, F. VERNADAT. *« L'outil TINA – Construction d'espaces abstraits pour les réseaux de Petri et réseaux temporels »* Modélisation de Systèmes Réactifs 2003. (MSR' 2003). Metz, October 2003.

[BS01] BARCO SILEX. " SoC Design Methodology". Présentation pendant le seminaire Design and Reuse, Grenoble, France le 17 octobre 2001. Barco Silex , Louvain-la-Neuve, Belgique.

[CCE+99] F. CLOUTE, J. N. CONTENSOU, D. ESTEVE, P. PAMPAGNIN, P. PONS, Y. FAVARD, "Hardware/ software co-design of an avionics communication protocol interface system: an industrial case study". 7th ACM/ IEEE International Workshop on Hardware/ Software Codesign (CODES'99), Rome (Italie), 3-5 mai, pp. 48-51.

[CDS03a] Cadence Design Systems Inc, « Using PSpice » [En ligne]. Adresse URL : http://www.orcadpcb.com/pspice/default.asp?bc=F. Cadence 2003.

[CDS03b] Cadence Design Systems. "Personal Productivity Solutions: OrCad unison suites and technologies". Cadence Design Systems, Inc. 2003.

[CDS04a] Cadence Design Systems. "Cadence ENCOUNTER digital IC design platform. Digital implementation for nanometer ICs". Cadence Design Systems, Inc. San José, Californie, Etats Unis. 2004.

[CDS04b] Cadence Design Systems. "NC-VHDL Datasheet". Cadence Design Systems, Inc. San José, Californie, Etats Unis. 2004.

[CE00] Condor Engineering. « ARINC Protocol Tutorial Manual » Condor Engineering, Inc. Santa Barbara CA, 2000

[Cel02] Celoxica Ltd. « HANDEL-C language Overview ». Celoxica Ltd, août 2002.

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

[Cla01] P. CLARKE. « ESTEREL system-level language emerges from the lab ». EE TIMES, EE Times Network 2001.

[Clo01] F. CLOUTE, « Etude de la conception des systèmes embarqués sur silicium : Une approche de codesign matériel / logiciel », Thèse doctorat, Institut National Polytechnique de Toulouse, 2001.

[CMM04] CCARR C.T, McGINNITY T.M et McDAID L.J. “Integration of UML and VHDL-AMS for analogue system modeling”. Formal aspects of computing, 2004, vol. 16 , no 1 , pp. 80 – 94.

[Cof03] Cofluent Design. « The MCSE Methodology Overview ». Cofluent Design 2003.

[Coo01] R. Scott COOPER. “The Designer’s Guide to Analog & Mixed-Signal Modeling”. Avant Corp. ISBN 0-9705953-0-1. 2001.

[Cow01] CoWare, Inc. « CoWare N2C Design System », Coware N2C Data Sheet, Coware, Inc, Santa Clara, Californie, Etats Unis d’Amérique 2001.

[CSC+04] J.M. CARDOSO; J.B SIMOES; C.M.B.A CORREIA, A. COMBO, R. PEREIRA, J; SOUSA, N. CRUZ, P. CARVALHO, N. VARANDAS. “A High Performance Reconfigurable Hardware Platform for Digital Pulse Processing C.A.F”.;Nuclear Science, IEEE Transactions on , Volume: 51 , Issue: 3, Pages:921 – 925. juin 2004.

[DAR02] Design And Reuse. « IP Catalog Builder™ ». [En ligne], Adresse URL : http://www.us.design-reuse.com/. Design and Reuse. Grenoble France. 2002.

[DD00] B. DION, S. DISSOUBRAY, “Modeling and implementing critical real-time systems with Esterel Studio”, Esterel Technologies 2000.

[Del03] J. DELATOUR. “*Towards the analysis of real-time systems: The UML/PNO approach”*, Ph.D. Thesis, LAAS-CNRS. Toulouse, France. September 2003.

[Dew03] T. DEWEY. “Design Documantation with HDL Designer™”. Mentor Graphics Corporation. Etats Unis 2003.

[DGG01] R. DOMER, A. GERSTAULER, D. GAJSKI, « SpecC Language Reference Manual, Version 1.0 », Université de Californie, Irvine,Mars 6, 2001.

[DH97] T. DEIß, T. HILLENBRAND. “A Case Study on the Use of SDL”. SFB 501 Report No. 03/97, SFB 501 University of Kaiserslautern, 1997.

[DHJ+02] J. DAVIS II, C. HYLANDS, J. JANNECK, E. LEE, J. LIU, X. LIU, S. NEUENDORFFER, S. SACHS, M. STEWART, K. VISSERS, P. WHITAKER, Y. XIONG. “Overview of the PTOLEMY Project”, Department of Electrical And Computer Science, University Of California. Berkeley mars 6, 2002.

[DI03] Dolphin Integration. “Dolphin Medal. New Features in SMASH™ 5.0.0 – 5.1.3”. Dolphin Integration 2003.

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

[Dob03] A. DOBOLI. «Towards Automated Synthesis of Analog and Mixed-Signal Systems from High-Level Specifications». University of New York. FDL 2003. Frankfurt, Germany, September 2003.

[Dol03] L. DOLDI. "UML 2 Illustrated. Developing Real-Time and Communication Systems". LegoPrint S.p.a, Italy, 2003.

[DPK00] A. DIAZ-CALDERON, C.J. PAREDIS, P.K. KHOSLA. “Reconfigurable Models: A Modeling Paradigm to Support Simulation-Based Design”. 2000 Summer Computer Simulation Conference. Vancouver, British Columbia, Canada July 2000

[Dua04] H. DUARTE. « Incorporación de herramientas de validación y verificación de redes de Petri, mediante TINA, en HiLeS Designer ». Universidad de los Andes. Bogota, Colombie, 2004.

[DV03] A. DOBOLI, R. VEMURI. “A VHDL-AMS Compiler and Architecture Generator for Behavioral Synthesis of Analog Systems”, Proceedings of DATE'99, pp.338-345, 1999.

[Enh02] Enhanced Aeronautical Concurrent Engineering. « ENHANCE Forum 3 Proceedings ». ENHANCE Consortium, Toulouse France. Avril 23 et 24 2002.

[Eur04] EUROPRACTICE Software Service, « RAL-EUROPRACTICE Software Service Home Page » [En ligne]. Adresse URL : http://www.te.rl.ac.uk/europractice/. 2004.

[Fau90] C. FAURE. « Interconnexion de réseaux informatiques et réseau numérique à intégration de services : modélisation et vérification d’architectures de communication », thèse doctorat, Laboratoire d’Analyse et d’Architecture des Systèmes du CNRS. Toulouse, France 1990.

[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.

[FLL+98] E. FILIPPI, L. LAVAGNO, L. LICCIARDI, A. MONTANARO, M. PAOLINI, R. PASSERONE, M. SGROI, A. SANGIOVANNI-VINCENTELLI. « Intellectual Property Re-use in Embedded System Co-design: an Industrial Case Study ». Proceedings of *International Symposium System Synthesis*, Hsinchu, Taiwan, December 1998

[GO04] C. GUTIERREZ, A. OUARDANI. « Fiche Partagée pour la Conduite de Projet et la Conception de Système ». LAAS-CNRS. 1 octobre 2004.

[GEV04] C. GRIMM, K EINWICH et A. VACHOUX. « Analog and Mixed-Signal System Design with SystemC ». FDL’04 Tutorial. Septembre 16 2004. Lille, France.

[Gom01] M. GOMEZ. « Hardware-in-the-loop Simulation ». Embedded System Programming ». Etats Unis d’Amérique, novembre 2001.

[GP99] C. GOLDFARB, P. PRESCOD. "Manual de XML". Prentice Hall, Madrid, Espagne 1999.

[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.

[Ham01] J.C HAMON. « Plate-forme de Prototypage Virtuel, Conception Système et « Co-design » micro électronique », Stage DEA CCMM, Institut National Polytechnique de Toulouse, 2001.

[Ham03] J.C. HAMON. « Spécification de l’outil de conception amont : HiLeS Designer » LAAS-CNRS. 2003.

[Har00] N. HARCHANI. « Etude d'une méthodologie de conception descendante des micro systèmes : Conception d'un Micro Système pour la Surveillance de Contraintes Mécaniques en Aéronautique », Thèse doctorat, Institut National Polytechnique de Toulouse, France 2000.

[Hau03a] M. HAUSE.", "The UML™ for Systems Engineering Initiative". International Council on Systems Engineering (INCOSE) UK Chapter Newsletter. Septembre 2003.

[Hau03b] M. HAUSE.", "The UML™ for Systems Engineering Initiative Part Two". International Council on Systems Engineering (INCOSE) UK Chapter Newsletter. Novembre 2003.

[Hau04a] M. HAUSE.", "The UML™ for Systems Engineering Initiative Part Three". International Council on Systems Engineering (INCOSE) UK Chapter Newsletter. Mars 2004.

[HEP+03] J.C.HAMON , D.ESTEVE , P.PAMPAGNIN , P.SCHMITT , J.Y.FOURNIOLS. « Une proposition de plate-forme pour la conception système et microsystème Journées » Scientifiques Francophones (JSF'2003), Tozeur (Tunisie), 20-22 décembre 2003.

[HEP03] J.C HAMON, D. ESTEVE, P. PAMPAGNIN. « *HiLeS Designer: A Tool For Systems Design* ». CONVERGENCE '03: Aeronautics, Automobile & Space International Symposium. Paris (France), 1-3 December 2003.

[HEP04] J.C. HAMON, D. ESTEVE, P. PAMPAGNIN. « High level system design using HiLeS designer » Forum on specification & design languages, FDL'04. Lille, France, 13-17 septembre 2004.

[Her02] Y. HERVE. « VHDL-AMS : «Applications et enjeux industriels». Dunod-Université - collection : Sciences-sup - préface d'Alain Vachoux. ISBN : 2-10-005888-6 - mars 2002.

[Her03] Y. HERVE. « Simple Models for Complex Systems : A-FSM Template ». Forum on design and specification Languages FDL'03. Frankfort, Allemagne 2003.

[HF04] Y. HERVE, A. FAKHFAKH. "Specification and Verification through an extension of VHDL-AMS". FDL'04 Forum on Specification & Design languages. Lille, France, 14-17 September 2004.

[HJA+00] N HARCHANI., F. JIMENEZ., M. ALMOHAMED, M. COURVOISIER, D. ESTEVE. "Tools and Models for Systems Design and Synthesis Of MEMS based on Asynchronous Circuits". IFAC 4th Symposium SICICA, Buenos Aires, Argentina, 13-15 September 2000.

[HSE+03] JC. HAMON, P. SCHMITT, D. ESTEVE, H. CAMONi, JY. FOURNIOLS. « *Mise en place d'outils de conception de micro-systèmes : Application à la conception d'un accéléromètre* ». LAAS-CNRS. JNRDM 03 Toulouse France 2003.

