Ingénierie système guidée par les modèles: Application du standard IEEE 15288, de l'architecture MDA et du langage SysML à la conception des systèmes mécatroniques

Skander Turki

THÈSE

Présentée
pour obtenir le titre de

Docteur de L'Université du Sud Toulon-Var
Spécialité : Sciences et techniques industrielles

Par

Skander TURKI

Ingénieur informaticien de la faculté des sciences de Tunis
DEA MCAO Aix-Marseille III

Ingénierie système guidée par les modèles: Application du standard IEEE 15288, de l'architecture MDA et du langage SysML à la conception des systèmes mécatroniques

---

Sous la direction de : M. Thierry Soriano

Soutenue le 02 Octobre 2008 devant le jury composé de :

Jacques LOTTIN	Polytech'Savoie Annecy	Président
Pierre de SAQUI-SANNES	ISAE Toulouse	Rapporteur
Hubert KADIMA	EISTI Cergy Pontoise	Rapporteur
Jean Philippe DIGUET	UBS Lorient	Examinateur
Yves LACROIX	USTV Toulon	Examinateur
Thierry SORIANO	SUPMECA Toulon	Directeur de thèse

Thèse préparée au Laboratoire LISMMA (EA 2336) Supmeca Toulon
Dans l'équipe : Ingénierie intégrée des systèmes industriels et mécatroniques.

Avant propos

Les travaux présentés dans ce mémoire ont été réalisés au Laboratoire d'Ingénierie des Systèmes Mécaniques et des MAtériaux (LISMMA) à l'institut SUPérieur de MECAnique de Toulon (SUPMECA).
Je voudrais remercier toutes les personnes qui m'ont soutenu, de près et de loin, en particulier:
– *Monsieur* Thierry SORIANO, Maître de conférences HDR au LISMMA pour m'avoir accueilli dans son laboratoire et soutenu tout au long de la thèse, qui a consacré son temps pour encadrer ces travaux. Merci pour ses conseils scientifiques, l'encouragement et la confiance qu'il m'a accordés.
– *Monsieur* Jacques LOTTIN, Professeur au Laboratoire SYMM à Polytech' Savoie, pour avoir bien voulu présider ce jury.
– *Monsieur* Pierre de SAQUI-SANNES, Professeur à l'Institut Supérieur de L'Aéronautique et de L'Espace (ISAE) pour l'honneur qu'il m'a fait en acceptant d'être rapporteur de ce mémoire. Je le remercie pour les remarques très pertinentes apportées à cette thèse.
– *Monsieur* Hubert KADIMA, Professeur HdR à l'Ecole Internationale des Sciences du Traitement de l'Information EISTI, pour avoir accepté d'être rapporteur de cette thèse et pour les conseils très intéressants qu'il m'a adressés.
– *Monsieur* Jean Philippe DIGUET, chargé de recherche HDR au Lab-STICC / CNRS à l'Université de Bretagne Sud, pour avoir accepté de me faire l'honneur de faire partie du jury.
– *Monsieur* Yves LACROIX, Professeur au Laboratoire Systèmes Navals Complexes à L'Institut des Sciences de l'Ingénieur de Toulon et du Var (ISITV), pour l'honneur qu'il m'a fait en acceptant de participer à ce jury.
– *Monsieur* Adel SGHAIER docteur de l'Université du Sud Toulon-Var et *Monsieur* Hassan HADJ AMOR doctorant au LISMMA à SUPMECA Toulon pour leur soutien moral et les discussions scientifiques qui m'ont permis de découvrir des sujets de recherches variés.
– *Monsieur* Alain RIVIERE, *Monsieur* Jean-Yves CHOLEY et *Monsieur* Régis PLATEAUX du LISMMA à SUPMECA Paris pour les discussions scientifiques et les échanges qui nous ont permis de mieux appréhender le sujet de la thèse.

Je tiens à remercier très sincèrement le personnel administratif et scientifique du LISMMA *Madame* Pascale AZOU-BRIARD, *Monsieur* Patrice CHOLLET, *Monsieur* Dominique MILLET, *Monsieur* Marc BRIARD, *Monsieur* Olivier TORREMOCHA, *Monsieur* Christian TOUSSAINT, *Madame* Lucette ARCHER, *Madame* Ingrid DESCAREGA, *Madame* Sabine SEILLIER, *Madame* Sabine SEGAL, *Monsieur* Christian PAILLER pour leurs compétences, leur gentillesse et leur disponibilité.

Je remercie l'ambassade de France en Tunisie qui, dans le cadre de la coopération franco-tunisienne, a bien accepté de financer cette thèse.

Je remercie également toute la famille, d'abord ma mère et mon père pour leur support inconditionnel, puis mes sœurs Azza et Souhir pour leur gentillesse et leur affection, et aussi ma femme Nadia et mon fils Zakarya qui m'ont permis de trouver un équilibre loin de ma terre natale.

Je dédicace cette thèse à ma famille : ma mère, mon père, qui m'ont moralement soutenu malgré la distance qui nous sépare.

Table des matières

Nomenclature générale

Nous commençons ce mémoire par une introduction à l'ensemble des acronymes utilisés pour faciliter la compréhension. Le lecteur trouvera dans cette partie une référence complète, il n'aura pas à rechercher, pour chaque acronyme, sa première occurrence pour en déceler la signification.

AADL : Architecture Analysis and Design Language

AMDEC : Analyse des Modes de Défaillance, de leur Effet et leur Criticité

AEEL : Analyse des Effets des Erreurs du Logiciel

BDD : Block Definition Diagram

CAO : La Conception Assistée par Ordinateurs

CAE : Computer-Aided Engineering

CASE : Computer-Aided Software Engineering

CFD : Control Flow Diagram (Diagramme de flot de contrôle)

DFD : Diagramme de Flot de Données

DSL : Domain Specific Language
Tout langage spécifique à un domaine particulier est un DSL. Un DSL est destiné à un problème restreint. Contrairement à un langage général destiné à un large éventail de problèmes.

DSM : Dependency Structure Matrix ou encore Design Structure Matrix.
C'est une matrice carrée qui résume les dépendances existantes entre les composants d'un système.

EMF : Eclipse Modelling Framework
EMF est un plugin (additif) de la plate-forme de développement Eclipse. Il intègre un ensemble de fonctionnalités permettant de manipuler des modèles.

GEF : Graphical Editing Framework

GMF : Graphical Modelling Framework

IBD : Internal Block Diagram

MARTE : Modeling and Analysis of Real-Time and Embedded systems

MBSE : Model-Based Systems Engineering

MCSE : Méthodologie de Conception des Systèmes Electroniques

MDA : Model Driven Architecture

MDD : Model Driven Development

MDE : Model Driven Engineering

OCL : Object Constraint Language

OMG : Object Management Group

PIM : Platform Independent Model

PSM : Platform Specific Model

SA : Structured Analysis

SDL : Specification and Description Language

SysML: Systems Modelling Language

SDRTS: Structured Development for Real-time Systems)

UML : Unified Modelling Language

Chapitre 1

I - Problématique

« Toutes choses étant causées et causantes,… et toutes s'entraînant par un lien naturel et insensible, qui lie les plus éloignées et les plus différentes, je tiens impossible de connaître les parties sans connaître le tout, non plus que de connaître le tout sans connaître particulièrement les parties ».

Pascal.

1.1 Introduction de la problématique du mémoire :

L'ingénierie des systèmes moyennant des outils informatiques ou encore le Computer-aided engineering (CAE) est aujourd'hui et depuis quelques années un sujet qui attire l'attention des scientifiques et particulièrement les ingénieurs pour les différents avantages que le monde numérique peut apporter en termes de gestion de documents et d'informations, de traitement de ces informations, de leurs visualisation, d'automatisation des tâches répétitives, d'aide à la décision et d'expérimentation. Cette conception informatisée a beaucoup évolué et a donné naissance à des outils d'ingénierie système très puissants tels que DOORS[1] et SMARTEAM[2] pour la gestion des exigences. Elle a aussi donné des outils de modélisation structurale et comportementale tels que STATEMATE[3] et CORE[4], mais aussi des outils d'ingénierie de domaines tels que la CAO mécanique avec CATIA **[CATIA]**, la CAO électronique avec KICAD **[KICAD]** ou HyperLynx **[HyperLynx]**. Les possibilités qu'offre la conception informatisée aux ingénieurs en ont fait un outil indispensable pour concevoir des systèmes qui semblent diverger en termes de complexité et de taille, aussi bien vers l'infiniment grand que vers l'infiniment petit. Ce qui rend le nombre d'informations humainement impossibles à gérer et à prendre en compte. Mais les limites de ces outils sont encore loin d'être atteintes.

Dans toute cette panoplie d'outils, chaque ingénieur peut trouver l'outil qui correspond à son domaine, or, les systèmes conçus aujourd'hui sont de plus en plus intégrés et de plus en plus hétérogènes, ainsi, l'approche systémique en conception est devenue inévitable car la complexité de l'intégration des sous-systèmes de domaines technologiques différents peut souvent être la cause de l'échec du projet. C'est pour remédier à de tels problèmes que la conception informatisée des systèmes doit aujourd'hui offrir des implémentations efficaces de méthodes systémiques qui permettent le support et/ou la régie de tout le cycle de conception des systèmes. Cette

[1] DOORS est l'outil de gestion des exigences de Telelogic.
[2] ENOVIA SMARTEAM est le logiciel de gestion de données collaboratives de Dassault Systèmes.
[3] STATEMATE est un produit de Telelogic permettant la conception graphique de systèmes embarqués.
[4] CORE est un outil de VALTECH supportant tout le cycle de vie d'ingénierie système.

proposition n'implique pas de ne plus utiliser des outils spécifiques à des domaines technologiques particuliers tels que ceux cités précédemment, mais il est nécessaire de mettre en place une plateforme qui puisse représenter un repère pour tous les concepteurs qui travaillent sur le projet en offrant l'interopérabilité des différents outils. En tenant compte de ce double développement in extenso ; non seulement la complexité des systèmes conçus croît continuellement mais en plus ils sont de plus en plus hétérogènes et surtout intégrés, nous concentrons nos efforts dans cette thèse sur la conception des systèmes hétérogènes hautement intégrés dits mécatroniques suivant l'approche systémique ou l'ingénierie système.

Le terme « mécatronique » est un néologisme rassemblant les termes « Mécanique » et « Electronique ». Les systèmes dits mécatroniques intègrent les technologies mécaniques, électroniques, automatiques et informatiques. La mécatronique en elle-même a été proposée comme domaine de recherche à part entière pour mettre l'accent sur la nécessité de trouver des méthodes innovantes permettant la conception de tels systèmes hautement intégrés. En mécatronique, il ne s'agit pas d'assembler des composants de domaines technologiques différents, mais il est nécessaire de considérer le système dans sa globalité pendant tout le cycle de sa conception. Ceci est d'autant plus vrai sachant que l'assemblage des solutions optimales de chaque composant d'un domaine technologique différent ne donne pas forcément la solution globale optimale.

Dans l'approche mécatronique, la pluridisciplinarité du projet nécessite une collaboration de plusieurs intervenants (sociétés, équipes, etc.) autour d'un référentiel unique qui permet d'entreprendre une ingénierie systémique et concourante. Les différents métiers doivent participer ainsi à la mise en place d'un modèle commun du système. L'existence de ce modèle unique permettrait la vérification et la validation du modèle complet du système ce qui permettrait de découvrir le plutôt possible les incompatibilités et les problèmes d'intégration susceptibles d'apparaître entre les différents domaines technologiques. Le projet STEP **[STEP]** est un projet qui a été entrepris pour répondre à ce besoin, il définit une série de formats d'échange

spécifiques à des domaines particuliers, ces formats sont des AP (Application Protocol). Parmi ces formats, l'AP233 **[AP233] [AP233ConceptModel]** est la spécification des échanges relatifs aux données des activités d'ingénierie système.

L'INCOSE **[INCOSE]** (International Council On Systems Engineering) a de son côté développé le langage d'ingénierie système SysML **[SysML2007]**, adopté par la suite par l'OMG[5] **[OMG]** (Object Management Group) pour fournir un langage standard supportant les activités de l'ingénierie système. Ce langage a été développé de manière a supporter l'AP233, ce qui permettra d'avoir des outils interopérables (grâce à l'AP233) utilisant un langage standard (SysML).

Ainsi, les méthodes et outils dédiés aux concepteurs des systèmes mécatroniques doivent leur permettre de s'insérer dans ce cadre général de standardisation. De plus, ils doivent leur fournir les moyens d'entreprendre cette démarche systémique et collaborative tout en leur permettant la créativité, l'expérimentation et le prototypage virtuel qui est un moyen indispensable de vérification très tôt dans le cycle de conception.

Il est clair que l'étude d'un système doit permettre de le comprendre sous un certain nombre d'angles différents pour pouvoir le construire. Dire qu'il faut le comprendre sous tous ses angles est bien sûr utopique, car tout système réel est régi par un nombre infiniment grand de variables que nous ne pouvons pas dénombrer. Le concepteur sélectionne alors, selon les limites de la science dont il dispose (ses compétences), selon le type du système étudié et ses finalités, les différents angles ou les vues sous lesquels il juge pertinent d'étudier le système. On peut ici citer *Ross Ashby* qui a défini le mot « système » en ces termes: « *A System is a set of variables sufficiently isolated to stay discussable while we discuss it*»[6]; Il met l'accent dans cette définition sur l'importance de l'abstraction dans la conception. Ceci implique qu'un modèle du système est spécifié (figure 1.1) car un modèle est une abstraction d'un système qui

[5] OMG : L'Object Management Group est un organisme de standardisation de technologies objet.
[6] Page web dédiée au professeur W.Ross Ashby sur le site web de la Georges Washinghton University, http://www.gwu.edu/~asc/biographies/ashby/quotations.html

contient les propriétés les plus pertinentes de ce système de manière que le concepteur peut supposer que son système final sera fidèle au modèle ou au moins aura des différences qu'il juge insignifiantes.