[IEEE99] IEEE 1076.1-1999 standard, Language Reference Manual. "VHDL Analog and Mixed Signal extensions". ISBN 0-7381-1640-8.

[IUN98] H INAMORI, K. UEDA; K. NISHIKAWARA. **"**Common software platform for realizing a strategy for introducing the TMN". Network Operations and Management Symposium, 1998. NOMS 98., IEEE , Volume: 2 , 15-20 février 1998 Pages:579 - 589 vol.2.

[Jer02] A. JERRAYA. « Conception de haut-niveau des systèmes mono-puces ». Lavoisier, Paris 2002.

[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.

[KC97] A. KAABOUCH, J.-N. CONTENSOU. « Le cablage des algorithmes par SYNCIRC ».

Premier colloque du GDR CAO Villard de Lans (Grenoble), 15-17 janvier 1997.

[KHG+04] T.KAZMIERSKI, J.C.HAMON, C.GRIMM, W.HEUPKE, K.WALDSCHMIDT, Y.HERVE, P.DESGREYS, A.DOBOLI. "VHDL-AMS in the design flow from high level specification to automated synthesis". Acceptée à IEE Special Issue CDT.

[Lem99] P. LEMAN. « Modelisation et simulation des instrumentations complexes à architectures materielle et logicielle ». These de doctorat Université paul Sabatier. Toulouse France 1999.

[LW98] T. LIEBICH, J. WIX. « Highlights of the Development Process of Industry Foundation Classes ». International Alliance for Interoperability. 1995.

[Mal02] D. MALINIAK. « From CAD to CAE to EDA, Design tools have wrestled with complexity ». ED Online ID #2311. Penton Media, Inc. juin 2002.

[Mar04] J.N. MARTIN. "Overview of the EIA 632 Standard: Processes for Engineering a System", URL: http://www.incose.org/stc/OvrwEIA632.htm. INCOSE 2004.

[MCEB04] R. MAURICE, E. CAMPO, D. ESTEVE, D. BOUCHET. "Méthodologie de conception pour la réalisation d'un microsystème multicapteurs autonome communicant". LAAS-CNRS. JNRDM 04 Marseille, FRANCE 2004.

[MG00a] Mentor Graphics Corporation, « Seamless CVE User's Reference and Manual, software version 4.0 », Mentor Graphics 2000.

[MG00b] Mentor Graphics Corporation, « Getting Started with Seamless CVEl, software version 4.0 », Mentor Graphics 2000.

[MG01a] Mentor Graphics Corporation, « AdvanceMS Datasheet » Mentor Graphics Corporation.2001;

[MG03] Mentor Graphics Corporation, « System Modeling » [En ligne]. Adresse URL : http://www.mentor.com/systemvision/. Mentor Graphics Corporation.2003.

[MG03] Mentor Graphics, "HDL Designer Data Sheet", Mentor Graphics Corporation, 2003

[MG04] Mentor Graphics, "Platform Express™ Datasheet", Mentor Graphics Corporation, 2004

[Mil03] R. MILLER. « Practical UML™: A Hands-On Introduction for Developers ». Borland Developer Network. [En ligne]. Adresse URL : http://community.borland.com/article/0,1410,31863,00.html. Borland Software Corporation, Inc, 2003.

[MM98] E. MOSER, N. MITTWOLLEN. "VHDL-AMS: The Missing Link in System Design - Experiments with Unified Modelling in Automotive Engineering". Proceedings DATE'98 Design Automation and Test in Europe. Paris, France, 23-25 février 1998.

[Mul02] P. MULLEY. "Sharing Your HDL Design Through a Web Browser". Mentor Graphics Corporation. Etats Unis 2002.

[Nak03] Y. NAKAMOTO. "The next generation software platform for mobile phones"**.** Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03) , 14-16 Pages:46 – 48. May 2003.

[OG04] J. OUDINOT, S. GUESSAB. "Can MATLAB/Simulink Co-Exist with VHDL-AMS?". Euro DesignCon 2004. Munich, Allemagne. 11-14 octobre 2004.

[Omg03] Object Management Group, Inc., "OMG Unified Modeling Language Specification, Version 1.5", Object Management Group, Inc. Etats Unis d'Amérique 2003.

[Per02] A. PERRY. "System Specifications Using Rosetta". Design Automation Conference" DAC'39. New Orléans, Etats Unis d'Amérique. 10-14 juin 2002.

[Per04] V. PERRIER. "System Architecting complex designs". Embedded Systems Europe. Janvier/février 2004. pp. 24 - 26.

[PHM+04] L. PIAZZA, J.C. HAMON, R. MAURICE, D. ESTEVE, A.M. GUE. « Conception et réalisation de l'environnement d'un micro-système de dépôt localisé pour la synthèse de l'ADN à l'aide du logiciel HiLeS-Designer ». Rapport LAAS N°04141, février 2004.

[Rai01] P. RAINJENNOU. « Spécifications Techniques Préliminaires pour l'ECP A380 ». AIRBUS France. Toulouse. Novembre 2001.

[Ros72] DT, ROSS. "Structured Analysis and Design Technique (SADT)". The Massachusetts Institute of Technology. Cambridge, Massachusetts, Etats Unis 1972.

[Sal04] H. SALZWEDEL. « Mission Level Design of aerospace systems ». MLDesign Technologies, Inc. DASC, Salt Lake City, octobre 2004.

[San96] A. SANGIOVANNI-VINCENTELLI. « Trends in Electronic Systems ». Proceedings of Mediterranean Electrotechnical Conference on Industrial Applications in Power Systems, Computer Science, and Telecommunications, mai 1996.

[Sch03] R. SCHWITTER. "PENG – Processable English", URL: http://www.ics.mq.edu.au/~rolfs/peng/. Centre for Language Technology, Macquarie University of Sydney 18th November 2003.

[SDC99] P. De SAQUI-SANNES, L. DAIRAINE et C. CHASSOT. « Expérimentation d'outils pédagogiques destinés à l'ingénierie des protocoles ». *ENSICA, °LAAS-CNRS, Toulouse France 1999.*

[Sim03] SIMEC GmbH & Co KG. « hAMSter The High Performance AMS Tool for Engineering and Research ». [En ligne]. Adresse URL : http://www.hamster-ams.com/. SIMEC GmbH & Co KG, 2003.

[SP04] SysML Partners. « System Modeling Language™: SysML™ . Version 0.3 ». SysML Partners February 4, 2004.

[Syn00] Synopsys, « Hardware/Software Co-Verification with Synopsys Eaglei Tools », Synopsys, Inc, Etats Unis d'Amérique 2000.

[Syn03] Synopsys®. « Saber HDL: Language-Independant Mixed-Signal Multi-Technology Simulator ». Synopsys® Etats Unis d'Amérique 2003.

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

[Syn99] Synplicity®. "Designing Safe VHDL State Machines with Synplify". Sinplicity, Inc. 1999;

[Sys03] SysML Partners. « Systems Modeling Language™: SysML™ , version 0.3 (first draft)». SysML Partners 12 January 2003.

[Tho02] J.M THOMAS. « Importance of New Ways of Working in the Airbus Virtual Enterprise Environment from A380 to future trends ». ENHANCE Forum 3. Toulouse, avril 2002.

[TMW03] The MathWorks. « Link for ModelSim® 1.1 ». The mathWorks, Inc. 2003.

[TMW04a] The MathWorks. « System Specification and Modeling ». [En ligne]. Adresse URL : http://www.mathworks.com. The Mathworks, Inc. 2004.

[TMW04b] The MathWorks. « Embedded System Design ». [En ligne]. Adresse URL : http://www.mathworks.com. The Mathworks, Inc. 2004.

[TMW04c] The MathWorks. « Aerospace and Defense – Engineering Tasks ». [En ligne]. Adresse URL : http://www.mathworks.com. The Mathworks, Inc. 2004.

[Too04] Toolsys. “Toolsys: Projets en cours”. [En ligne], Adresse URL : http://www.laas.fr/toolsys/projets.htm. LAAS-CNRS, groupe MIS. Toulouse France. 2004.

[Too99] Toolsys. “Open toolset for mixed-simulations of multi-domains systems” Adresse URL : http://www.laas.fr/Toolsys_old/index.html. Toolsys 1999.

[VG01] F. VAHID, T. GIVARGIS. “Platform tuning for embedded systems design”. Computer, Volume: 34, Issue: 2 , Zeb 2001 Pages:112 – 114.

[Wal98] N. WALSH. « What is XML? ». The O’Reailly xml.com XML from the inside out. Adresse URL : http://www.xml.com/pub/a/98/10/guide1.html. Octobre 1998.

[WM85] P. WARD, J. STEPHEN J. MELLOR. “Structured Development for Real-Time Systems”. Vol. 1-3, Yourdon Press, Englewood Cliffs, 1985.

[Yac04] M.YACOUB. « Conception des systèmes: calculs technico-économiques et aide à la décision dans la conduite d'un projet "recherche » Rapport LAAS No04329 Doctorat, Université Paul Sabatier, Toulouse, 12 juillet 2004.

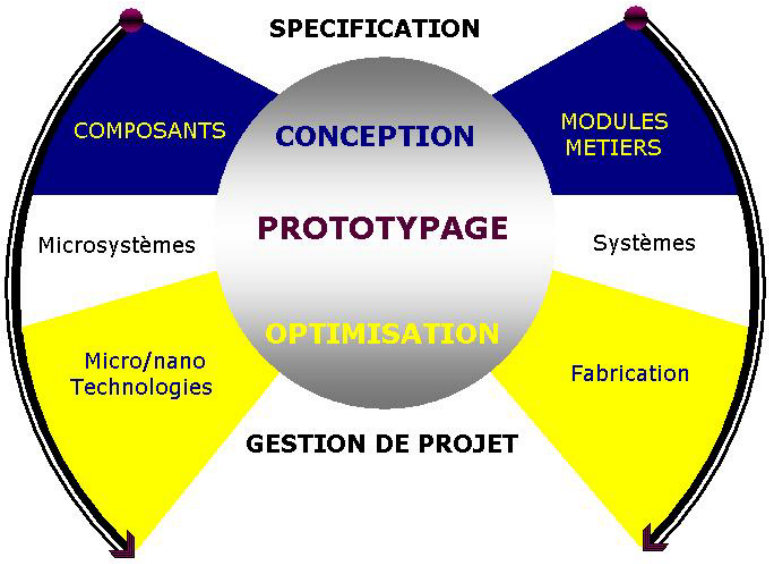
[ZAB+99] Zhou, F.; Archer, N.; Bowles, J.; Clarke, D.; Henry, M.; Peters, C. “A general hardware platform for sensor validation”. Intelligent and Self-Validating Sensors (Ref. No. 1999/160), IEE Colloquium on , 21 juin 1999.

[Zam97] E. ZAMAI, « Architecture de Surveillance – Commande pour les Systèmes a Evénements Discrets Complexes », thèse doctorat, Université Paul Sabatier. Toulouse, France 1997.

[Zho02] S. ZHONG ZHANG. “Creating Safe State Machines”. Mentor Graphics Corporation. Etats Unis 2002.

[ZWI99] Z-World, Inc. “BL 1100. C-Programmable Controller. User’s Manual”. Z-World, Inc. Etats Unis 1999.

ANNEXE A



Méthodes et Outils de la Conception « amont » des Systèmes et Microsystèmes « MOCAS »

Projet Coopératif LAAS

Concernés

Groupe MIS
Groupe OLC
Equipe Système
Service 2I


LAAS


SPECIFICATION
CONCEPTION
PROTOTYPAGE
GESTION DE PROJET

Table de matières


LAAS


SPECIFICATION
CONCEPTION
PROTOTYPAGE
GESTION DE PROJET

1. LES MOTIVATIONS GENERALES

Le projet a son origine dans l'analyse suivante : « Le progrès technologique propose tous les jours davantage de composants et de procédés d'assemblage qui invite à **un art de l'intégration système… ».** Les circuits intégrés sont une pré-figuration historique de cette tendance qui s'illustre aujourd'hui tous les jours par des intégrations hétérogènes de type « systèmes on chip » ou « system on package » et qui émerge de plus en plus au cœur de la conception macro-système à base d'électronique dans tous les secteurs industriels : Informatique, Telecom, Aéronautique et Espace, Automobile, Domotique,…

La proposition générale est donc que le LAAS, de manière concertée, réfléchisse et précise sa stratégie. En l'état actuel de la réflexion conduite au sein par exemple du groupe d'animation TOOLSYS, il semple que les bons objectifs scientifiques pour le LAAS soient :

- dans la modélisation et la conception 'amont' des systèmes conduisant jusqu'au prototypage virtuel (VHDL/AMS),
- dans l'optimisation fondée sur le prototypage virtuel et appliquant des critères très divers de performances, de robustesse, de fiabilité,
- peut-être, dans la conduite de projets…

Le projet propose une approche dans la conception amont fondée sur :

- la mise au point d'un outil de description compatible VHDL/AMS (Hiles design),
- de son couplage avec TINA pour la temporisation des blocs et la validation des procédés,
- de son immersion dans une réflexion systémique concernant l'agrégation des modèles pur la réutilisation et le partitionnement des tâches.

Il applique une démarche de démonstrateur : prendre **des exemples** et les traiter avec cet ensemble d'outils pour en présenter les résultats au collectif LAAS ; c'est dire que ce projet a des objectifs scientifiques et des visées **d'animation dans la réflexion interne** comme les promoteurs en ont déjà fait la démonstration avec TOOLSYS et l'organisation d'une formation VHDL/AMS.

Il souhaite ouvrir deux débats pour le plus long terme :

- celui d'une **plateforme de conception LAAS** associant 2I aux activités technologiques internes : projets en Micro-Nano Technologie, IMPACT, RTB,… et aux activités contractuelles où le LAAS intervient régulièrement en support conception,
- celui de la construction d'un **grand pôle de compétences régional sur la conception et l'intégration systèmes** qui tirerait parti de excellences locales industrielles et académique tournées vers la maîtrise Système : Micro Nano Technologies, Aéronautique et Espace, Automobile, Informatique et Télécommunications…

2. METHODES ET OUTILS DE LA CONCEPTION « AMONT » DES SYSTEMES ET MICROSYSTEMES

La conception Système désigne le secteur de la Recherche-Développement qui vise à prototyper un Système à partir de la description des fonctions qu'il doit réaliser. L'activité historique du LAAS ouvre un large spectre d'études Systèmes : socio-économique, productique, automatique, robotique, informatique, microélectronique et microsystèmes…. Les contributions sont diverses par le développement des méthodes et des outils spécialisés en modélisation, simulation, optimisation, technologie, ou par des études plus globales mettant en œuvre plusieurs de ces méthodes et de ces outils sur un même objectif d'accompagnement industriel ou de Systèmes innovants.

La conception des systèmes et de plus en plus confrontée à une complexité croissante du système à concevoir, à sa pluridisciplinarité incontournable et aux exigences de performances : consommation, robustesse, sécurité,… Elle bénéficie d'efforts considérables consacrés aux développements des méthodes et des outils informatiques visant une plus grande efficacité de conception : rapidité (time to market), qualité et sécurité.

La conception des systèmes, considérée en termes de méthodes et d'outils, fait appel à des compétences très diverses : modélisation, simulation, optimisation,… dans des applications différentes : Instrumentation, commande, traitement du signal, informatique et télécommunication,… supportées par des technologies en évolution rapide vers des miniaturisations ultimes. On peut facilement imaginer qu'une prime à l'efficacité se situe dans la mise en cohérence des différentes contributions. C'est évident, pour le développement industriel, c'est vrai aussi pour la **Recherche qui doit rester au contact des pratiques pour pouvoir faire des propositions utiles en méthodologies, nouveaux outillages et objets innovants…**

Le Département STIC est sensible à cette problématique. Le LAAS en fait un de ses grands objectifs avec une stratégie de rapprochement des compétences fondée sur les acquis internes ou disponibles et sur une démarche pragmatique fondée sur l'exemple[1].

Le lancement des projets LAAS est une excellente occasion d'harmoniser les points de vue et les approches sur la conception des systèmes. On trouvera donc ici des propositions fondées sur la conception effective de systèmes pluridisciplinaires et des propositions plus axées sur les méthodes et les outils de cette conception.

2.1 Les différents domaines de la conception des systèmes

Ce sont 'grosso modo' des domaines historiquement couverts par le LAAS en Automatique, Informatique, Robotique et Micro/Nano Systèmes,… Pour ne pas trop détailler, on trouve dans la démarche classique de la conception systèmes des travaux en conception amont, prototypage virtuel, optimisation, modélisation, technologies de fabrication,…

Ces travaux s'appuient sur l'accès aux modèles et à l'interconnexion de ces modèles au sein d'une représentation globale : le **prototypage virtuel**. Dans la conception système, le langage central VHDL/AMS, a permis le développement d'outils encore incomplets mais déjà diffusés commercialement, sur lesquels le LAAS se positionne en utilisateur. Par contre, le LAAS va être fournisseur de Recherches en apportant à ce nœud central de conception : les modèles, les architectures, les algorithmes (commande, optimisation), les réseaux (communication, énergie), les règles de fabrication et d'assemblage,…

[1] Un inventaire des exemples possibles complètera ce document de travail.

2.2 Le niveau de conception 'amont'

Dans le cadre de la réflexion collective (TOOLSYS), beaucoup d'efforts ont été consacrés à la conception amont en liaison étroite avec des questionnements Microsystèmes et Avioniques (EADS) ; elle part des spécifications textuelles,… du premier niveau de modélisation systèmes, architecture temporisée, pour passer au prototypage virtuel. Dans notre projet, le point de départ de l'outillage de conception amont est le couplage HILeS/TINA (HILeS développé entre MIS et l'ancien groupe OCSD (équipe Système) et TINA développé au sein de OLC).

L'objectif technique de ce projet est de spécifier et développer un outil d'aide à la conception amont des systèmes qui permette de proposer et valider une architecture système (temporisée) compatible avec la simulation VHDL/AMS (figure 1).

On attend donc de cet outillage :

- une aide à la re-écriture de spécifications textuelles,
- une aide à l'établissement de l'**architecture du système** et à sa vérification,
- une aide à la **temporisation logique de cette architecture** et à sa vérification,
- une aide à la **décomposition du système en sous-système définissant des tâches distinctes applicables à la construction d'un plan de travail,**
- une aide à une **capitalisation** et une **ré-utilisation** des acquis de conception,
- Une **porte d'entrée au prototypage virtuel sur VHDL/AMS.**

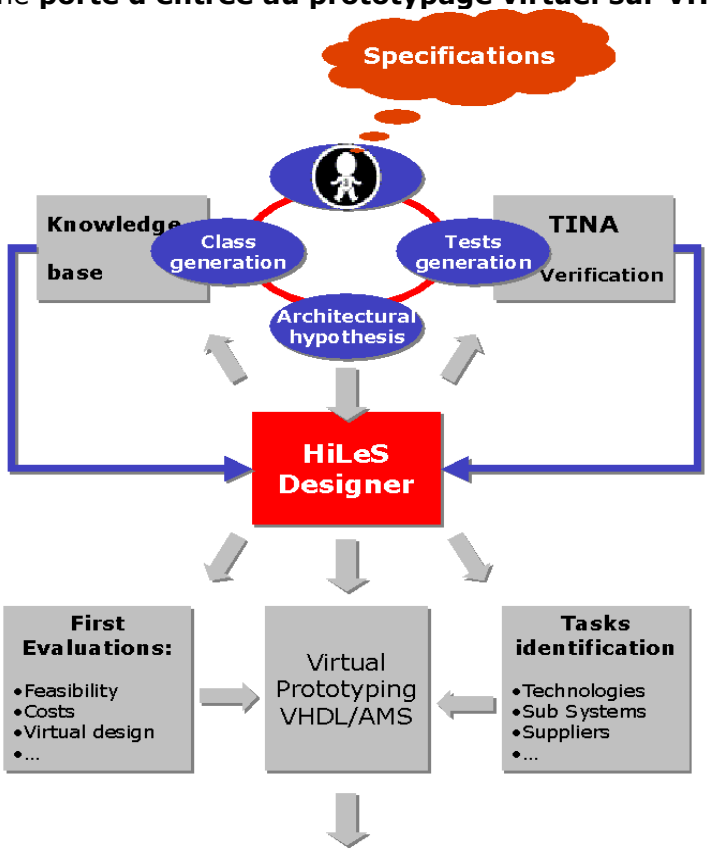


Figure 1

3. LE PROGRAMME DE RECHERCHE

Le Programme de Recherche comporte trois volets :

- **Le traitement d'exemples** portant sur des travaux de Recherche en cours dans les groupes, qui serviront de démonstrateurs et d'appui à la définition de nouvelles étapes de Recherche.

  Les exemples sont présentés de manière sommaire au paragraphe 1. Ils sont au nombre de 3. Ils donneront lieu, pour chacun d'entre eux, à l'application d'une démarche complète de conception 'amont' qui sera décrite de manière détaillée pour montrer son intérêt et donner lieu à des débats scientifiques et techniques.
  Ces exemples sont choisis dans la réalité des besoins exprimés dans des partenariats industriels. Un exemple porte sur un projet LAAS : Systèmes de Vision.

- **Un programme de conception et de développement d'outils 'amont' :** il n'existe pas aujourd'hui d'outil d'aide à la conception partant du **niveau de spécifications jusqu'à la création d'une architecture fonctionnelle temporisée**. De nombreux efforts vont dans ce sens en partant de concepts très ambitieux (PTOLEMEY-Berkeley), de langage de description haut niveau tel UML ou à partir d'extensions haut niveau de langage de simulation : langage C, VHDL, AMS,… et se préparent à proposer des outils : 'system-level design'. L'expérience interne de conception du laboratoire en systèmes électroniques et notamment en microsystèmes par nature pluridisciplinaire, associée à des collaborations étroites avec des secteurs démarcheurs comme l'Aéronautique, l'Espace, l'Automobile, permet de bien définir les besoins.