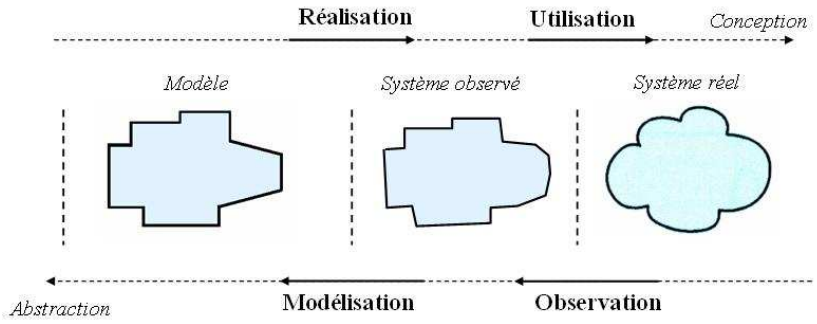


Figure 1.1. Niveaux d'abstraction en modélisation.

Pour comprendre un système, les vues nécessaires que les concepteurs doivent appréhender sont de natures différentes (figure 1.2); Elles peuvent être relatives à la structure du système, à son comportement interne ou aux connections de ses interfaces avec son environnement. Les méthodes et outils de conception doivent ainsi permettre la spécification de ces éléments.

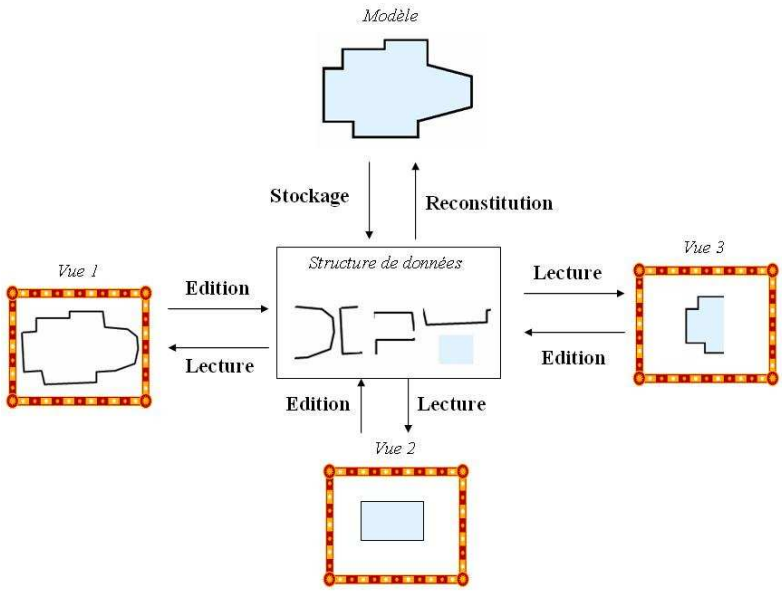


Figure 1.2. Un modèle peut être vu sous plusieurs angles.

Il est ainsi nécessaire de mettre à la disposition des concepteurs, les moyens de décrire les vues les plus pertinentes du système. Mais il est aussi nécessaire de disposer de moyens d'adapter ces vues aux besoins spécifiques des concepteurs et d'intégrer des

aspects sémantiques permettant de donner plus de précisions sur les idées exprimées dans ces vues. Finalement il faut fournir les moyens aux concepteurs de vérifier leurs idées et décisions.

De plus, dans cette approche mécatronique de conception, puisque nous cherchons à travailler autour d'un modèle commun, l'utilité des outils informatiques réside non seulement dans l'organisation et la visualisation de ces informations qui est le rôle statique mais aussi dans l'exploitation de ces informations pour en extraire d'autres qui sont implicites dans les informations de départ. Dans le cas de l'étude des systèmes, pour le concepteur, une des exploitations les plus utiles de ces informations ou modèles est la simulation et le prototypage virtuel. La simulation réside dans l'expérimentation du comportement en contrôlant ses entrées et en observant ses sorties. Le prototypage virtuel permet la visualisation 3D et la manipulation du modèle du système. La combinaison de ces deux types d'exploitations donne ainsi la construction d'un modèle virtuel simulable du système. Cette expérimentation virtuelle est un outil qui permet de réduire les coûts de la réalisation d'un système car elle permet de diminuer le nombre de prototypes réels nécessaires, de découvrir les erreurs de conception très tôt ce qui permet de réduire le time-to-market du produit, cela permet aussi de disposer de moyens de communication plus convaincants avec les clients. De plus, dans certaines situations l'expérimentation virtuelle permet de vérifier des systèmes critiques (nucléaire), ou impossibles à réaliser (infiniment grand, infiniment petit).

1.2 Objectifs de cette thèse

Dans cette thèse nous nous sommes posé pour objectif de proposer une méthode destinée à supporter les activités de l'ingénierie système et plus spécifiquement des systèmes mécatroniques. Cette méthode doit couvrir toutes les tâches d'ingénierie système à partir de l'expression des besoins jusqu'au prototypage virtuel et la simulation en passant par l'analyse des besoins et la description du modèle avec ses vues statiques et dynamiques. Ceci doit tenir compte de l'ouverture que doit offrir cette méthode pour permettre l'intégration au fur et à mesure, de nouvelles vues pour la

représentation de la structure et du comportement ou encore des méthodes de fiabilité (AMDEC, AEEL, QMU, etc.). De plus, cette méthode doit offrir une base pour entreprendre une démarche de conception itérative qui permet d'exploiter les retours d'informations des simulations en réintégrant itérativement des évolutions aux modèles. Nous devons proposer dans chaque étape de conception les techniques et modèles adéquats pour l'étude des systèmes mécatroniques permettant de laisser au concepteur les libertés indispensables à la créativité.

Après avoir situé la problématique, nous nous intéressons maintenant à une solution qui permettra de faire face à ces besoins. Nous avons constaté que les techniques mises en œuvre dans la conception diffèrent d'un domaine à un autre (mécanique, logiciel, etc.). Etant donné que des domaines variés évoluent généralement de manière parallèle, il serait judicieux d'élargir les domaines d'application des techniques utilisées dans chaque domaine à tous les autres domaines, si cela se trouve être bénéfique. Ainsi, nous avons décidé d'entreprendre une approche de conception orientée objet pour les raisons que nous citons ici :

- Tout d'abord, le concept objet est un concept universel qui permet d'englober toute chose ou concept.
- Les méthodes de conception Objet dans le monde du génie logiciel ont atteint une certaine maturité en termes de vues, de processus (cycles) et de traitements sur les modèles qui peut être profitable à l'ingénierie système.
- Les théories Objet sont en plein développement et leurs évolutions ont donné naissance à l'ingénierie guidée par les modèles qui peut, grâce à l'expertise dans le génie logiciel, être étendue à l'ingénierie système.
- L'émergence d'une panoplie d'outils ouverts favorise l'implémentation d'une telle approche.

Notre thèse se concentre ainsi sur la spécialisation à l'ingénierie système de techniques de conception utilisées en génie logiciel. En effet, le génie logiciel se trouve être très avancé en terme de méthodes et d'outils de conception vu la proximité entre la

nature du produit final (en génie logiciel du code source) et celle des modèles utilisés par ces outils (Données ou bases de modèles).

Nous avons décidé d'explorer cette piste et d'en rendre compte dans cette thèse pour fournir une réponse à la problématique introduite auparavant. Nous commencerons par présenter les techniques, méthodes et formalismes qui nous intéressent en l'occurrence dans ce travail.

Le mémoire que nous présentons ici s'organise de la manière suivante :

- Au début nous dressons un état de l'art de la conception des systèmes et en particulier les systèmes hétérogènes tels que les systèmes mécatroniques.
- ensuite nous explorons en détail ces nouvelles technologies Objet, en particulier l'architecture MDA (Model Driven Architecture) et SysML (Systems Modeling Language).
- Nous nous focaliserons par la suite sur le caractère évolutif de ces technologies et nous exposerons quelques applications de cette évolutivité en particulier une extension à SysML qui intègre l'utilisation des Bond Graphs et des métriques qui sont un moyen de contrôle de la qualité des modèles produits très tôt dans le cycle de conception.
- Ensuite nous présentons les outils permettant l'implémentation de cette plate-forme et la solution adoptée.
- Enfin nous exposerons l'implémentation de cette méthode et quelques exemples d'application.

Chapitre 2

II - Etat de l’art des méthodologies d’ingénierie système et de conception

« Le processus de découverte de nouvelles représentations est le chaînon manquant de nos théories de la pensée. Il est désormais un des principaux domaines de recherche en psychologie cognitive et en intelligence artificielle »

Herbert Simon

2.1 Introduction à la conception

Nous allons commencer par définir ce qu'est la conception pour pouvoir comprendre les besoins des ingénieurs par rapport aux processus et par rapport aux modèles. La conception de systèmes est un processus itératif dans lequel prend part une équipe multidisciplinaire qui tente de transformer des besoins client en une solution optimisée **[INCOSE2004]**.

La conception est aussi une tâche de créativité et de résolution de problèmes. Chaque problème étant unique, la conception nécessite une certaine créativité. Pour stimuler cette créativité, des outils et méthodes appropriés doivent être mis à la disposition des concepteurs. Herbert Simon **[Simon1992]** dit qu'un expert dispose d'un ensemble de 50.000 astuces et outils mentaux utiles dans un domaine d'expertise et que résoudre un problème nécessite la mise en correspondance du problème avec les connaissances dont il dispose. Il ajoute que plus les connaissances sont larges et accessibles, plus le processus va réussir. Curtis **[Curtis1988]** ajoute que la réussite d'un projet dépend d'un petit nombre de concepteurs exceptionnels qui réfléchissent au niveau « système ». Ceci dit, on peut déduire de ces deux affirmations qu'il est indispensable de fournir les méthodes et outils qui permettront de mieux utiliser ces connaissances et expertises en fournissant une vue globale du système aux concepteurs.

Dans le domaine logiciel Dijkstra **[Dijkstra1968]**, Hoare et Wirth **[Wirth1971]** expriment le besoin de maîtriser la complexité des systèmes car la construction de programmes corrects nécessite que les programmes soient gérables intellectuellement. Ils indiquent que la clé de cette « gérabilité » est dans la structure même du programme et que l'utilisation judicieuse de la composition de blocks de programmes améliore la correction. Dans **[Dijkstra1968b]** Dijkstra explique que la directive GO TO est préjudiciable à la conception des logiciels puisqu'il est difficile de retrouver les points stables du programme dans lesquels les données sont conformes à la progression voulue. Cette affirmation est généralisable à tout système et elle est aussi vraie pour les

systèmes mécatroniques d'autant plus qu'ils sont à grande composante logicielle. Ainsi l'encapsulation est un patron de conception sécuritaire.

2.1.1 Quelques notions théoriques de la conception :

Un modèle est une interprétation explicite par son utilisateur de la compréhension d'une situation, il peut être exprimé par des mathématiques, des symboles, des mots etc. C'est une description d'entités et de relations entre elles **[Calvez1990].** D'après **[Baum2001]**, un modèle est une représentation d'une théorie ou d'une partie d'une théorie souvent utilisée pour mieux comprendre les liens entre causes et effets.

L'association française d'ingénierie système **[AFIS]** explique qu'« Il est nécessaire de s'appuyer sur des représentations tant du problème que de ses solutions possibles à différents niveaux d'abstraction pour appréhender, conceptualiser, concevoir, estimer, simuler, valider, justifier des choix, communiquer. C'est le rôle de la modélisation ». La modélisation **[Endres2003]** est une manière de décrire le système de sorte que certaines propriétés soient visibles et de sorte de pouvoir analyser le système automatiquement.

Avant l'avènement de l'ingénierie des modèles, l'utilisation des modèles avait principalement pour but de comprendre le système et de communiquer avec les membres d'une même équipe. Dans l'ingénierie des modèles (MDE) par contre, les modèles représentent réellement le système à développer. Ces deux situations différentes impliquent des modèles de qualités différentes. Contrairement à la première situation, dans l'ingénierie des modèles, un modèle doit être précis, non ambigu et complet.

D'après Friedenthal et Burkhart **[Friedenthal2003],** un langage de modélisation doit satisfaire ces 9 critères :

- Facilité d'utilisation : Le langage doit pouvoir être utilisé et compris (bien interprété) par un large public.