L'ambition opérationnelle de ce programme est triple :

- **établir des liens** entre des compétences dispersées au sein du laboratoire de manière à bien spécifier le champ de la conception amont et les caractéristiques d'un outil diffusable le cas échéant,
- **développer un outil commun** à partir des acquis du laboratoire et de la sélection d'outils extérieurs compatibles et démonter son intérêt et efficacité, sur des **exemples** précis,
- **approfondir des connaissances** sur la représentation des acquis (modèles physiques, modèles comportementaux, méta modèles,…) et leur réutilisation sur l'applicabilité des réseaux de Pétri aux procédures de validation-vérification et sur l'optimisation des choix technologiques : matériels et logiciels.

Il se fonde sur une vision de la conception basée sur l'analyse des besoins en microsystèmes, avionique et autres systèmes à base d'Electronique. Il est centré sur HILeS Designer, outil graphique déjà en cours de développement coopératif au LAAS, avec l'appui de EADS.

Il comporte un **très important volet d'animation** déjà engagé au sein de TOOLSYS et illustré par la formation VHDL/AMS. Il prévoit de restituer les résultats de Recherche au collectif LAAS, de proposer des réunions de travail ouvertes sur les méthodes et les outils, sur l'intérêt de capitaliser les acquis sur une plateforme interne, sur la perspective de concertation régionale au sein de FERIA, CCIT,…

4. LES TACHES A ACCOMPLIR

Les tâches techniques à accomplir sont décrites sur la figure 2. En résumé, elles sont distribuées comme suit :

- Les interfaces HiLeS avec TINA, la base de connaissances (Knowledge Base), et les simulateurs VHDL/AMS, sont en charge du groupe MIS,
- Les générations de tests et l'activation TINA sont en charge du groupe OLC,
- La création d'un outil de synthèse des acquis pour leur réutilisation (KB) et l'identification (partitionnement) des tâches sont en charge de l'activité système. Ce point est à approfondir : les principes sont décrits dans l'annexe 1.

L'approche choisie est de procéder pas à pas, à partir d'exemples choisis parmi les projets d'applications en cours (AIRBUS, CNES, EDF), soit choisis parmi les projets LAAS axés sur la conception des systèmes (ex : caméras couplées IR et visibles).

4.1 Développement de l'outil *HiLeS Designer* : Contributions MIS et Equipe Système

Notre travail se centre dans le développement et amélioration de l'outil HiLeS Designer et son application dans un exemple fourni par la société Airbus France. Le but est de construire une plate-forme de prototypage virtuel autour de HiLeS Designer, telle qu'elle remplisse les fonctions détaillées sur la figure 2.

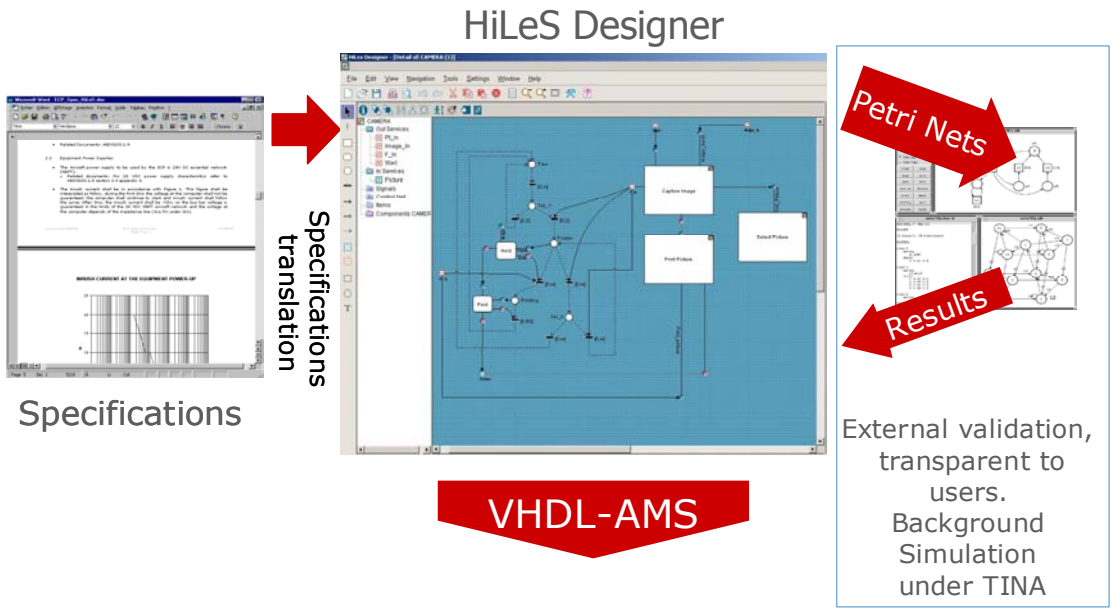


Figure 2*: les fonctions de l'outil Hiles et les contributions de MIS*

Les actions ponctuelles à exécuter sont les suivantes :

1. de vérifier les bonnes fonctionnalités de l'outil HiLeS designer dans un contexte de laboratoire et industriel réel par le biais du calculateur ECP (EADS), de modéliser

de façon comportementale et hiérarchique le calculateur en détaillent les aspects suivants :
   - la source d'alimentation
   - les modules de communication ARINC 429
   - le clavier
   - la procédure de mise en tension
2. De développer une interface entre HiLeS Designer et TINA, permettant de vérifier la structure du modèle écrite à base des réseaux de Petri.
3. De développer une interface entre HiLeS Designer et le langage VHDL-AMS permettant de simuler la totalité de l'architecture et de proposer une première représentation du prototype virtuel.

4.2 Validation des diagrammes HiLeS : Contribution OLC

Les diagrammes Hiles sont interprétés comme des réseaux de Petri temporels associés à des blocs fonctionnels. Les réseaux de Petri temporels expriment à la fois une information de contrôle et des contraintes temporelles régissant le comportement du système spécifié. La contribution de OLC concerne la validation de ces diagrammes. Valider un diagramme Hiles est assurer la satisfaction d'un certain nombre de propriétés de l'application en cours de conception, énoncées par l'utilisateur. Ces propriétés peuvent être de nature générale ou spécifique, et ces dernières de nature qualitative ou quantitative. Pour vérifier ces propriétés, il faut dans un premier temps disposer d'une représentation du comportement de l'application modélisée. Les diagrammes Hiles peuvent automatiquement être traduits en réseau de Petri temporels. L'outil Tina permet ensuite de construire une représentation du graphe des états accessibles du réseau (et donc du système). La validation s'exercera sur cette structure. Elle peut tout d'abord concerner des propriétés générales d'accessibilité telles que : nombre fini d'états de contrôle, absence de blocage, etc. En pratique, la validation de ces seules propriétés générales d'accessibilité n'est pas suffisante pour assurer qu'un système spécifique satisfait un service particulier. On peut ainsi vouloir exprimer et vérifier que l'accès à une ressource se fait en exclusion mutuelle, que le temps d'utilisation d'une ressource est borné par k unités de temps, ou encore que toute requête d'accès à cette ressource sera inévitablement honorée au bout d'un temps fini ou en moins de k unités de temps. Pour exprimer ces propriétés, on a recours à des formalismes dédiés tels que les logiques temporelles ou les équivalences de comportement. Ces formalismes permettent de spécifier des propriétés spécifiques mettant en jeu des états partiels du système ou des événements particuliers. Ces propriétés peuvent aussi mettre en jeu des contraintes temporelles. Les techniques classiques de model-checking permettent de vérifier qu'un système dont le comportement est représenté par un graphe des états accessibles tel que produit par Tina satisfait un ensemble de propriétés spécifiques telles qu'exprimées en logique temporelle.

Toutefois, dans le contexte du projet, il ne peut être supposé que l'utilisateur est un expert en logiques temporelles, pas plus qu'en techniques d'analyse des réseaux de Petri temporels. Bien que devant être basée sur ces aspects formels, la démarche de vérification souhaitable devra en masquer les détails techniques, et doit être exprimée dans le langage de l'utilisateur. Ainsi, plutôt que de manipuler des formules de logique temporelle, **l'utilisateur aura à sa disposition un certain nombre de propriétés spécifiques "métier" prédéfinies. Ces propriétés seront automatiquement traduites en propriétés de l'espace d'état du réseau de Petri temporel capturant le comportement du système, exprimées dans une logique formelle adéquate.**

Une vérification de modèle, choisi parmi les vérificateurs existants, ou adapté, assurera la vérification de ces propriétés. En cas d'échec, le diagnostic formel (contre-exemple) devra être expliqué à l'utilisateur en termes concrets (relatifs au diagramme Hiles),

plutôt qu' en termes formels (violation d'une propriété de logique temporelle par le comportement du diagramme).

Cette démarche est illustrée dans la Figure ci-dessous, qui fait apparaître les niveaux utilisateur (Hiles, étendu par un langage d'expression de propriétés), et vérificateur (Traduction de spécifications et propriétés Hiles, vérification, et diagnostic). A ce jour, seule la partie traduction de diagrammes Hiles en réseau de Petri temporel, et génération du comportement sont effectives, les comportements sont générés par l'outil Tina. Le groupe OLC contribuera à :

- la spécification, avec les utilisateurs, d'un langage de propriétés "métier" pour Hiles;
- la traduction de ces propriétés en formalismes logiques;
- l'intégration d'un outil de vérification pour ces propriétés;
- la traduction inverse des diagnostics de vérification vers le langage de propriétés Hiles.

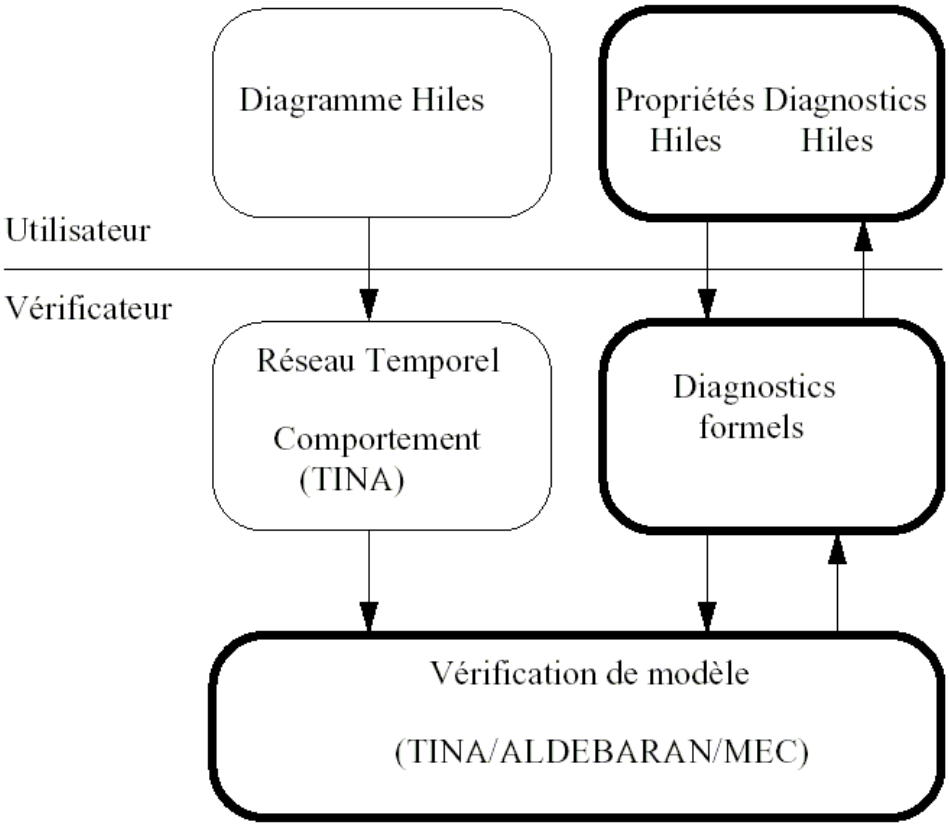


Figure 3 En gras les aspects de la contribution du groupe OLC

5. EXEMPLES D'APPLICATION

Nous traiterons en priorité trios exemples, sachant que le collectif de travail participe dans le cadre contractuel à une application décrite en l'annexe2.

5.1 Les exemples de la conception Microsystème

La conception des Circuits Intégrés a démarré avec l'avènement de la technologie planar en 1959. Dès 1964, le Texas proposait des premiers circuits intégrés bipolaires. A la fin des années 60, on voyait émerger les premiers processeurs… Aujourd'hui, la conception en circuits intégrés numériques gère 10^8 composants par puce. Evidemment, cette évolution s'est aussi appuyée sur le **développement d'outils de CAO et le travail coopératif indispensable pour traiter de telles complexités**.

Le LAAS, dans cette évolution, a été très actif et très utile jusque dans les années 80, par sa contribution à la modélisation et à la technologie des composants innovants Si et GaAs. Mais il n'a pas su s'organiser en conception de circuits intégrés pour développer une activité faisant le pont entre la Microélectronique et les autres domaines du LAAS : Informatique, Robotique, Automatique.

Les Systems on chip, System on packages, Microsystèmes, Intégration de Puissance, Intégration Opto-électronique,… nous donnent une **deuxième chance fondée sur l'intégration des Technologies.**

Nous sommes, à peu près, au stade des années 80 pour la Microélectronique : maîtrise de la modélisation et des technologies composants.

La question est donc : comment mettre en place une méthodologie et des outils communs de conception des Microsystèmes intégrés ou assemblées, comportant des fonctions de gestion d'énergie, de mesure, d'actionnement, de calcul,… ? Pour ce faire, il faut pour le moins :

- **Entretenir et même développer notre culture de Modélisation Physique** en accompagnement de nos développements de dispositif ou de technologie les plus stratégiques.

- **Prendre en compte la Modélisation et la Simulation Microsystèmes globale par le Prototypage Virtuel.** Nos composants s'inscrivent dans un stratège Microsystème et Intégration Microsystème. Il faut donc assurer le point entre le concept composant et l'application. Concevoir consiste alors à associer des modèles représentatifs de tous les constituants : capteurs, traitement des signaux, actionneurs, circuits de commande, interfaces communication et interfaces utilisateur, gestion d'énergie,… pour simuler globalement le système dans ses conditions d'utilisation.

- **Appliquer sur le prototypage virtuel des méthodes et critères d'optimisation** sur les performances, la robustesse, la fiabilité…

Si le point 1 est habituel dans la pratique de notre laboratoire, le point 2 n'a été que très récemment introduit. Le point 3 est identifié depuis quelques années seulement sur l'objectif d'évaluation prédictive de la fiabilité des Microsystèmes sans avoir été encore développé.

Que doit-on attendre d'une action coopérative sur la Conception Microsystème ?
Un outil et une procédure permettant :

- de décrire toutes les fonctions génériques (architecture) qui composent le microsystème, sous forme de graphique compréhensible et partageable par une équipe
- de gérer temporairement ces fonctions,
- de valider architecture et gestion au stade du concept,
- de substituer aux **fonctions génériques le modèle réel** jusqu'à aboutir au prototypage virtuel,
- de simuler le tout (les outils de simulation sont disponibles : PSPICE pour l'analogique, VHDL pour le numérique, VHDL/AMS pour le mixte… et pour le pluridisciplinaire : thermique, mécanique, optique, biochimique,…).

La démarche proposée est de prendre les exemples suivants et les traiter avec des outils : Hiles designs, Tina… disponibles des spécifications au prototypage virtuel… Tirer les conséquences de cette expérience largement partagée pour définir une stratégie concertée.

5.1.1. Micro système « Accéléromètre Lanceur »

Une question majeure de la conception microsystème aujourd'hui est celle de l'évaluation, au stade de la conception, de leur fiabilité. Cette évaluation est possible si l'on dispose d'une modélisation physique des composants incluant les effets de fatigue et plus précisément les effets d'influence des contraintes d'utilisation et d'une modélisation globale pour calculer les conséquences des évolutions des caractéristiques des composants sur les performances d'ensemble.

Cette question nous est posée dans le cadre de la conception d'un microsystème « accéléromètre lanceur » dont la mission est de mesurer les vibrations du lanceur Ariane pendant la phase de décollage. Les spécifications générales du microsystème sont :

- Accélérations : ± 2g
- Energie électrique par pile embarquée
- Procédure de test télécommandé
- Enregistrement durant vol
- Renvoi des données avant destruction

Notre plan de travail comporte les étapes suivantes :

1. L'écriture des spécifications détaillées

2. La conception « amont » des microsystèmes avec deux objectifs :
   i. Introduire la physique du fonctionnement du capteur capacitif pour pouvoir associer de la physique des défaillances
   ii. Représentation fonctionnelle de tout le système par Hiles Designer

3. Etablir un prototype virtuel de l'ensemble pour le choix des composants COTS et le développement d'une modélisation détaillé en VHDL-AMS (Hiles est compatible avec VHDL-AMS)

4. Appliquer sur le prototype virtuel des procédures d'optimisation en particulier des calculs de fiabilité : optimisation des choix, de l'architecture et des modes de fonctionnement pour assurer le bon fonctionnement de l'expérience dans des conditions extrêmes.

5. Illustrer le bien-fondé de l'expérience par une réalisation exemplaire.

Notre participation au projet LAAS : Etablir une modélisation amont du microsystème lanceur sous HiLes et explorer les intérêts et insuffisances de l'outil aux interfaces Hiles/TINA/VHDL-AMS.

5.1.2. Exemple d'un microsystème multi-mesures, autonome et Communicant LAAS/EDF R&D

Ce projet de conception système a démarré en Octobre 2002 en collaboration avec EDF R&D. Son objectif de démonstration est la conception et la réalisation au stade d'une maquette d'un dispositif multi-mesures, autonome et communicant dont l'architecture d'intégration microsystèmes sera suffisamment générique pour des réutilisations ultérieures. Le but pour cette première application est de concevoir un maillage de microsystèmes répartis sur les parois de béton d'une centrale EDF (distance d'un mètre environ) pour avoir une cartographie des paramètres mesurés. Les microsystèmes devront être capables en particulier de mesurer la déformation de la paroi de béton et d'enregistrer des paramètres d'environnement tels que l'humidité, la pression et la température.

Le travail se déroule en périodes distinctes :

1- Analyse du besoin / Rédaction d'un cahier des charges et d'un cahier de spécifications / Description des modes opératoires.
2- Conception amont / Prototypage virtuel haut niveau (HiLes-Tina) / Vérification fonctionnelle et séquentielle.
3- Matérialisation virtuelle, insertion de composants COTS dans le modèle / Simulation VHDL-AMS.
4- Test / Modifications du prototype virtuel.
5- Réalisation d'une maquette prototype.

Notre objectif scientifique est de contribuer au développement méthodologique de la conception des microsystèmes et de dégager des solutions technologiques dans l'intégration multi-sensorielles sur silicium et par assemblages hybrides.

Le travail de conception a démarré à très haut niveau, à partir d'une analyse fonctionnelle des besoins. De là nous avons dégagé un cahier des charges fonctionnel et des spécifications techniques du besoin qui ont fait apparaître des blocs fonctionnels. Nous avons ensuite écrit les processus qui décrivent les principaux modes opératoires du système. La finalité de cette étude est d'aboutir à une description virtuelle d'un système complet et simulable. L'outil choisi à cette fin est le logiciel HiLes qui permet de décrire l'architecture du système et de réaliser des simulations fonctionnelles et événementielles associées. Cette étape est partie prenante du projet LAAS sur la conception amont des systèmes. Au delà, nous devrons transférer les résultats de cette étape en VHDL-AMS vers le prototypage virtuel.

Dans cet objectif, nous voulons regrouper les outils et les méthodes nécessaires à chaque étape de conception pour conserver le savoir et proposer une démarche générique de conception pour tous les microsystèmes de même nature.

5.2. La conception système

La Conception Système, telle que nous la considérons ici, ne se différencie pas fondamentalement de la Conception Microsystème sinon que l'on n'a pas, a priori, d'accès direct aux technologies comme c'est le cas en Micro Nano Technologies. On

retrouve les mêmes questions, certaines posées avec plus d'acuité comme par exemple le partage Matériel-Logiciel.

5.3. Exemple d'une plate-forme multivision IR et visible

Cet exemple est partie intégrante d'un autre projet LAAS proposé autour d'une plate-forme associant en stéréovision deux types de caméra IR et visible. L'objectif de la conception amont sera de rédiger les spécifications de cette plate-forme avec tous les partenaires de projet et de le retranscrire en Hiles Designer pour en valider l'architecture et sa temporisation.

C'est une hypothèse de travail très intéressante puisque, dans ce cas, la conception est complètement gérée par le LAAS et le laboratoire pourra apprécier l'effort méthodologique par rapport aux pratiques plus empiriques.

6. PROGRAMME D'INFORMATION ET DE FORMATION

Le financement demandé pour le fonctionnement assurera les dépenses courantes et des actions plus spécifiques de formation et d'information :

- Missions d'information en France, Europe et les Etats Unis, en particulier à Berkeley et Stanford pour bien analyser les fonctionnements de chacun.
- Invitations de spécialistes des méthodes et des outils de conception.
- Recherche et formation :
  - Nouvelle séance sur VHDL-AMS avec la participation du Club des Affilies
  - Présentation et diffusion de HiLeS Designer en version 0, puis en version 1
  - Séminaire sur la conception logicielle
  - Séminaire sur la simulation matériel-logiciel et sur les outils et interfaces à développer.
  - Séminaire sur les bases de données, la capitalisation des acquis et les IP
  - Colloque sur la conception système et microsystème.