- Non-ambiguïté : Le langage doit être basé sur des sémantiques bien définies avec une notation non-ambiguë, donnant lieu à un ensemble consistant de vues de modèles qui adhérent à des règles de bonne formation (théorie des langages).
- Précis : Le langage doit spécifier la sémantique qui peut être transformée en une représentation formelle basée sur les mathématiques (voir méthodes Z, B, OCL..). Ce critère doit faciliter l'exécution de la spécification et des modèles dans n'importe quel niveau hiérarchique afin de valider les besoins par rapport au cahier des charges et vérifier que le modèle satisfait bien à ces besoins.
- Complet : Le langage doit supporter l'expression de tous les détails émanant de la modélisation de systèmes, de l'analyse à la spécification, la conception et la vérification.
- Adaptatif/taille du système: Le langage doit supporter l'abstraction, l'élaboration et le raffinement (généralisation/spécialisation, décomposition, collection, vues multiples..) pour fournir des solutions adaptatives à la modélisation de systèmes complexes.
- Adaptatif/domaine : Le langage doit fournir les moyens d'extension vers des domaines spécifiques (aérospatial, télécommunications, automobile…)
- Evolutif : Le langage doit permettre le changement, l'évolution et supporter la compatibilité avec les versions antérieures.
- Echange de modèles et de diagrammes : Le langage doit supporter le mapping vers l'AP233[7] **[AP233]** (Application Protocol 233) pour l'échange d'informations entre outils. L'AP233 et XMI vont fournir des mécanismes d'échange de données, ces données pourront éventuellement être représentées dans d'autres langages de modélisation (diagrammes comportementaux, IDEF0,…langages formels..) aussi bien que des langages/outils de gestion de besoins et d'autres outils/modèles d'analyse et de conception. En plus de l'échange sémantique, le langage doit supporter un échange de diagrammes pour faciliter l'échange sémantique entre outils.

[7] L'AP233 ou Application Protocol 233 est un projet STEP **[STEP]** visant à établir un format d'échange des informations d'ingénierie système.

- Indépendant du processus et méthode : Le langage doit pouvoir supporter des processus d'ingénierie système tels que les standards EIA 632 **[ANSI/EIA632]**, ISO/IEC 15288 **[IEEE15288]** et non contraindre un choix de processus ou méthode.

2.1.2 Outils mentaux de la conception

En programmation structurée, Dijkstra **[Dijkstra1968]** décrit trois outils mentaux, le raisonnement par énumération, l'induction mathématique et l'abstraction ; nous en donnons les définitions généralisées à la conception inspirées de celles de Dijkstra qui sont spécifiques à la programmation :

- L'énumération est la capacité à énumérer les différentes solutions possibles pour une problématique donnée en distinguant les subtilités de chaque solution.
- L'induction mathématique est le moyen de raisonnement permettant d'induire des informations à partir de prédicats mathématiques (règles, théorèmes) basé sur l'inférence logique.
- L'abstraction se fait par rapport à l'architecture du système (Dijkstra parle de données en programmation) ou par rapport à son comportement (Dijkstra parle des procédures). Elle permet de limiter l'espace du raisonnement par énumération.

Il déduit alors qu'il ne faut utiliser que des outils de conception qui permettent de mettre en œuvre ces outils mentaux. Dijkstra **[Dijkstra1968]**, avec le système THE[8] développe un système en utilisant les niveaux d'abstraction pour venir à bout de la complexité et définit les règles qui lient ces niveaux d'abstraction :

- La conception est décrite en couches.
- Les niveaux supérieurs utilisent les services des niveaux plus bas.
- Les bas niveaux ne peuvent pas accéder aux niveaux plus hauts.

[8] Le système THE est un système d'exploitation supportant le multiprogrammation conçu par une équipe dirigée par Edsger Dijkstra.

- Les bas niveaux sont implémentés en premier et fournissent une machine virtuelle pour implémenter les niveaux suivants. C'est une technique « Bottom Up ».

L'abstraction entraîne *de jure* l'utilisation du raffinement. Le raffinement est un outil mental qui permet, après avoir décomposé le système, d'agrandir la granularité de la conception en décrivant plus de détails. Dans Wirth **[Wirth1971]** : « Diviser pour régner » l'auteur utilise une technique « Top Down » pour décomposer un système à partir d'une spécification de conception vers des niveaux élémentaires. Il entreprend une séquence d'étapes de raffinement pour construire son système. Cette approche implique qu'une intégration de ces éléments doit ensuite être menée. DeRemer **[DeRemer1976]** explique que la structuration d'un grand nombre de composants pour composer un système est une activité intellectuelle complètement différente de la construction des composants de manière indépendante et que l'intégration de ces composants peut devenir une tâche très complexe. Ainsi la définition des interfaces entre ces composants doit être entreprise dés le départ. D'autre part, la définition des interfaces permet une meilleure réutilisation. La réutilisation doit être un objectif pour le concepteur ; pour cela la connectivité des composants doit être réduite au minimum. Ceci permet de développer des composants le plus indépendamment possible des autres. Les preuves de correction sont plus faciles à faire. De plus la réutilisation est augmentée et le système global devient plus compréhensible.

On distingue cette première frontière entre composants/sous-systèmes sur le plan de la réalisation, celle de la technologie. Un processus de décomposition du système permettra de séparer les sous-systèmes tout en identifiant les interfaces de chacun pour pouvoir réassembler le tout. Cette décomposition/Recomposition doit pouvoir être décrite par les modèles utilisés. La description des interfaces passe aussi par la description des communications qui ont lieu entre différents composants/sous-systèmes, particulièrement ceux dits réactifs : Ceci implique un échange d'informations de l'environnement vers le système et un ensemble d'actions entreprises par ce système vers l'extérieur, donc des réactions à des événements. La vue des interactions

système/environnement fait alors partie des vues nécessaires pour expliciter le fonctionnement du système. Les échanges doivent être bien décrits en particulier le sens, type (continu, discret) et flux de l'échange.

Il est aussi important que la méthode proposée à l'ingénieur lui permette au moins de rester créatif sans le restreindre dans sa réflexion : améliorer sa créativité est bien sûr meilleur. Budgen **[Budgen1999]** met l'accent sur cela: "Quand arrêterons-nous de prétendre que concevoir du logiciel peut être vu comme un ensemble d'activités bien définies à suivre, et reconnaîtrons que c'est un processus créatif qui demande que nous trouvions des moyens de développer les habilités à concevoir dont nous avons besoin pour construire les systèmes logiciels de demain ? »

Ce que dit Budgen est aussi valable pour les systèmes hétérogènes car chaque problème est différent, un processus dans lequel on réutilise beaucoup de composants est différent d'un processus où on reconstruit tout de zéro. Mais quel que soit le processus, nous pouvons définir un certain nombre de vues du système qui nous permettront de concevoir une famille de systèmes que sont les systèmes mécatroniques.

Au cours de ce processus il est important d'évaluer la qualité de la solution qui réalise un certain besoin. Une quantification de cette qualité est essentielle pour pouvoir orienter les itérations en aval du cycle. Nous présenterons quelques propositions permettant cette quantification qualitative dans le chapitre 8 « identification de métriques ».

2.2 L'ingénierie système

2.2.1 Définition du terme « Système »

Nous commençons par définir ce qu'est un « système ». Nous pouvons trouver diverses définitions du système, nous en citons les suivantes :

- Un système est un ensemble de composants inter reliés qui interagissent les uns avec les autres d'une manière organisée pour accomplir une finalité commune. **[NASA1995]**
- Un système est un ensemble intégré d'éléments qui accomplissent un objectif défini. **[INCOSE2004]**

Nous nous restreindrons à caractériser les systèmes conçus par l'homme et nous les définissons par :

Un système est un ensemble de composants qui collaborent *à la réalisation d'un ensemble de tâches en vue de fournir un ensemble de services, cet ensemble est soumis à un environnement donné et* interagit *ainsi avec un sous-ensemble des éléments de cet environnement (Figure 2.1).*

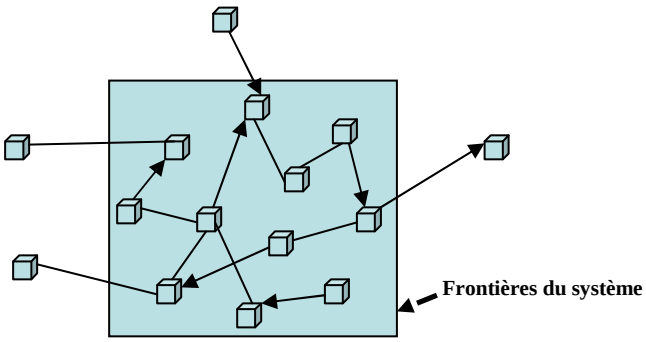


Figure 2.1. Définition du « Système »

2.2.2 Introduction aux systèmes mécatroniques SM :

Les systèmes mécatroniques (SM) sont les systèmes intégrant principalement les technologies informatiques, mécaniques électroniques et automatiques. L'évolution de la technologie a donné naissance à ces systèmes complexes qui améliorent les rendements, ajoutent de la fonctionnalité, de l'autonomie dans tous types de systèmes. Schöner **[SCHONER2004]** ajoute que les systèmes mécatroniques permettent d'arrêter l'expansion physique des systèmes qui est due à cette multiplication des fonctionnalités dans un volume donné. Ces systèmes mécatroniques deviennent alors de plus en plus riches en fonctionnalités et de plus en plus complexes et leur distribution devient de plus en plus large. La demande évoluant constamment, il y a aussi besoin d'une grande

réactivité face aux exigences du marché. Maîtriser le cycle de vie du produit devient un impératif vital. Les techniques visant à améliorer la conception des systèmes tentent d'apporter des solutions à ces contraintes.

Plusieurs définitions sont données pour définir les systèmes mécatroniques dont :

« La mécatronique est l'intégration synergique de l'ingénierie mécanique avec l'électronique et le contrôle intelligent de calculateurs dans la conception et la fabrication de produits et processus industriels. » **[Harashima1996]**

Isermann **[Isermann2000]** résume les définitions données à la mécatronique. Il estime que toutes les définitions sont d'accord pour dire que la mécatronique est un domaine interdisciplinaire dans lequel les disciplines suivantes agissent ensemble :

- Systèmes mécaniques (éléments mécaniques, machines, mécanique de précision)
- Systèmes électroniques (micro-électronique, électronique de puissance, capteurs et actionneurs).
- Technologie de l'information (théorie des systèmes, automatisation, génie logiciel, intelligence artificielle).

Nous définissons les systèmes mécatroniques de la manière suivante :

Les systèmes mécatroniques sont des systèmes multi-technologiques, à dominantes mécanique et électronique, réactifs à contraintes temps réel.

Nous allons axer nos réflexions sur ces spécificités dans notre étude des méthodes de conception de tels systèmes.

Multi technologiques : Plusieurs domaines technologiques sont mis en œuvre pour les parties commande et opérative. Les domaines technologiques auxquels nous nous intéressons sont particulièrement la mécanique, l'électronique, l'automatique et l'informatique.

Réactifs : L'électronique est la technologie la plus courante et économique aujourd'hui pour mettre en œuvre cette réactivité.

A contraintes temps réel : Ces systèmes sont le plus souvent immergés dans leur environnement et doivent permettre une automatisation d'un ensemble de tâches. Les services exécutés par un système mécatronique doivent répondre à des contraintes de temps satisfaisants pour permettre aux clients de ces services de les exploiter correctement.

Pour donner un exemple typique des systèmes mécatroniques nous citons un système intelligent de suspension de camion qui permet d'équilibrer un chargement pour maintenir le camion droit, qui réagit aux virages et aux terrains difficiles pour maintenir l'équilibre et une conduite agréable. L'application de la mécatronique concerne plusieurs domaines dont la robotique (Aibo **[sony]**, Nao **[Aldebaran robotics])** et l'automobile (ABS, ESP, Suspension active).

L'appellation « mécatronique » est un classement selon la technologie de réalisation. Du point de vue fonctionnel, un système mécatronique est un système pouvant être vu comme une collaboration de deux grandes parties :

- Une partie *Opérative* : Elle est constituée de parties mécaniques et électromécaniques.
- Une partie *Commande* : Elle est constituée à partir des technologies électroniques, informatiques et automatiques. Elle peut être un système de contrôle en boucle ouverte ou fermée.

Les systèmes mécatroniques, étant généralement des systèmes complexes, l'approche de conception d'un système mécatronique aussi appelée conception mécatronique veut que l'on entreprenne une approche de conception simultanée de ces deux parties. La distinction des parties commande et opérative n'est plus une division d'un projet en deux sous-projets interagissant mais la collaboration de plusieurs compétences sur un projet unique avec un référentiel unique. Dans l'article « Pour une

réforme de la pensée » Edgar Morin décrit globalement la modélisation systémique complexe en disant : « La pensée complexe est une pensée qui cherche à la fois à distinguer - mais sans disjoindre - et à relier. D'autre part, il faut traiter l'incertitude. Le dogme d'un déterminisme universel s'est effondré. L'univers n'est pas soumis à la souveraineté absolue de l'ordre, il est le jeu et l'enjeu d'une dialogique (relation à la fois antagoniste, concurrente et complémentaire) entre l'ordre, le désordre et l'organisation ». Ainsi le fait de distinguer une partie opérative d'une partie commande n'implique pas de les disjoindre et de les penser séparément mais de les comprendre et de comprendre les liens qui existent entre elles. Ceci est très bien présenté par Pascal qui dit : « Toutes choses étant causées et causantes,… et toutes s'entraînant par un lien naturel et insensible, qui lie les plus éloignées et les plus différentes, je tiens impossible de connaître les parties sans connaître le tout, non plus que de connaître le tout sans connaître particulièrement les parties ». L'ingénierie système est une discipline qui entreprend cette vision systémique dans la conception.

2.2.3 Qu'est-ce que l'« Ingénierie système » :

L'ingénierie système est définie dans **[ANSI/EIA 632]** ainsi : « L'ingénierie système est une approche interdisciplinaire qui englobe tous les efforts techniques pour développer et vérifier un ensemble de solutions relatives aux systèmes, aux utilisateurs et aux processus dans un cycle de vie total et intégré pour satisfaire des besoins client.

L'ingénierie système réunit :

- Les efforts techniques reliés au développement, fabrication, vérification, déploiement, opérations, support, mise à disposition et formations d'utilisateurs des produits et processus d'un système.
- La définition et la gestion de la configuration du système.
- La traduction de la définition du système en termes d'une séquence de tâches structurées.
- Et le développement d'informations pour la prise de décisions du management. »

Selon le DoD (Department of Defence): « L'ingénierie système est un processus appliqué par une équipe en vue de transformer des besoins client en un système effectif ». Cette définition limite l'ingénierie système aux activités techniques.

La U.S Defence Systems **[Tien2003]** définit l'ingénierie système par : « L'application d'efforts scientifiques et d'ingénierie pour :

- transformer un besoin opérationnel en une description de paramètres de performance d'un système et une configuration du système à travers l'utilisation d'un processus itératif de définition, de synthèse, d'analyse, de conception, de test et d'évaluation.
- Intégrer des paramètres techniques reliés et s'assurer de la compatibilité des interfaces physiques, fonctionnelles et des programmes d'une manière qui optimise la définition et la conception globale du système
- Et d'intégrer la fiabilité, la maintenabilité, la sécurité, la survie, l'ingénierie humaine, et autres facteurs dans l'effort global d'ingénierie en respectant les objectifs en termes de coûts, délais, supportabilité et performances techniques. »

L'association des industries électroniques donne la définition suivante de l'ingénierie système **[Tien2003]**: « Une approche interdisciplinaire rassemblant tous les efforts techniques pour faire évoluer et vérifier un ensemble intégré de systèmes, de gens, de produits et de solutions de processus de manière équilibrée au fil du cycle de vie pour satisfaire aux besoins client ».

L'institut des ingénieurs électrique et électronique IEEE (The Institute of Electrical and Electronics Engineers) **[IEEE]** le définit comme étant **[Tien2003]**: «Une approche interdisciplinaire et collaborative pour dériver, faire évoluer, et vérifier une solution d'un système de manière équilibrée au cours du cycle de vie pour satisfaire les attentes du client et accéder à l'acceptation du public ». Ces définitions limitent la portée de l'ingénierie système, mais l'ingénierie système a évolué et des standards couvrant plus d'activités ont été définis.

Le biologiste Ludvig Van Bertalanffy est identifié comme étant le premier a avoir développé l'idée de généralisation de la réflexion systémique à tout type de système. Les contributions sont ensuite venues de tous les domaines, de la psychologie, l'anthropologie, la biologie, les communications. La théorie générale des systèmes a été définie par Bertallanffy K.E Boulding, R.W. Gerard et A.Rappoport en 1955 **[Yurstseven1999]**.

Un processus d'ingénierie système doit donner lieu à une solution qui soit :

- Robuste : Par rapport aux évolutions techniques/technologiques, par rapport à la taille du système (production) et aux évolutions du contexte.
- Evolutive/adaptative : Par rapport aux besoins utilisateur.
- Un point d'équilibre : Par rapport à l'ensemble des besoins, choix de conception, contraintes de conception, budgets, délais.

L'aspect qui nous intéresse de l'ingénierie système dans cette thèse est l'approche de conception systémique qui est appelée dans le standard IEEE15288 processus techniques. Les autres processus d'entreprise ou de gestion de projets décrits dans les méthodes d'ingénierie système sont en dehors de la portée de notre réflexion dans ce travail.

Dans cette thèse, nous nous sommes fixés de mettre en place une méthodologie supportant un processus d'ingénierie système orienté vers les systèmes mécatroniques. Nous définissons alors les termes « méthodologie », « processus », « méthode » et « outil » :

- Un processus est une séquence de tâches. C'est la description de l'articulation des tâches à accomplir sans la définition même de ces tâches.
- Une méthode est la définition des techniques permettant de réaliser une tâche particulière.
- L'outil est la plateforme permettant de s'avancer dans un processus donné et d'exécuter les méthodes propres à chaque tâche.

- Finalement, une méthodologie est l'ensemble processus/méthodes/outils pouvant être appliquée à un domaine particulier. **[Estefan2007]**

2.3. Les méthodologies d'ingénierie système

Nous avons vu dans le monde des systèmes se construire une panoplie de méthodologies de conception. Nous allons en présenter quelques unes du domaine logiciel, électronique ou d'ingénierie système. Parmi elles :

- MCSE **[Calvez1990]** qui repose sur un cycle de vie en « Y » **[Massiani2005]**, et qui se base sur une vue fonctionnelle, une vue temporelle et une vue matérielle.
- l'analyse structurée SA, elle utilise le diagramme de flot de données qui est un modèle hiérarchique permettant une approche descendante par raffinement.
- SADT, elle utilise l'enchaînement de boîtes élémentaires pouvant être raffinées en d'autres modèles SADT (boites + flots). Les modèles utilisés peuvent être des actigrammes (les boites sont les fonctions et les flots sont les données) ou des datagrammes (les boîtes sont les données et les flots sont les transformations).
- Ward et Mellor **[WARD1985]** est basée sur l'utilisation d'un DFD complété par un diagramme de flot de contrôles CFD en pointillés pour ajouter une sémantique d'exécution au diagramme. En effet, un DFD ne permet pas d'exprimer l'évolution temporelle de l'exécution.

Ces méthodes sont limitées par leurs portées et leur expressivité et par la difficulté de leurs intégrations avec d'autres méthodes ou formalismes. SADT et SA ne couvrent pas les vues dynamiques. De plus, le lien entre ces méthodes et les exigences ne sont pas couverts et toute implémentation de l'une ou l'autre de ces méthodes est une variante propre à l'éditeur de l'outil. D'autres approches, par souci de standardisation et de continuité du processus de conception, séparent les processus des modèles et

définissent des standards pouvant être adaptés dans des méthodes différentes : les processus d'ingénierie système sont le fruit de cette approche.

2.4 Les processus d'ingénierie système

Suivant la précision de leurs descriptions, trois types d'approches existent en conception **[Clarkson2005]**:

- Les approches abstraites : Elles décrivent le processus de conception à un haut niveau d'abstraction, elles peuvent être appliquées généralement à un large éventail de systèmes mais sont difficilement applicables et peuvent être interprétées de manières très différentes, elles apportent généralement peu au concepteur.
- Les approches procédurales, plus concrètes, se focalisent sur un aspect spécifique du projet de conception, plus pertinent par rapport à la pratique.
- Les approches analytiques : elles servent à des projets spécifiques, ce genre d'approches consiste en deux parties : une représentation utilisée pour décrire certains aspects du système puis des techniques, procédures et outils utilisant les représentations pour améliorer les activités de conception.

2.4.1 Panorama des processus d'ingénierie système

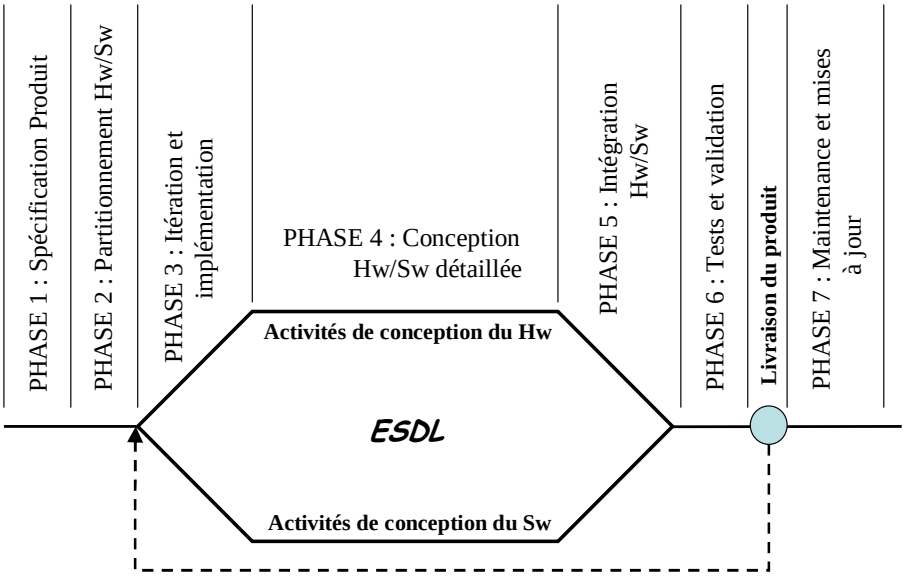
Une multitude de cycles de conception ont été développés dont le modèle en cascade (ou waterfall) **[Royce1970]**, le modèle en V **[Forsberg1995]** et le modèle du cycle en spirale de Boehm **[Boehm1988]**. Ce dernier est le plus adapté aux systèmes mécatroniques, il est basé sur la création de **prototypes et la simulation de ces prototypes itérativement**. Chaque cycle commence par une évaluation des risques et se termine par la résolution de ces risques suivie de l'implémentation et de tests.

Le cycle en spirale de Boehm est à l'origine de beaucoup d'autres évolutions comme le modèle Win-Win **[Boehm1994]** basé sur des conditions de réussite (win conditions). Ces cycles de conception ont été largement utilisés en génie logiciel et dans des systèmes à grande composante logicielle. D'autres modèles ont aussi été définies en

génie logiciel tel que le RUP (Rational Unified Process) **[RUP]** qui est un processus itératif et incrémental guidé par les besoins des utilisateurs et basé sur six *Best Practices*[9] **[RUP]**:

- Développement itératif.
- Gestion des exigences.
- Description de l'architecture par des composants.
- Modélisation graphique.
- Vérification de la qualité.
- Controller les changements.

Le processus ESDL (Embedded Systems Design Lifecycle) **[Berger2002]** se base sur le cycle de conception de la figure 2.2. La première phase consiste en la compréhension et la définition des besoins du client. Dans la deuxième phase on choisit quels sont les problèmes qui seront résolus par le matériel et ceux qui seront résolus par la partie logicielle. Ensuite deux activités concourantes se déroulent pour la conception du matériel et pour la conception de la partie logicielle. Ensuite, il y a une phase d'intégration, une phase de tests et de validation qui peut nécessiter une réitération du processus et finalement le produit est livré puis maintenu.



[9] *Best Practice* est un terme utilisé pour faire référence à un moyen ayant fait ses preuves et permettant de résoudre une problématique de la manière la plus efficace possible sans qu'une preuve formelle de son optimalité n'ai été produite.

Figure 2.2. Embedded Systems Design Lifecycle

Une autre variante de ESDL a été développée par Daniel Mann dans laquelle le choix du processeur se fait avant le partitionnement des activités de conception du matériel et de conception du logiciel. Ceci se fait en se basant sur l'expérience des concepteurs, des anciens produits similaires, etc. **[Berger2002]**

En ingénierie système, SIMILAR[10] **[Bahill1998] [Weilkiens2008]** est un processus qui définit sept activités principales (figure 2.3) que sont la définition du problème, la découverte de solution alternatives, la modélisation du système, l'intégration, la mise en service, la vérification des performances et la réévaluation de décisions prises au préalable suivant le résultat de cette vérification.

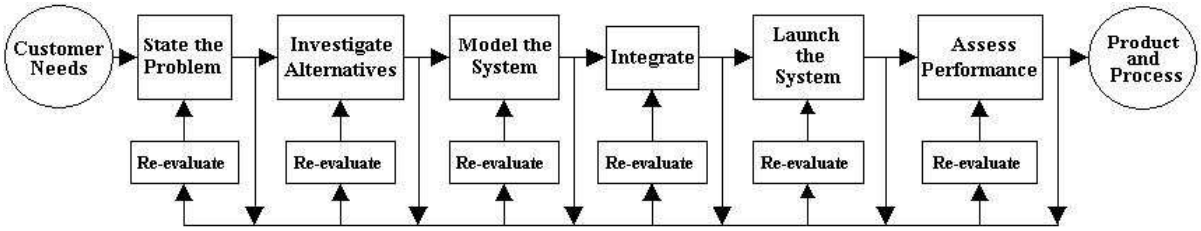


Figure 2.3. Le processus SIMILAR **[Bahill1998]**.

Harmony **[Harmony2004]** est un autre exemple d'un processus d'ingénierie système destiné à des systèmes à forte composante logicielle. Harmony s'organise autour des exigences (figure 2.4), il commence par identifier ces exigences et en déduire les fonctionnalités requises du système, ensuite par concevoir le système, définir ses modes de fonctionnement et ses états, ensuite concevoir et implémenter la partie logicielle puis l'intégrer en allouant le logiciel à l'architecture physique et finalement le valider.

[10] *SIMILAR* est l'acronyme de: ***S***tate the problem, ***I***nvestigate alternatives, ***M***odel the system, ***I***ntegrate, ***L***aunch the system, ***A***ssess performance, ***R***e-evaluate.