7. PLATE-FORME

Le group de travail constitue autour de HiLeS Designer s'efforcera d'étudier la voie de création d'une plate-forme ouverte au niveau interne LAAS et à niveau régional.

8. LA DUREE DE L'ETUDE : 24 MOIS A PARTIR D'OCTOBRE 2003.

Livrable: Un premier prototype opérationnel de l'outil complet de conception amont devrait alors être disponible pour des usagers intéresses et pour des actions contractuelles extérieures.

- L'application de l'outil sur des exemples microsystème et système illustrant la manière de l'utiliser.
- Un document de réflexion sur le perspective ouverte au LAAS en conception système et plate-formes de conception.

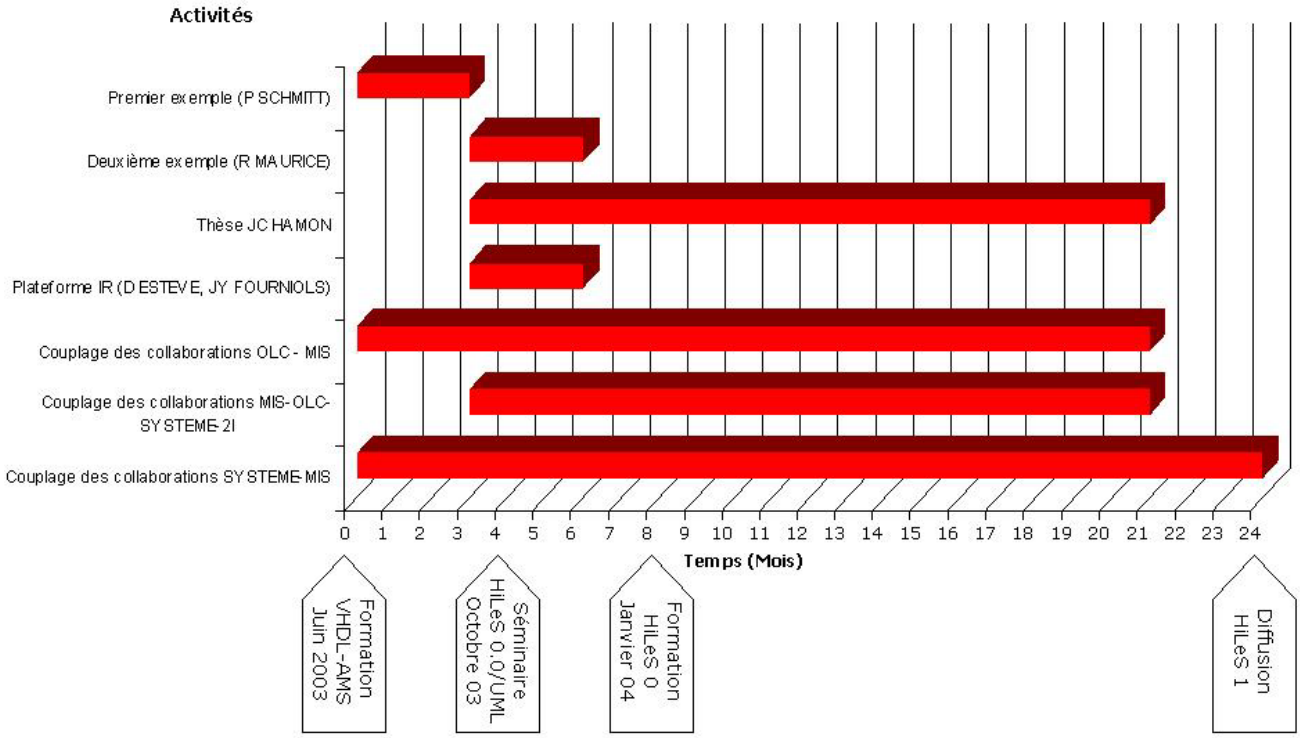


Figure 4. Chronogramme du projet

9. LE COUT DE L'ETUDE

Partant d'exemples concrets, l'objectif de ce travail coopératif est de :

- mettre des acquis en commun,
- forger un outil complet diffusable pour la conception amont,
- Contribuer à une prise de conscience collective et à la construction d'une démarche de conception système.

Le montant de l'étude est de **50.000 €**

Les personnels impliqués (à temps partiel) :

1 correspondant 2I sur la Conception Système

MIS : D. Estève, J.Y. Fourniols, E. Campo, B. Jammes, P. Schmitt, J.C. Hamon, R. Maurice

OLC : B. Berthomieu, F. Vernadat

Equipe ISI : M. Paludetto, P. Esteban, J.C. Pascal

ANNEXE 1 : MODELISATION DE HAUT NIVEAU DANS LA CONCEPTION SYSTEME : CONTRIBUTION EQUIPE SYSTEME

Dans une conception système à haut niveau, deux aspects fondamentaux de la modélisation sont à considérer:

Les besoins fonctionnels qui fixent les fonctions des systèmes et leur architecture. Généralement, leur modélisation matérielle ne pose pas de problème. Chaque métier a choisi sa technique préférée (équations différentielles, automates, réseaux de Petri, VHDL, …) et dispose de sa bibliothèque de composants.

Les besoins dits non fonctionnels (coûts, sûreté de fonctionnement, sécurité, délais, performances, …). On aimerait bien modéliser ces besoins pour les prendre en compte le plus tôt possible dans le processus décisionnel du développement. Dans ce cas, la modélisation est autrement plus difficile qu'elle n'est pas structuré pour l'instant. Cette difficulté est augmentée par les aspects multi-métiers des systèmes hétérogènes.

Le point de rencontre entre ces deux aspects réside dans le fait qu'il est nécessaire de faire cohabiter des modèles différents produits par des métiers différents. Sans préjuger d'une solution miracle, l'approche "Système" doit de faire selon un processus descendant/ascendant qui met un accent fort sur:

- L'expression des besoins et des exigences fonctionnelles et non fonctionnelles dans la rédaction des spécifications.
- La modélisation/simulation système adaptée aux exigences de convivialité pluridisciplinaire et multiutilisateur.
- La traduction des modèles métiers clients en des modèles de haut niveau qui alimenteront la base de connaissance pour l'Ingénierie Système (Bottom-Up).
- La traduction des modèles des concepteurs de haut niveau en des modèles génériques qui alimenteront la base de connaissance pour l'Ingénierie Système (Top Down).
- Le mécanisme décisionnel dans le partitionnement des tâches, le choix des technologies et le scenarii de fabrication.

Ceci peut être résumé dans le schéma ci-dessous, **Figure 5**.

L'objectif au sein du projet et double :

1. A partir de la modélisation amont globale (fourni par HiLeS Designer et éventuellement par le prototypage virtuel) et de critères de mise en œuvre, de proposer le partitionnement des tâches « métiers » sous la forme de plusieurs scenarii et si possible de rechercher dans l'intégration des besoins non fonctionnels, la hiérarchisation de ces scenarii.
2. A partir du retour des tâches « métier », des travaux de partitionnement évoqués en 1, du travail amont de conception amont (ingénierie de système), de traduire les acquis en modèles génériques et de les capitaliser dans une Base de Connaissances facilitant leur réutilisation (reuse).

Ces deux objectifs seront abordes dans le cadre des exemples du projet commun de manière à dégager une stratégie de recherche centrée sur la modélisation de mise en œuvre et de capitalisation des connaissances dans une démarche globale de conception-fabrication système.

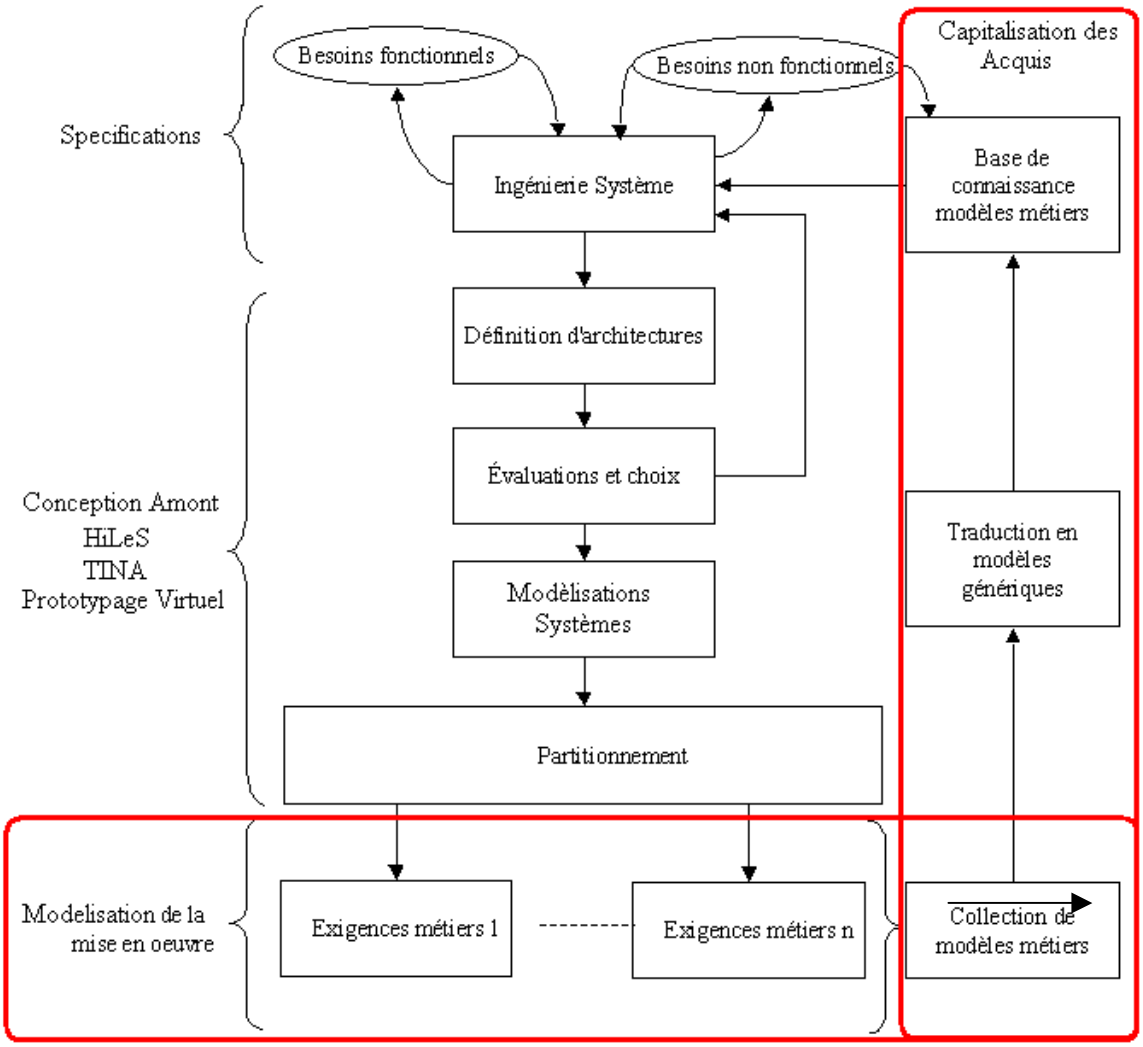


Figure 5. En rouge les secteurs de contribution de l'équipe système.