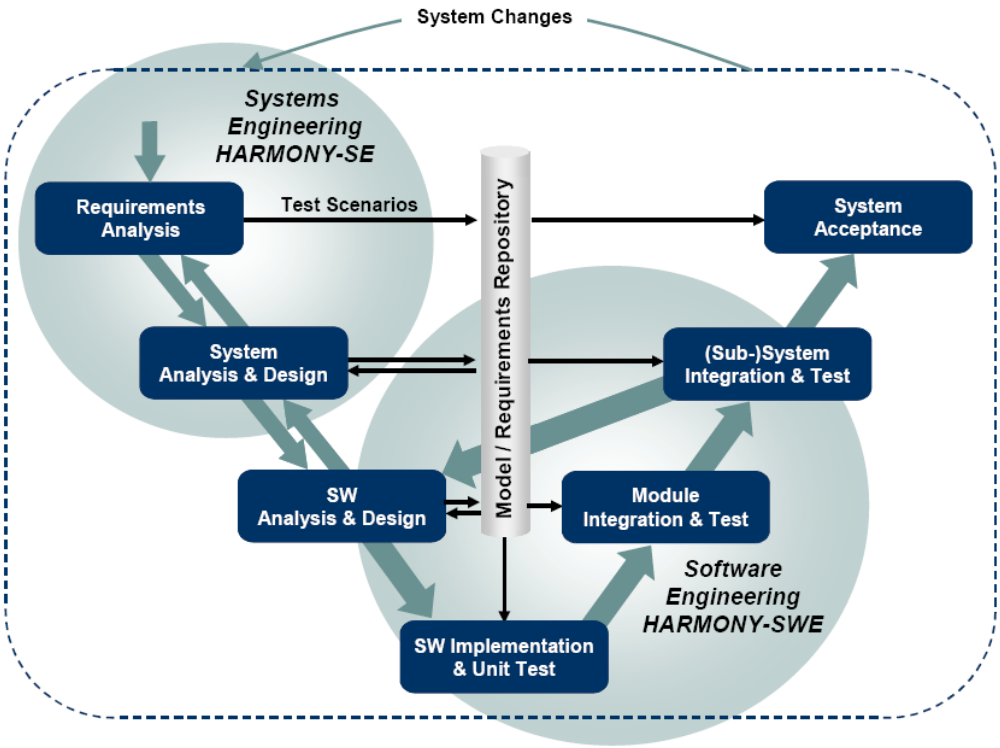


Figure 2.4. Le processus HARMONY de I-Logix.

Ces processus n'entreprennent pas une approche système, car ils séparent la partie matérielle de la partie logicielle dès les premières phases d'analyse puis prévoient l'intégration. En mécatronique, nous devons entreprendre une approche systémique.

En ingénierie système, des recommandations (approches abstraites selon **[Clarkson2005]**) ont été définies pour donner un cadre global à l'ingénierie système. Par exemple, la norme ANSI/EIA 632 **[ANSI/EIA 632]** qui a été développée par la EIA (Electronic Industries Alliance) **[EIA]** et adoptée en 1999 par l'ANSI (American National Standards Institute) **[ANSI]** décrit le processus d'ingénierie système comme suit (figure 2.5) :

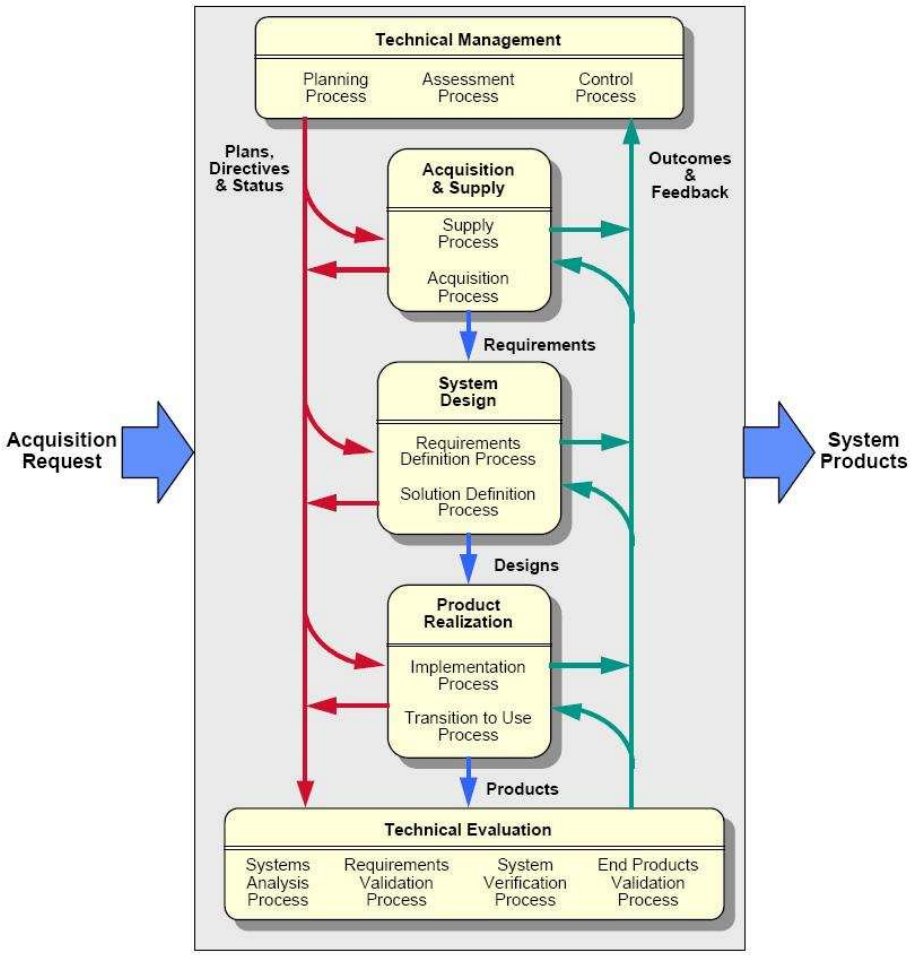


Figure 2.5. Processus d'ingénierie système ANSI/EIA 632.

Ce processus a ensuite été adapté par l'IEEE aux systèmes électriques et électroniques sous la forme du standard IEEE 1220 **[IEEE1220_1998]**. Le standard ISO 15288 a ensuite été réalisé pour couvrir toutes les phases du cycle de vie du produit, il a été ensuite standardisé par L'IEC (International Electrotechnical Commission) **[IEC]** puis par l'IEEE. La figure suivante montre les portées de chacun de ces standards (figure 2.6). L'IEEE 1220 se focalise sur les activités de conception. Tandis que l'EIA 632 étend ses activités à la gestion du projet et à l'acquisition au niveau de l'organisation et va jusqu'aux phases d'intégration et d'exploitation au niveau du système lui-même. Le standard ISO 15288, quant à lui, englobe toutes les activités susceptibles d'exister dans l'entreprise au cours d'un projet donné y compris les activités de gestion du projet et les activités de conception du système jusqu'à la phase terminale de retrait du système de son environnement d'exploitation (figure 2.6. Dans **[Swarz2006]** l'auteur fait remarquer que ces processus deviennent obsolètes dans un

cadre plus large. Par exemple Internet où l'entité qui contrôle le système global est un ensemble de clients qui déterminent leurs standards en fonction de leurs besoins (le W3C World Wide Web Consortium), des systèmes inter-opérants très variés sont ainsi conçus et mis en place. Le standard ISO/IEC 15288 (ou IEEE15288) ne traite cette interaction entre projets d'une entreprise que de manière très succincte (voir Annexe D **[IEEE15288]**).

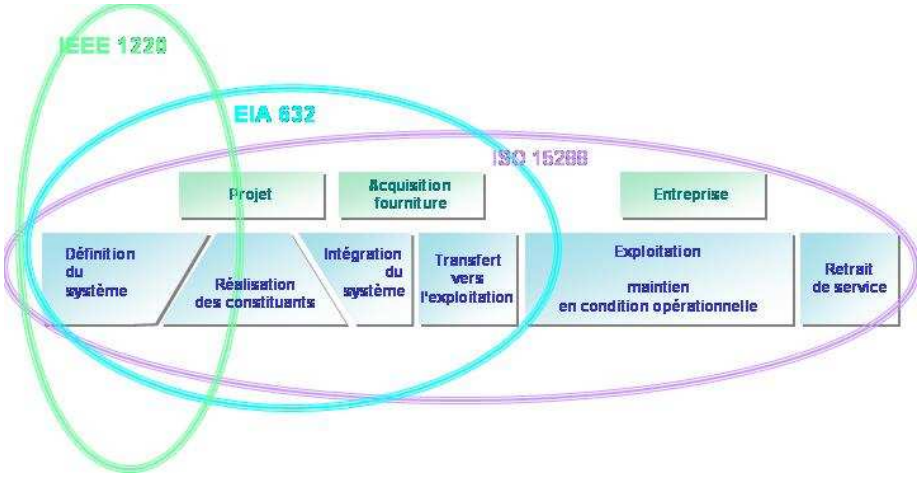


Figure 2.6. Les trois normes d'ingénierie système et leurs portées **[AFIS]**.

2.4.2 Le standard IEEE 15288

Le standard IEEE 15288[11] **[IEEE15288]** est un standard *IEEE* décrivant un ensemble de processus élémentaires intervenant dans le cycle de vie des systèmes crées par l'homme. A partir de cet ensemble on peut sélectionner et articuler les processus que l'on juge utiles pour un système particulier. Ces processus sont organisés en quatre catégories ; Les processus d'accords, les processus d'entreprise, les processus de projets et les processus techniques. De plus, un processus d'adoption du standard est décrit en annexe A de la norme et un exemple de cette adoption est présentée en annexe B. Finalement la spécification présente la définition de quelques concepts liés à l'ingénierie système. Dans le cadre de cette thèse, seuls les processus techniques nous sont utiles, ce sont les processus définissant les activités permettant de transformer des

[11] IEEE 15288 est au départ appelé l'ISO/IEC 15288 ensuite adopté par l'IEEE en 2004, nous avons eu à notre disposition le document [IEEE15288], pour celà, dans cette thèse, nous faisons référence au standard par l'appellation IEEE15288.

besoins pour un système donné en produit effectif, de le produire, l'utiliser, le maintenir et finalement le mettre hors service. Ces processus techniques sont détaillés (en vue de l'exploitation de ces détails dans notre proposition) dans l'annexe A de ce document mais sont brièvement décrits dans ce qui suit :

- Le processus de définition des besoins des parties prenantes : Ce processus permet la définition des exigences des parties prenantes sur le système. Les parties prenantes étant les utilisateurs, mais aussi les agents de maintenance, de formation et autres intervenants sur le système au cours de son cycle de vie. Ce processus doit permettre l'identification de ces intervenants et leurs besoins puis la transformation de ces besoins en exigences sur le système. Ces exigences seront la référence pour valider le système.
- Le processus d'analyse des exigences : Ce processus doit permettre la transformation des services désirés en produit désiré.
- Le processus de conception de l'architecture : Ce processus vise à synthétiser une solution qui satisfait aux exigences.
- Les trois processus décrits jusque là sont réitérés récursivement jusqu'à obtenir des éléments qu'on peut acheter, réutiliser ou construire.
- Le processus de réalisation : Il a pour but de produire un élément du système. Il vise à définir les actions de fabrication.
- Le processus d'intégration : Ce processus vise à assembler le système.
- Le processus de vérification : Il permet de confirmer que le système est conforme à la conception et qu'il satisfait aux exigences du système.
- Le processus de transition : Il vise à installer le système de manière à pouvoir fonctionner et fournir ses services.
- Le processus de validation : Ce processus veut prouver que le système fournit bien les services exigés.
- Le processus de mise en service : Il permet d'utiliser le système dans son environnement réel.
- Le processus de maintenance : Il permet au système de continuer à fournir correctement ses services.

- Le processus de fin de vie : Il permet l'arrêt de service du système.

Nous nous proposons d'adapter cette recommandation à la méthodologie que nous présenterons dans cette thèse, en l'adaptant à un processus permettant le prototypage virtuel et la simulation.

2.5 La modélisation en ingénierie système :

2.5.1 La modélisation de la spécification des besoins :

Elle représente l'ensemble des services qu'un système est destiné à fournir dans un certain contexte. Elle doit être précise et compréhensible par le client demandeur du projet et le concepteur. Cette spécification joue le rôle de contrat entre les deux parties. Elle décrit le « quoi » non le « comment ». Cette spécification doit être transformée par les concepteurs en une spécification de la conception. Elle comporte :

- Introduction : Il s'agit de la description du besoin du système ainsi que le contexte dans lequel il devra fonctionner. Elle décrit les fonctions du système et la manière dans laquelle le système est inscrit dans la stratégie et les objectifs de l'entreprise cliente.
- Le modèle du système : C'est le modèle mettant en évidence les relations entre les composants du système et entre le système et son environnement.
- L'évolution du système : Représente les hypothèses sur lesquelles est basé le système et les changements anticipés grâce à une évolution technologique attendue ou des besoins clients. Exemple : (Un PC muni d'un lecteur CD doit être prévu pour évoluer en remplaçant le lecteur CD par un lecteur DVD-D ou en augmentant la mémoire RAM à un certain point).
- Les objectifs du système ou besoins fonctionnels : Ce sont les services offerts par le système, tenant compte des performances en temps et en qualité.
- Les contraintes du système : Les restrictions sur le comportement du système et sur les libertés du concepteur. Ce sont des contraintes sur la manière de réaliser les objectifs. Exemple : sécurité, technologies spécifiques, standards, qualité, maintenabilité, robustesse…

- Les priorités du système : Si des choix entre les objectifs et les contraintes doivent être faits, quelles sont les priorités ?
- L'interfaçage avec l'environnement : Les interfaces d'entrée/sortie et les interactions avec d'autres systèmes.
- Le glossaire : Définition des termes utilisés dans le document.
- Les index : Différents types d'index peuvent être inclus.

2.5.2 La modélisation de la conception

La conception est une description abstraite du système qui explique comment les besoins seront réalisés. Elle est destinée aux concepteurs et ingénieurs. Les objectifs et contraintes exprimées dans les besoins doivent être retraçables dans cette spécification. Une spécification adaptée à l'ingénierie des modèles doit être structurée dans une hiérarchie de modèles conçue pour décrire le système de différents points de vue avec une traçabilité entre les modèles. Une telle structure doit :

- Faciliter le traçage des besoins de niveau système et les contraintes de conception vers la conception détaillée et l'implémentation et vice-versa.
- Assister le concepteur à assurer différentes propriétés du système.
- Réduire le coût de l'implémentation des changements. Ainsi que les coûts de la revalidation lorsque le système est modifié.

Les vues/modèles nécessaires à la conception peuvent être des vues statiques ou dynamiques. Les vues statiques permettent de décrire les caractéristiques, la composition, l'architecture et les interfaces du système. Les vues dynamiques permettent de décrire le comportement du système, les protocoles utilisés et les scénarii possibles**.**

Dans **[SynchTechRT94]** Benveniste et ses collaborateurs expriment le besoin d'un ensemble d'outils intégrés munis d'un formalisme permettant la conception modulaire garantissant un assemblage correct tout en s'abstrayant à la rédaction de la spécification de l'architecture matérielle utilisée. Ils ajoutent que cette plate-forme

devrait fournir les moyens de vérifier des propriétés et d'effectuer des preuves formelles sur les modèles produits. Ensuite, pour les parties non vérifiables formellement, la simulation doit être possible pour apporter des éléments de réponse à la vérification de manière informelle. Ils appuient le fait que la génération automatique de code puisse donner un code performant en termes de rapidité d'exécution et de ressources. Finalement, ils expriment le besoin d'évaluer les performances du code généré en termes de rapidité d'exécution en tenant compte des contraintes temporelles.

Nous devons adapter cette approche pour la conception de systèmes mécatroniques. Ainsi les langages ou méta-modèles utilisés doivent être spécifiques aux systèmes mécatroniques. Ensuite, le code généré représentera un prototype virtuel du système. Ce prototype devrait être testé sur une plate-forme de simulation pour en valider le fonctionnement et les performances.

2.6 Langages et Formalismes multi-technologiques

Pour étoffer notre méthode de modèles multi-domaines ou multi-technologiques, nous avons jugé utile d'intégrer les Bond Graphs **[Gandanegara2003]** et la DSM (Design Structure Matrix) qui sont des modèles indépendants des technologies. Cette intégration a aussi pour but de prouver que la méthodologie que nous nous apprêtons à proposer est extensible et peut ainsi être adaptée à différents besoins que nous avons. Les Bond Graphs puis la DSM seront ainsi présentés dans ce qui suit.

2.6.1 Les Bond Graphs

2.6.1.1 Introduction aux Bond Graphs

Un Bond Graph **[Karnopp2000]** décrit un système physique comportant des composants faisant intervenir des domaines énergétiques multiples **[Favre1997]**. Cette description se fait en termes de composants connectés entre eux par des liens à travers les ports dont ils disposent. Les composants sont classés par le nombre de ports dont ils disposent, ce sont des multiports ou des n-ports comme décrits dans **[Hales2000]**.

Il existe trois types de Bond Graphs utilisés chacun dans une étape particulière du processus de conception **[Zaytoon2001]**. Le premier type est celui des Bond Graphs

à mots où les composants représentent des sous-systèmes décrits par des boites noires, ce niveau permet une première décomposition du système pour avoir une vue globale des échanges énergétiques mis en œuvre. Le deuxième type est celui des Bond Graphs acausaux où les composants sont des composants élémentaires indivisibles et dont le comportement est connu (résistance, tige, condensateur, etc.), ce niveau est utilisé à une étape avancée du processus de conception, où on peut assimiler les composants à des composants élémentaires parfaits. Le dernier type est celui des Bond Graphs causaux qui permet d'établir les équations du système.

2.6.1.2 Les éléments du langage

Dans le formalisme Bond Graphs, un composant est représenté par une ligne nommée fermée qui modélise les frontières du composant. Pour tout échange énergétique avec l'extérieur, on associe à ce composant un port énergétique d'un type donné (mécanique, électrique, etc.). Un port est caractérisé par une semi-flèche unidirectionnelle qui porte les informations de l'énergie transportée, à savoir l'effort et le flux correspondants au domaine énergétique du port. (tableau 2.1).

Tableau 2.1. Effort et flux dans différents domaines énergétiques

Domaine énergetique	Effort e	Flux f
Mécanique Translationnelle	Force	Vitesse
Mécanique de rotation	Couple	Vitesse angulaire
Electricité	Tension	Courant
Magnétisme	Force Magneto-motrice	Flux Magnétique

2.6.1.3 Les composants élémentaires

Les composants élémentaires sont classés par rapport à leurs comportements énergétiques ou leurs fonctions (voir tableau 2.2). En plus de ces éléments de base, chaque élément peut être modulé mis à part I et C. Un élément modulé est noté avec un M précédent son nom tel que MSe et MGy.

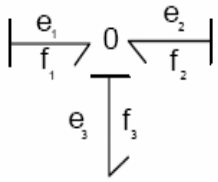
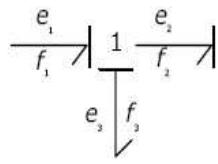
Tableau 2.2. Composants élémentaires en Bond Graphs

Eléments Actifs	$S_e \longrightarrow$	Génération d'effort.
	$S_f \longrightarrow$	Génération de flux.
Eléments passifs	$\longrightarrow R$	Nœud de dissipation d'énergie.
	$\longrightarrow I$	Nœud de stockage d'énergie.
	$\longrightarrow C$	Noeud de stockage de flux.
Capteurs	$\longrightarrow D_f$	Capteur de flux.
	$\longrightarrow D_e$	Capteur d'effort
Eléments de conversion	$1 \xrightarrow{m} TF \xrightarrow{} 2$	$e_1 = m e_2 \; ; \; f_2 = m f_1$
	$1 \xrightarrow{r} GY \xrightarrow{} 2$	$e_1 = r f_2 \; ; \; e_2 = r f_1$

2.6.1.4 Les jonctions

Les jonctions (tableau 2.3) sont utilisées pour associer les composants élémentaires. Ils décrivent la transmission d'énergie de manière instantanée. Ils connectent deux ou plusieurs liens.

Tableau 3. Les deux types de jonctions

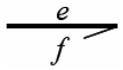
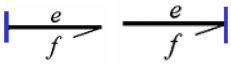
Jonction 0 : tous les efforts sont égaux. Ex: Connexion parallèle en électricité.		$e_1 = e_3$ $e_2 = e_3$ $f_3 = f_1 - f_2$
Jonction 1 : Tous les flux sont égaux Ex : Connexions série en électricité.		$f_1 = f_2$ $f_3 = f_2$ $e_2 = e_1 - e_3$

2.6.1.5 Les flèches ou "Bonds"

En BG, il y a deux types de liens (tableau 2.4); Le premier est un transfert informationnel et est représenté par une flèche unidirectionnelle. Le second est un

transfert énergétique et est représenté par une semi flèche unidirectionnelle. Dans le cas d'un BG causal, une ligne verticale est ajoutée à une des extrémités de la flèche. Les liens énergétiques sont numérotés.

Tableau 2.4. Les bonds ou flèches dans les BG

Transfert énergétique		Transfert informationnel
Pas de causalité	Avec causalité	
		

2.6.2 La DSM ou Design Structure Matrix

2.6.2.1 Introduction aux DSM

Les DSM (Design Structure Matrix ou encore Dependency Structure Matrix) ont été crées par Steward en 1981 **[Steward1981]**. L'équipe DSM Group at MIT **[DSM]** travaille à développer la théorie des DSM. Les DSM sont des matrices carrées ayant pour lignes et colonnes les composants d'un système. L'élément (i,j) de la matrice représente une relation entre le composant de la ligne i avec celui de la colonne j. Ainsi les éléments de la diagonale n'ont aucun sens (figure 2.7).

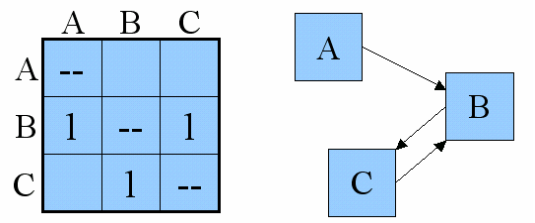


Figure 2.7. Exemple d'une DSM.

Les interdépendances entre les sous-systèmes peuvent être de différents types. Ceci dépend du type du système considéré et du type d'informations décrites **[DSM]**. Il existe ainsi plusieurs types de DSM :

- Les DSM basées sur les composants : Elles peuvent exposer plusieurs types d'interactions (spatiales, énergétiques, informationnelles, échanges de matière, etc.).
- Les DSM dédiées aux équipes de travail : Elles sont utilisées pour les analyses organisationnelles et les flux informationnels entre individus ou groupes participant à un projet.
- Les DSM d'activités : Elles expriment les interdépendances entre les tâches d'un certain processus.
- Les DSM paramétrées : Elles sont construites en découpant un système, en définissant les interactions entre ses composants puis en quantifiant un type d'interactions (proximité spatiale, liaison mécanique, besoin de transferts énergétiques, etc.)

Lors de la description d'un processus, une séquence du temps est associée aux positions des composants dans les entêtes des lignes et colonnes de la matrice. Dans ce cas **[Sharman2004]**, un élément non-nul au dessus de la diagonale implique un feedback et un élément non-nul en dessous de la diagonale représente une dépendance.

Dans **[Browning1998]** Browning utilise les DSM pour représenter un processus, puis effectue quelques transpositions de lignes (et donc de colonnes) de manière à avoir tous les éléments en dessous de la diagonale sinon en rapprochant le maximum les éléments supérieurs vers la diagonale pour obtenir une matrice triangulaire en blocks. Ceci se traduit par le réordonnancement du processus en minimisant les feedbacks.

Comme le souligne Browning **[Browning1998]**, les éléments non-diagonaux ne doivent pas nécessairement être binaires, ils peuvent contenir des informations codant une probabilité, un taux de flux, un type de flux, etc. Sharman et Yassine **[Sharman2004]** distinguent entre une forte dépendance et une faible dépendance.

[Harmel2007] Utilise une description en DSM pour appliquer un algorithme de *clustering* qui lui permet d'identifier le type de l'architecture du produit considéré selon des modèles prédéfinis.

2.6.2.2 Usages actuels de la DSM

Cette représentation de l'architecture d'un système sous forme d'une matrice a pour avantage d'offrir une structure de données bien adaptée aux manipulations algorithmiques (transpositions de lignes et de colonnes, parcours sur les éléments etc.) et aussi aux manipulations mathématiques (additions, multiplications de matrices, recherche de valeurs propres etc.).

En particulier les puissances de la matrice (M^n) peuvent révéler les dépendances cycliques du système. Nous pouvons interpréter l'élément (i,j) d'une DSM par « Le composant de la ligne j est atteignable par le composant de la colonne i en parcourant un seul lien de dépendance ». Pour la matrice carrée (M^2) l'interprétation devient « Le composant de la ligne j est atteignable par le composant de la colonne i en parcourant exactement deux liens de dépendance ». Ainsi, si on élève la DSM à la puissance n, chaque élément non-nul de la diagonale implique que cet élément est atteignable en parcourant n liens de dépendances en partant de lui-même. Ce qui veut dire qu'il intervient dans un cycle de longueur n. Pour illustrer cela, considérons le système suivant (figure 2.8) :

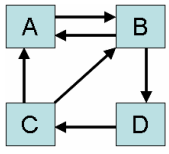


Figure 2.8. Graphe représentant des dépendances
entre composants d'un système.

Le carré de cette DSM qu'on appellera *M* est :
(Sachant que si un élément est supérieur à 1, il est remis à 1)

$M^2 = M \times M =$

	A	B	C	D
A	0	1	0	0
B	1	0	0	1
C	1	1	0	0
D	0	0	1	0

x

	0	1	0	0
	1	0	0	1
	1	1	0	0
	0	0	1	0

=

	A	B	C	D
A	1	0	0	1
B	0	1	1	0
C	1	1	0	1
D	1	1	0	0

Les éléments dans la diagonale de M^2 qui sont en surbrillance montrent que les composants A et B appartiennent à des cycles de longueur 2 que l'on appellera des 2-cycles (un n-cycle sera donc l'appellation adoptée pour un cycle de longueur n).

$M^3 = M^2 \times M =$

	A	B	C	D
A	1	0	0	1
B	0	1	1	0
C	1	1	0	1
D	1	1	0	0

x

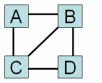
	0	1	0	0
	1	0	0	1
	1	1	0	0
	0	0	1	0

=

	A	B	C	D
A	0	1	1	0
B	1	1	0	1
C	1	1	1	1
D	1	1	0	1

En calculant M^3, nous découvrons que les composants B, C et D sont impliqués dans des 3-cycles et que le composant A n'appartient à aucun 3-cycle.

Il est à noter que l'utilisation excessive de liens non-orientés (ou bidirectionnels) donne lieu à des matrices inexploitables. Considérons l'exemple suivant :



$M^2 = M \times M =$

	A	B	C	D
A	0	1	1	0
B	1	0	1	1
C	1	1	0	1
D	0	1	1	0

x

	0	1	1	0
	1	0	1	1
	1	1	0	1
	0	1	1	0

=

	A	B	C	D
A	1	1	1	1
B	1	1	1	1
C	1	1	1	1
D	1	1	1	1

Le carré de la matrice de départ contient le nombre 1 dans tous les éléments diagonaux tant que le composant en cause possède un lien bidirectionnel sortant ou entrant. Car chaque lien bidirectionnel est un 2-cycle pour les deux composants reliés.