ANNEXE 2 : EXEMPLE D'UN PANNEAU ELECTRONIQUE CENTRALISE DE CONTROLE ET SURVEILLANCE D'AVION

Les avions d'aujourd'hui sont équipés de systèmes d'alarmes en vol qui permettent à l'équipage de surveiller le comportement de tout l'équipement et de détecter des situations anormales, ces systèmes sont connues sous l'acronyme FWS (« Flight Warning Systems »). Dans de tels systèmes d'avertissement de vol, des fonctions de l'interface homme - machine (HMI) sont exécutées par un panneau de contrôle centralisé qui permettre de conduire les écrans, d'accepter les messages d'alarme et de rechercher des informations spécifiques de l'appareil. C'est donc le ECP.

Le panneau de contrôle ECAM (ECP) est un sous-ensemble du FWS qui fournit à l'équipage un HMI au contrôle le FWS et notamment pour le système le contrôle de visualisation (CD). Le système d'avertissement de vol (FWS) est considéré comme un système centralisé de l'avionique dédié aux fonctions suivantes :

- Contrôle continu des principaux systèmes de bord avec :
  - affichage automatique ou manuel des messages du système,
  - Génération immédiate d'alertes vers l'équipage, en cas de fautes,
  - défaillances ou de configuration dangereuse des systèmes de bord,
  - affichage des listes de contrôle et de vérification,
  - de l'information additionnelle ou des procédures supplémentaires à l'équipage, selon la phase de vol ou sur demande, des alertes sont présentées à l'équipage en utilisant des signaux auditifs et visuels.

Le calculateur ECP permet à l'équipage de choisir manuellement les diverses pages du FWS sur l'unité de visualisation du CD.

Le choix des fonctions est fait par un ensemble de touches retro-éclairés. Ces touches retro éclairés permettront de choisir et de contrôler les messages d'alertes et l'affichage normal de procédures de vol, les listes de contrôle et de vérification, l'affichage de la page générale de statut, l'affichage d'une page d'information particulier du système et de réaliser manuellement la configuration de décollage.

Le calculateur ECP comporte un module d'entrée et un autre de sortie ARINC 429 pour se communiquer avec les autres dispositifs du FWS. Néanmoins, en parallèle, quelques commandes seront câblées directement vers l'extérieur pour assurer la disponibilité des fonctions principales en cas d'échec d'ECP. Le clavier aura 26 + 4 touches, 19 parmi elles avec retro-éclairage. L'alimentation d'énergie du ECP est reliée au réseau d'avion de C.C de 28 V et comporte trois sorties à 15V, -15V et +5V

Le ECP aura une fonction de surveillance intégrée, le défaut de BITE (équipement de test intégré). De la même façon, une fonction de remise automatique du logiciel doit être mise en place.

Nous disposons des spécifications détaillées de ce projet et le premier objectif est de les retranscrire en termes d'architecture et d'architecture temporisée.

En pratique, nous allons introduire l'outil HiLeS Designer, pouvant servir de support de collaboration à haut-niveau de manière à explorer la possibilité d'établir différents scenarii de partage de tâches, notamment : Hardware/Software

L'écriture de scenarii doit pouvoir s'appuyer sur une procédure de classification que sera développée dans ce projet coopératif.

ANNEXE B

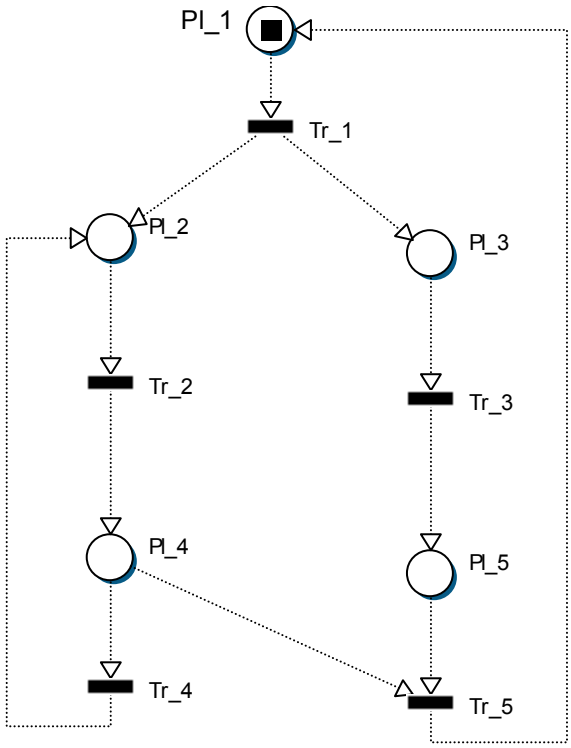


Figure 1. Réseau de Petri à traduire

Le réseau de Petri de la figure 1 fut dessine avec l'éditeur graphique de HiLeS Designer 0v7. Le code VHDL à été généré automatiquement par le module de conversion VHDL-AMS de HiLes Designer. Nous pressentons ci dessous le code VHDL produit par l'outil et une capture d'écran de la simulation du modèle produit réalisé sous SystemVision de Mentor Graphics. Tous les fichiers VHDL-AMS d'un projet HiLeS sont stockés dans le dossier VHDL_Folder crée automatiquement par l'outil lors de la création du projet.

1. Composant PLACE

Le composant *PLACE* fait partie de la bibliothèque de base d'un projet HiLeS.

```
ibrary ieee;
use ieee.std_logic_1164.all;
```

```
entity Place_Async is
generic (       nb_entrees    : natural:= 1;
                        nb_sorties     : natural:= 1;
t:time:= 2 us);
port    ( init   : in std_logic;
          ini_jetons    : in std_logic;
          aj_jetons     : in std_logic_vector(nb_entrees-1 downto 0);
          ret_jetons    : in std_logic_vector(nb_sorties-1 downto 0);
          marque                : out std_logic  );
end Place_Async;

architecture a_place of Place_Async is

signal marquage : std_logic;

begin
process
        variable sum_aj : std_logic;
        variable sum_ret : std_logic;
        variable tmp : std_logic;
        begin
                wait on init,aj_jetons, ret_jetons;
                if (init = '1') then marquage <= ini_jetons;
                        else
                sum_aj:='0';
                sum_ret:='0';

                for i in 0 to nb_entrees-1 loop
                        tmp:=aj_jetons(i);
                        sum_aj:=sum_aj or tmp;
                end loop; -- i

                for j in 0 to nb_sorties-1 loop
                        tmp:=ret_jetons(j);
                        sum_ret :=sum_ret or tmp ;
                end loop; -- j
                marquage <= ((marquage and (not sum_ret)) or sum_aj);
                end if;
end process;

        process
        begin
                wait on marquage;
                marque<=marquage after t ;
        end process;
end a_place;
```

2. Composant TRANSITION

Le composant *TRANSITION* fait partie de la bibliothèque de base d'un projet HiLeS.

```
library ieee;
use ieee.std_logic_1164.all;

entity Transition is
```

```
        generic (  nb_entrees        : natural:=2;
                   nb_sorties : natural:=2  );
        port ( c_t              : in  std_logic;
        type_transit_e          : in  std_logic_vector((nb_entrees-1) downto 0);
        type_transit_s          : in  std_logic_vector((nb_entrees-1) downto 0);
        marque_tie              : in  std_logic_vector((nb_entrees-1) downto 0);
        ret_amont               : out std_logic_vector((nb_entrees-1) downto 0);
        aj_aval                 : out std_logic_vector(nb_sorties-1 downto 0));
end Transition;

architecture a_transition of Transition is
signal sensibilisation : std_logic;

begin
        process (marque_tie,type_transit_e,type_transit_s)
                variable sensible : std_logic;
                variable sens_tr0,sens_tr1  : std_logic;
                begin
                        sensible:='1';
                        for k in 0 to nb_entrees-1 loop
                                if type_transit_e(k)='0' then sens_tr0 :='1';
                                        sens_tr1 :='0';
                                elsif type_transit_s(k)='0' then sens_tr0 :='1';
                                         sens_tr1 :='0';
                                elsif type_transit_s(k)='1' then sens_tr0 :='0';
                                        sens_tr1 :='1';
                                else sens_tr0 :='0';
                                        sens_tr1 :='0';
                                end if;
                                sensible := sensible and ((sens_tr0 and marque_tie(k)) or
                                (sens_tr1 and (not marque_tie(k))));
                        end loop; -- k
                        sensibilisation <= sensible;
                end process;

        process(sensibilisation, type_transit_e,c_t)
                variable sens_tr : std_logic;
                begin
                        for i in 0 to (nb_entrees-1) loop
                                if type_transit_e(i)='0' then sens_tr := (sensibilisation and
                                c_t);
                                else sens_tr :='0';
                                end if;
                                if sens_tr='1' then     ret_amont(i) <= '1';
                                else ret_amont(i) <= '0';
                                end if;
                        end loop;


                        for i in 0 to (nb_sorties-1) loop
                                if ((sensibilisation='1')  and (c_t='1')) then aj_aval(i) <= '1';
                                else aj_aval(i) <= '0';
                                end if;
                        end loop;
        end process;

end a_transition;
```