2.7 Conclusion

Nous avons présenté dans ce chapitre quelques méthodologies utilisées en conception. Ensuite nous avons présenté quelques processus standards d'ingénierie système. Nous avons rappelé l'importance de ces approches systémiques pour la conception des systèmes mécatroniques. Ensuite, nous avons présenté deux formalismes

de représentation multi-technologiques. Après avoir décrit ce cadre, nous rappelons les objectifs posés dans la problématique, en particulier, notre tentative d'appliquer des approches du génie logiciel, en l'occurrence l'ingénierie guidée par les modèles à la conception systémique des systèmes mécatroniques. Pour cela, nous allons présenter les bases théoriques qui ont permis, en génie logiciel, d'entreprendre de telles approches. Nous commençons, donc, dans ce qui suit, par nous focaliser sur les langages orientés objet, puis par présenter l'approche MDA qui établit un cadre général pour entreprendre une ingénierie système guidée par les modèles (ou MBSE : Model-Based Systems Engineering).

Chapitre 3

III - Quelques approches orientées objet

3.1 L'approche Objet

L'objet est un concept qui a été inventé pour regrouper dans une même entité un ensemble de procédures portant sur un ensemble de données. Ceci avait pour but de simplifier la programmation impérative classique en intégrant une approche en composants qui permet la réutilisation à la manière de ce qui se fait en électronique par exemple. Sont apparus alors des langages orientés objet tels que C++ ou SmallTalk. Dans la modélisation objet le concepteur analyse le système, son environnement et son domaine de connaissances puis représente par des objets ces éléments du monde réel ou des concepts abstraits. Les objets ainsi décrits sont les types des données qui seront utilisés lors des activités de conception orientée objet. Durant cette analyse le concepteur ne prend en compte aucun aspect d'implémentation, seul le champ du problème est important. C'est principalement cette particularité de l'analyse orientée objet OOA qui en fait une méthode extensible à tous les autres domaines non logiciels. Quant aux activités concernant la solution envisagée qui représentent la conception (Design) orientée objet ou OOD, les langages utilisés sont souvent orientés vers des domaines particuliers et peuvent viser spécifiquement une plate-forme d'implémentation donnée. Après les objets, les frameworks ont vu le jour (framework .NET, NetBeans). Un framework est un atelier contenant toute une panoplie d'objets qui permettent de résoudre un ensemble de problèmes liés. La dernière évolution (figure 3.1) est celle du tout « modèle ». Nous présenterons cette approche plus tard dans ce document. Les trois dernières évolutions utilisent toujours à leurs bases l'approche orientée objet.

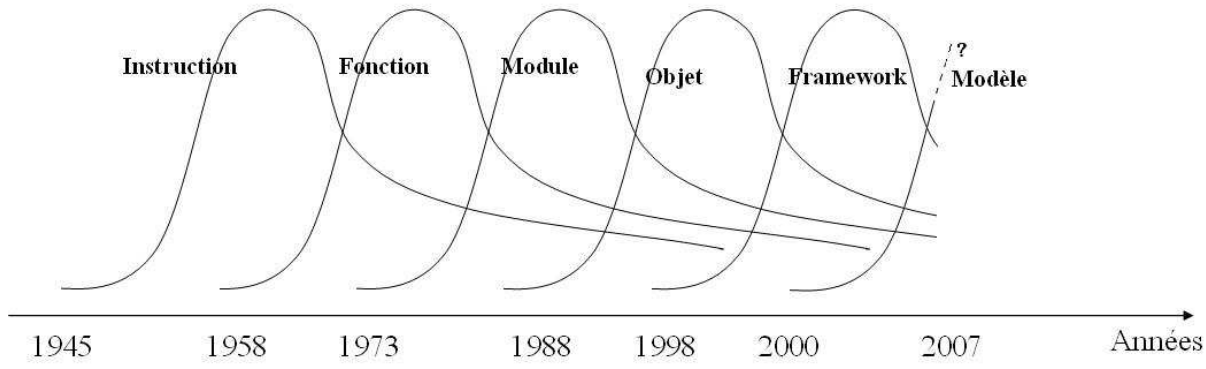


Figure 3.1. Vagues de l'évolution des technologies du logiciel.

Nous allons présenter le langage UML et quelques langages orientés-objets utilisés pour la conception de systèmes hétérogènes. UML est un langage qui a tenté de résoudre la problématique de la couverture (ou universalité) en offrant des éléments de modélisation qui puissent satisfaire à différents domaines :

3.2 Le langage UML

3.2.1 Historique

UML est le « Unified Modeling Language ». L'élaboration d'UML a commencé en 1994 par Grady Booch et Jim Rumbaugh qui ont entrepris d'unifier la méthode Booch et OMT **[UML1.4]**. En 1995 la version 0.8 d'UML a été publiée. En 1995, Ivar Jacobson rejoint l'équipe et sa méthode OOSE est fusionnée avec UML. Cette collaboration avait pour but de donner une certaine stabilité aux technologies orientée-objet en fournissant un langage unifié et plus riche que l'existant ce qui permettait aux éditeurs logiciels de fournir des outils qui pourront être utilisés par un public encore plus large que pour un outil ne supportant que la méthode OMT par exemple. Ces trois pionniers de la technologie objet se sont fixé quatre objectifs :

- Permettre la modélisation de systèmes hétérogènes pas seulement des systèmes logiciels en utilisant des concepts orientés-objets.
- Etablir un couplage entre les artefacts conceptuels et/ou architecturaux et les artefacts exécutables ou comportementaux.
- Fournir des éléments permettant d'adresser des systèmes complexes à objectifs critiques (dont le dysfonctionnement peut causer des pertes en termes de vies humaines, pertes économiques ou autre).
- Permettre au langage d'être manipulé par des intervenants humains ou par programme.

Cette collaboration a donné lieu à UML 0.9 en 1996. Ensuite l'OMG (Object Management Group) **[OMG]** a rassemblé un groupe de travail composé d'industriels du logiciel pour élaborer et standardiser UML. La version UML 1.0 a ainsi vu le jour puis la version 1.1 qui a été adoptée en 1997. L'évolution d'UML a continué depuis sous

l'égide de l'OMG. La version 1.3 a été publiée en 1999 et représentait une évolution mineure du langage. Ensuite la version 1.4 a été adoptée en 2001. En 2005 la version 2.0 qui est une réécriture du langage est adoptée.

3.2.2 Caractéristiques

Sémantique informelle : Les éléments du langage ne sont pas définis sur des bases mathématiques (Backus-Naur Form), il est ainsi impossible de vérifier mathématiquement la cohérence sémantique d'un diagramme UML. De plus, si nous avons recours à deux types de diagrammes différents, pour décrire le même système sous deux vues différentes, il est impossible de vérifier formellement la consistance du modèle global. Pour l'étape de réalisation, il est impératif de dépasser cette lacune. Une solution est de créer des profiles de langages formels.

UML 2 a été conçu en tenant compte de quatre principes globaux. **[UMLInfra 2004]** :

- La modularité et la réutilisation : Les éléments du langage sont regroupées dans des packages dont la cohésion est maximisée et dont l'interdépendance (ou couplage) est minimisée. Cette modularité améliore la réutilisation des packages de l'infrastructure ainsi que de la superstructure.
- Une architecture en couches : Elle est réalisée selon deux aspects. Le premier est l'hiérarchie des packages qui sépare les éléments de base des éléments de haut niveau qui les utilisent. Le deuxième aspect est visible dans l'architecture en quatre couches qui sera présenté avec plus de détail dans le paragraphe « architecture actuelle d'UML ».
- Le partitionnement : Les packages de l'infrastructure d'UML sont partitionnés de manière à faciliter la création de nouveaux langages à partir du méta-langage d'UML. Les packages de la superstructure sont partitionnés de manière modulaire.
- L'extensibilité : Le partitionnement et la modularité favorisent l'extension par création de nouveaux langages basés sur l'infrastructure d'UML. De plus, des mécanismes d'extension permettent l'adaptation d'UML à des domaines

particuliers. Ces mécanismes seront présentés avec plus de détail dans le paragraphe « Qu'est-ce que la méta-modélisation ?».

3.2.3 UML n'est pas une méthode

Contrairement à ses ancêtres (OMT, Booch et OOSE) UML n'est pas directement associé à une méthode, c'est seulement un langage. Le processus qui met en œuvre la boîte à outils UML n'est intentionnellement pas spécifié. UML se veut adaptable à tous les processus pouvant susciter l'intérêt suivant le domaine d'application et la nature du projet. Par contre, la nature des diagrammes UML suscite un certain enchaînement logique. En effet le diagramme des Use Cases par exemple qui permet de décrire les fonctions que doit réaliser le système est naturellement utilisé comme point de départ.

3.2.4 L'architecture d'UML2

UML est décrit par un méta-modèle (figure 3.2). Le méta-modèle d'UML permet de décrire les éléments du langage UML et la structure qui les relie, ce sont les règles syntaxiques du langage.

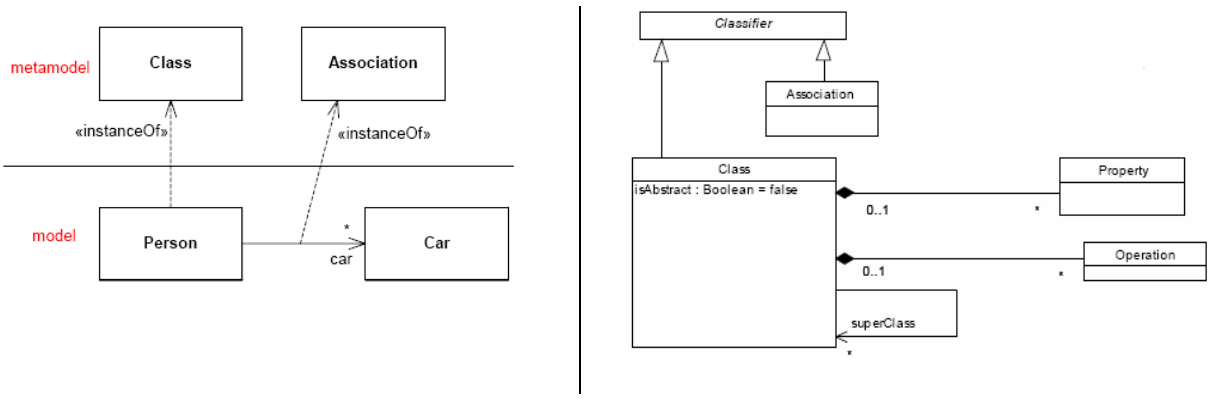


Figure 3.2. Exemples de méta-modèles.

Le MOF (Meta Object Facility) **[MOF2006]** est le méta-modèle d'UML. L'OMG **[OMG]** a établi une architecture à 4 couches pour représenter les niveaux d'abstractions de langages (figure 3.3). La couche M3 représente le méta-métalangage. La couche M2 représente le métalangage. La couche M1 représente un modèle et la couche M0 représente une application. Ainsi, dans ce modèle, la classe du niveau M3

est une méta-méta-classe. La classe du niveau M2 est une méta-classe, c'est l'élément de langage qui représente la classe UML dont l'instanciation permet de créer une classe particulière (niveau M1) d'objets (niveau M0).

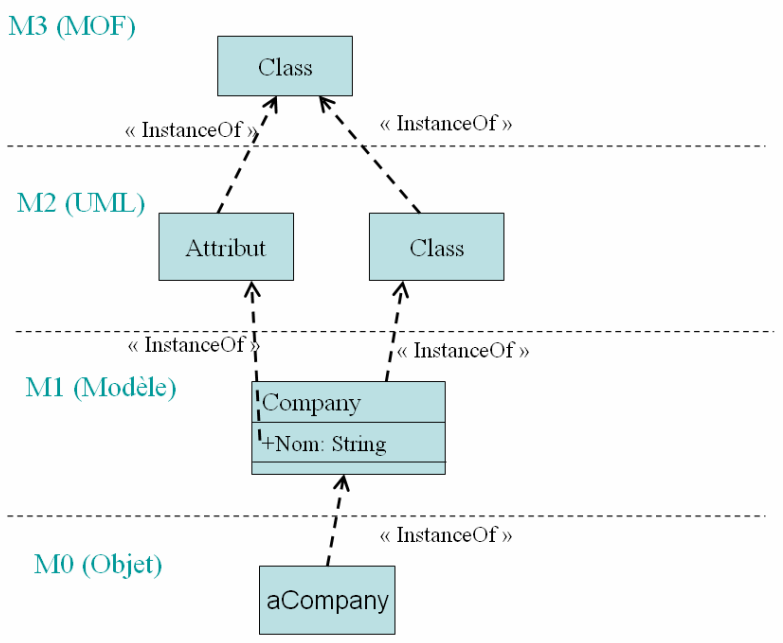


Figure 3.3. Exemple du modèle en 4 couches de la méta-modélisation de l'OMG.