3. Réseau équivalent en VHDL

La topologie du réseau est reproduite par l'outil HiLeS Designer en utilisant les composants VHDL *PLACE* et *TRANSITION*.

```
library ieee;
use ieee.std_logic_1164.all;

entity testRDP is
     port(init     : in std_logic;
          cond_ti : in std_logic_vector(5 downto 1);
          imarque : out std_logic_vector(5 downto 1));
end testRDP;

architecture main of testRDP is

component Place_Async
generic(   nb_entrees   : natural:= 1;
           nb_sorties   : natural:= 1);
port  (init   : in std_logic;
       ini_jetons   : in std_logic;
       aj_jetons   : in std_logic_vector(nb_entrees-1 downto 0);
       ret_jetons   : in std_logic_vector(nb_sorties-1 downto 0);
       marque   : out std_logic);
end component;

component Transition
generic(   nb_entrees   : natural:= 1;
           nb_sorties   : natural:= 1);
          -- ta   : time:= 2 us);
port  (c_t   : in std_logic;
       type_transit_e   : in std_logic_vector(nb_entrees-1 downto 0);
       type_transit_s   : in std_logic_vector(nb_entrees-1 downto 0);
       marque_tie   : in std_logic_vector(nb_entrees-1 downto 0);
       ret_amont   : out std_logic_vector(nb_entrees-1 downto 0);
       aj_aval   : out std_logic_vector(nb_sorties-1 downto 0));
end component;

-- Signals Pl_1
Signal init_marq_Pl_1     : std_logic;
Signal marq_Pl_1          : std_logic;
Signal add_token_Pl_1      : std_logic_vector( 0 downto 0);
Signal remove_token_Pl_1  : std_logic_vector( 0 downto 0);

-- Signals Pl_2
Signal init_marq_Pl_2     : std_logic;
Signal marq_Pl_2          : std_logic;
Signal add_token_Pl_2      : std_logic_vector( 1 downto 0);
Signal remove_token_Pl_2  : std_logic_vector( 0 downto 0);

-- Signals Pl_3
Signal init_marq_Pl_3     : std_logic;
Signal marq_Pl_3          : std_logic;
Signal add_token_Pl_3      : std_logic_vector( 0 downto 0);
```

```
Signal remove_token_Pl_3  : std_logic_vector( 0 downto 0);

-- Signals Pl_4
Signal init_marq_Pl_4     : std_logic;
Signal marq_Pl_4          : std_logic;
Signal add_token_Pl_4      : std_logic_vector( 0 downto 0);
Signal remove_token_Pl_4  : std_logic_vector( 1 downto 0);

-- Signals Pl_5
Signal init_marq_Pl_5     : std_logic;
Signal marq_Pl_5          : std_logic;
Signal add_token_Pl_5      : std_logic_vector( 0 downto 0);
Signal remove_token_Pl_5  : std_logic_vector( 0 downto 0);

-- Signals Tr_1
Signal type_arc_e_Tr_1      : std_logic_vector( 0 downto 0);
Signal type_arc_s_Tr_1      : std_logic_vector( 0 downto 0);
Signal c_Tr_1     : std_logic;
Signal marq_pe_Tr_1      : std_logic_vector( 0 downto 0);
Signal add_aval_Tr_1      : std_logic_vector( 1 downto 0);
Signal ret_amont_Tr_1      : std_logic_vector( 0 downto 0);

-- Signals Tr_2
Signal type_arc_e_Tr_2      : std_logic_vector( 0 downto 0);
Signal type_arc_s_Tr_2      : std_logic_vector( 0 downto 0);
Signal c_Tr_2     : std_logic;
Signal marq_pe_Tr_2      : std_logic_vector( 0 downto 0);
Signal add_aval_Tr_2      : std_logic_vector( 0 downto 0);
Signal ret_amont_Tr_2      : std_logic_vector( 0 downto 0);

-- Signals Tr_3
Signal type_arc_e_Tr_3      : std_logic_vector( 0 downto 0);
Signal type_arc_s_Tr_3      : std_logic_vector( 0 downto 0);
Signal c_Tr_3     : std_logic;
Signal marq_pe_Tr_3      : std_logic_vector( 0 downto 0);
Signal add_aval_Tr_3      : std_logic_vector( 0 downto 0);
Signal ret_amont_Tr_3      : std_logic_vector( 0 downto 0);

-- Signals Tr_4
Signal type_arc_e_Tr_4      : std_logic_vector( 0 downto 0);
Signal type_arc_s_Tr_4      : std_logic_vector( 0 downto 0);
Signal c_Tr_4     : std_logic;
Signal marq_pe_Tr_4      : std_logic_vector( 0 downto 0);
Signal add_aval_Tr_4      : std_logic_vector( 0 downto 0);
Signal ret_amont_Tr_4      : std_logic_vector( 0 downto 0);

-- Signals Tr_5
Signal type_arc_e_Tr_5      : std_logic_vector( 1 downto 0);
Signal type_arc_s_Tr_5      : std_logic_vector( 1 downto 0);
Signal c_Tr_5     : std_logic;
Signal marq_pe_Tr_5      : std_logic_vector( 1 downto 0);
Signal add_aval_Tr_5      : std_logic_vector( 0 downto 0);
Signal ret_amont_Tr_5      : std_logic_vector( 1 downto 0);

begin

imarque(1) <= marq_Pl_1;
```

```
imarque(2) <= marq_Pl_2;
imarque(3) <= marq_Pl_3;
imarque(4) <= marq_Pl_4;
imarque(5) <= marq_Pl_5;

c_Tr_1 <= cond_ti(1);
c_Tr_2 <= cond_ti(2);
c_Tr_3 <= cond_ti(3);
c_Tr_4 <= cond_ti(4);
c_Tr_5 <= cond_ti(5);

Pl_1 :place_async       generic map( 1, 1)
       port    map    (init,    init_marq_Pl_1,    add_token_Pl_1,    remove_token_Pl_1,
       marq_Pl_1);
       init_marq_Pl_1 <= '1';
       add_token_Pl_1( 0)    <= add_aval_Tr_5( 0);
       remove_token_Pl_1( 0)    <= ret_amont_Tr_1( 0);

Pl_2 :place_async  generic map( 2, 1)
       port    map    (init,    init_marq_Pl_2,    add_token_Pl_2,    remove_token_Pl_2,
       marq_Pl_2);
       init_marq_Pl_2 <= '0';
       add_token_Pl_2( 0)    <= add_aval_Tr_1( 0);
       add_token_Pl_2( 1)    <= add_aval_Tr_4( 0);
       remove_token_Pl_2( 0)    <= ret_amont_Tr_2( 0);

Pl_3 :place_async  generic map( 1, 1)
       port    map    (init,    init_marq_Pl_3,    add_token_Pl_3,    remove_token_Pl_3,
       marq_Pl_3);
       init_marq_Pl_3 <= '0';
       add_token_Pl_3( 0)    <= add_aval_Tr_1( 1);
       remove_token_Pl_3( 0)    <= ret_amont_Tr_3( 0);

Pl_4 :place_async  generic map( 1, 2)
       port    map    (init,    init_marq_Pl_4,    add_token_Pl_4,    remove_token_Pl_4,
       marq_Pl_4);
       init_marq_Pl_4 <= '0';
       add_token_Pl_4( 0)    <= add_aval_Tr_2( 0);
       remove_token_Pl_4( 0)    <= ret_amont_Tr_4( 0);
       remove_token_Pl_4( 1)    <= ret_amont_Tr_5( 1);

Pl_5 :place_async  generic map( 1, 1)
       port    map    (init,    init_marq_Pl_5,    add_token_Pl_5,    remove_token_Pl_5,
       marq_Pl_5);
       init_marq_Pl_5 <= '0';
       add_token_Pl_5( 0)    <= add_aval_Tr_3( 0);
       remove_token_Pl_5( 0)    <= ret_amont_Tr_5( 0);

Tr_1 :transition     generic map( 1, 2)
       port map (c_Tr_1, type_arc_e_Tr_1, type_arc_s_Tr_1, marq_pe_Tr_1,
       ret_amont_Tr_1, add_aval_Tr_1);
       type_arc_e_Tr_1( 0)   <= '0';
       type_arc_s_Tr_1( 0)   <= '0';
       marq_pe_Tr_1( 0)    <= marq_Pl_1;

Tr_2 :transition     generic map( 1, 1)
       port map (c_Tr_2, type_arc_e_Tr_2, type_arc_s_Tr_2, marq_pe_Tr_2,
```

```
        ret_amont_Tr_2, add_aval_Tr_2);
        type_arc_e_Tr_2( 0)   <= '0';
        type_arc_s_Tr_2( 0)   <= '0';
        marq_pe_Tr_2( 0)    <= marq_Pl_2;

Tr_3 :transition     generic map( 1, 1)
        port map (c_Tr_3, type_arc_e_Tr_3, type_arc_s_Tr_3, marq_pe_Tr_3,
        ret_amont_Tr_3, add_aval_Tr_3);
        type_arc_e_Tr_3( 0)   <= '0';
        type_arc_s_Tr_3( 0)   <= '0';
        marq_pe_Tr_3( 0)    <= marq_Pl_3;

Tr_4 :transition     generic map( 1, 1)
        port map (c_Tr_4, type_arc_e_Tr_4, type_arc_s_Tr_4, marq_pe_Tr_4,
        ret_amont_Tr_4, add_aval_Tr_4);
        type_arc_e_Tr_4( 0)   <= '0';
        type_arc_s_Tr_4( 0)   <= '0';
        marq_pe_Tr_4( 0)    <= marq_Pl_4;
Tr_5 :transition     generic map( 2, 1)
        port map (c_Tr_5, type_arc_e_Tr_5, type_arc_s_Tr_5, marq_pe_Tr_5,
        ret_amont_Tr_5, add_aval_Tr_5);
        type_arc_e_Tr_5( 0)   <= '0';
        type_arc_s_Tr_5( 0)   <= '0';
        type_arc_e_Tr_5( 1)   <= '0';
        type_arc_s_Tr_5( 1)   <= '0';
        marq_pe_Tr_5( 0)    <= marq_Pl_4;
        marq_pe_Tr_5( 1)    <= marq_Pl_5;
end main;
```

4. Testbench

Le testbench est produit aussi par HiLeS Designer.

```
library ieee;
use ieee.std_logic_1164.all;

entity simulation_test is
end;
architecture main of simulation_test is

component testRDP
        port(init     : in std_logic;
        cond_ti : in std_logic_vector(5 downto 1);
        imarque : out std_logic_vector(5 downto 1));
end component;

signal i_init: std_logic:='1';
signal cd: std_logic_vector(5 downto 1);
signal iimarque: std_logic_vector(5 downto 1);

begin
        cd<="11111";
        simulation_run : testRDP
        port map (init=>i_init, cond_ti=>cd, imarque=>iimarque);

        inicialization:process(i_init)
        begin
```

```
        i_init<='0' after 1 us;
        end process;
end;
```

5. Simulation sous SystemVision

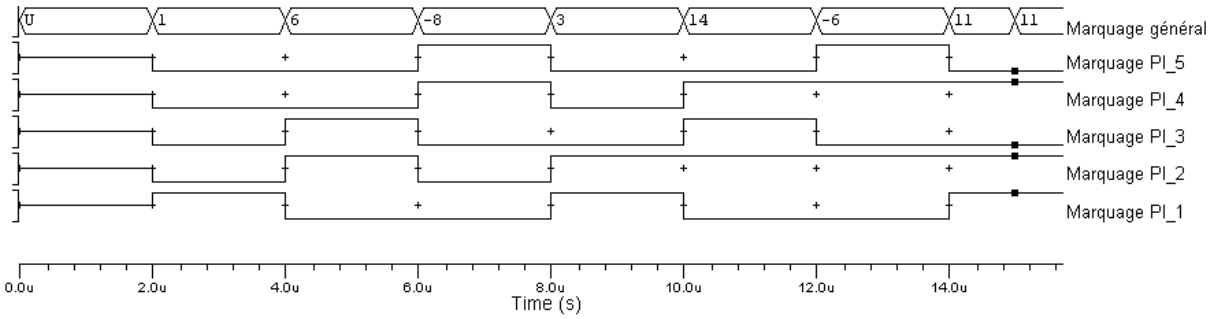


Figure 2. Simulation sous SystemVision du modèle obtenu.