Le MOF a été pensé non-seulement pour permettre de décrire le langage UML mais aussi d'autres langages graphiques (figure 3.4). C'est à partir du MOF que d'autres langages ont ainsi été définis tel que CWM (Common Warehouse Metamodel) **[CWM2001].**

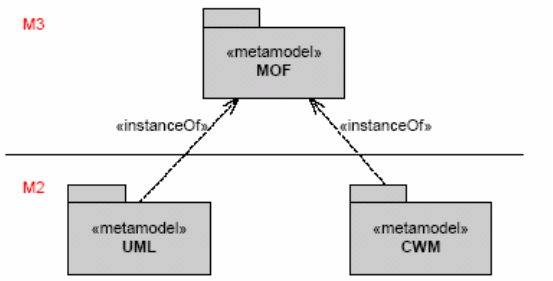


Figure 3.4. Niveaux M3 et M2, MOF et UML. **[UMLInfra2004]** page14.

L'OMG a organisé les méta-modèles des langages UML, MOF **[MOF2006]**, CWM[12] **[CWM2001]** et « Profiles » autour d'un noyau commun appelé « Common

[12] CWM ou Common Warehouse Metamodel est un metamodèle d'échange de méta-données entre les langages de datawarehousing, business intelligence et gestion de connaissances et technologies web.

Core » (figure 3.5). Ce noyau est un méta-modèle qui regroupe des éléments réutilisables de langages basés sur le MOF. Ses éléments sont réutilisés et/ou étendus pour former des langages tels qu'UML ou encore le package « Profiles ».

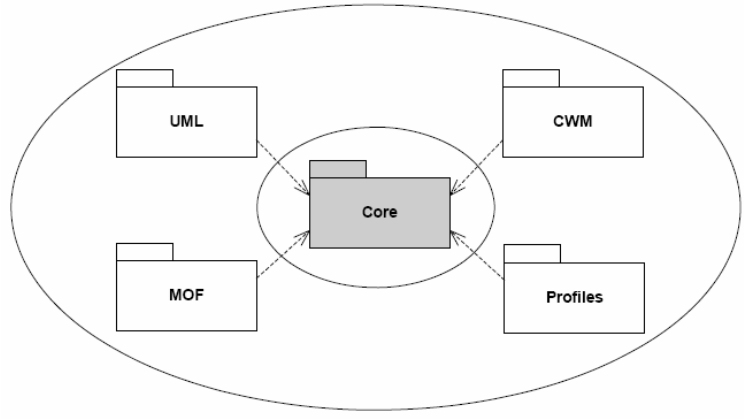


Figure 3.5. Liens entre le noyau « Common Core », les profiles et UML.

3.2.5 UML est un langage extensible : apports des profiles

Pour maintenir le caractère universel d'UML, les méta-modèles ne sont pas modifiables. En contrepartie UML offre des mécanismes d'extension décrits par le chapitre intitulé « profiles » de la spécification de la superstructure d'UML. En UML, parmi les tâches de méta-modélisation, la création de profiles est l'activité qui permet d'étendre ce langage général. Un profile UML réalise le rapprochement et la spécialisation d'UML à un domaine spécifique ce qui permet :

- Meilleure compréhension des utilisateurs.
- Optimisation des possibilités du langage.
- Accélération du processus de conception (méta-classes pré-conçues).
- Meilleure génération de code et de documents.
- Capitalisation du savoir-faire dans un méta-modèle, pérennité.
- Bénéfice des outils UML existants.

Dans le paragraphe suivant nous présentons en détail ces mécanismes tels que décrits par la spécification de l'infrastructure d'UML 2.1.1 issue en février 2007 **[UML2007]**.

3.2.6 Le package UML « Profiles » V2.1.1

Le package « Profiles » **[UML2007]** contient les mécanismes permettant d'étendre les métaclasses d'un métamodèle donné afin d'y intégrer une sémantique spécifique à un domaine ou à une plateforme particuliers. Le Package « Profiles » est aussi un package du noyau (Core ::Profiles). Bien qu'il ait été conçu au départ pour étendre UML, il en est indépendant et peut être réutilisé pour des langages basés sur le Core.

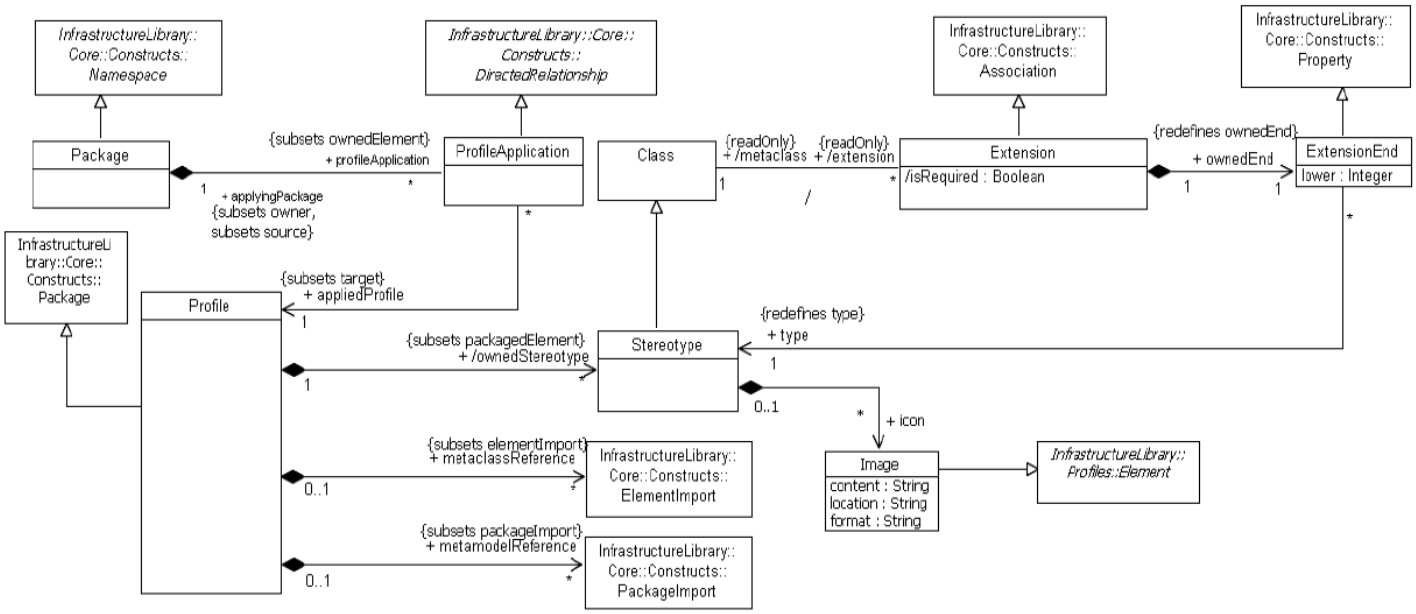


Figure 3.6. Méta-modèle du package « Profiles ».**[UML2007]**

Dans ce méta-modèle (figure 3.6), un profile peut être appliqué à un package à travers une application de profile « ProfileApplication ». Un profile contient un ensemble de stéréotypes qui contiennent des icônes. La méta-classe « Class » peut être dérivée par des stéréotypes. Seule la méta-classe « Stereotype » ne peut pas être stéréotypée, cette contrainte n'apparaît pas sur le méta-modèle de la figure 3.6 mais elle est exprimée textuellement dans la spécification **[UML2007]**. Cette limitation intentionnelle permet d'éviter un stéréotypage hiérarchisé trop lourd. Par contre, le stéréotypage hiérarchisé permettrait de mieux organiser un profile donné et de mettre en place des contraintes relatives à l'application des stéréotypes de manière structurelle non déclarative, ce qui est plus naturel dans la conception d'un profile (dans toute la

conception orientée-objet). L'exemple de la figure 3.7 suivante permet d'illustrer cette problématique :

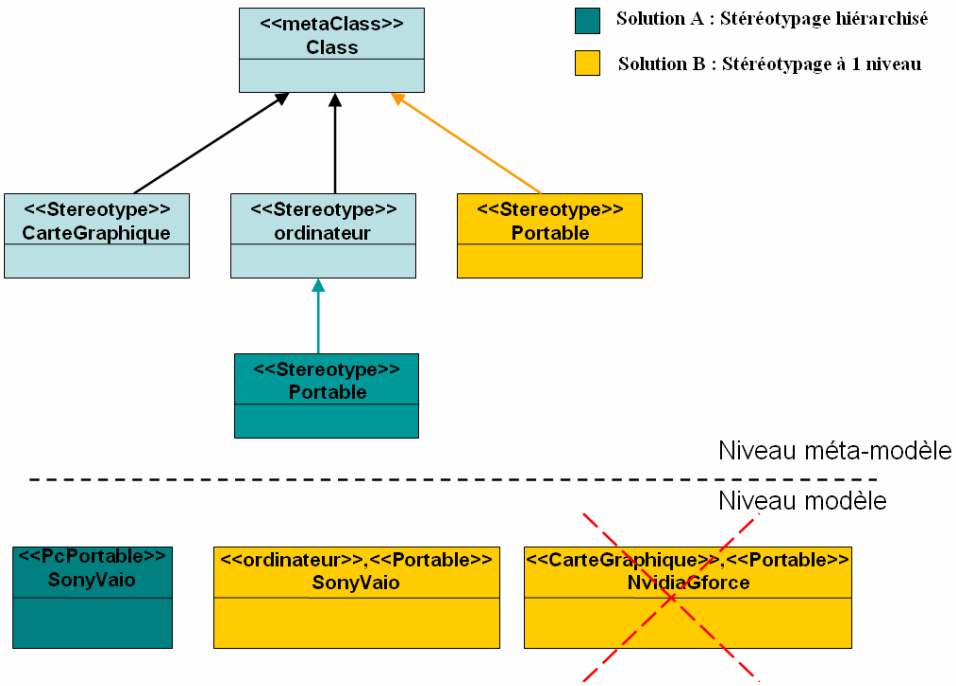


Figure 3.7. Avantage du stéréotypage hiérarchisé.

Dans cette figure (3.7) la solution A qui correspond au stéréotypage hiérarchisé a deux avantages essentiels :

- Pour un ordinateur portable, la solution B nous oblige à spécifier deux stéréotypes (<<ordinateur>> et <<portable>>) ; Si un traitement automatique est fait sur tous les ordinateurs, le stéréotype <<ordinateur>> est nécessaire. Quant à la solution A, elle nous permet de se contenter de spécifier le stéréotype le plus spécialisé et dans ce cas un traitement automatisé reconnaîtrait un portable comme étant un ordinateur. Du point de vue graphique, la solution A permet aussi un gain d'espace considérable ce qui est un point primordial dans la conception graphique ; Avoir un schéma moins encombré permet de mieux concevoir et mieux comprendre.
- Le deuxième point important est que la solution B permet de faire des erreurs de conception en utilisant n'importe quelle combinaison de stéréotypes même si, comme la figure ci-dessus le montre, dans certains cas, des stéréotypes peuvent

être mutuellement exclusifs. Cette contrainte doit alors être spécifiquement déclarée, car la structure du profile ne l'impose pas.

3.2.7 Etendre UML Vs Créer un DSL

Pour avoir un langage de conception pour un domaine donné, deux approches sont à envisager ; La première est de créer un DSL (Domain Specific Language) ou langage spécifique à un domaine. Utiliser cette approche a certains avantages. En effet, la création d'un langage à partir de zéro pour un certain domaine donnerait un langage parfaitement adapté à ce domaine contrairement à la deuxième solution qui reprendrait un langage existant ayant un large domaine d'expression en l'étendant au domaine. La première solution peut être réalisée en mettant en œuvre le MOF qui joue le rôle de méta-métalangage et qui permet de définir des langages à partir d'éléments généraux communs aux langages graphiques. La deuxième solution est mise en œuvre grâce aux profiles et a pour avantage de profiter de toute la panoplie d'outils UML avec toutes les fonctionnalités qu'ils implémentent tel que l'ergonomie des éditeurs de diagrammes, le reverse engineering, la génération personnalisée de code etc. Nous présentons dans ce tableau les avantages et inconvénients de chaque solution :

Tableau 3.1. Avantages et inconvénients de la création de profiles et de DSLs.

	Extension d'UML, avantages et inconvénients:		Conception d'un DSL, avantages et inconvénients:
+	Standard, outils disponibles, diversité des diagrammes, les diagrammes sont une base de travail pour la création de nouveaux diagrammes, documentation du langage, maturité.		adéquation au domaine (artefacts découlent du domaine), pas d'informations ou artefacts superflus, liberté de description, choix des diagrammes illimité (les limites sont celles du méta modèle).
–	Limites pour l'invention de nouveaux diagrammes, complexité du langage, Extension : plus de complexité, artefacts obsolètes pour certains domaines et donc complexité inutile.		Outil à construire (il existe des outils permettant cela tels que eclipse/EOF /GEF, Microsoft Software Factories, metacase+ etc.), définition des diagrammes à partir de zéro.

Ainsi on ne doit avoir recours à la première solution que si on constate que la nature du langage UML est complètement inadéquate au langage cible. Dans ce cas là il existe des solutions qui permettent de construire un outil en réutilisant des mécanismes génériques tel que Eclipse Modeling Framework (EMF) et Graphical Editing Framework (GEF) plug-ins de la plate-forme eclipse **[Eclipse Project]**, ou encore les DSL Tools de Microsoft **[DSL_Tools]**.

Pour répondre à l'un des inconvénients décrits dans le tableau 3.1, nous proposons un mécanisme d'extension ; le Blinder (figure 3.8) permettant de réduire un méta-modèle visuellement. Ce Blinder, lorsqu'il est appliqué à un élément du méta-modèle, veut dire que cet élément est inutile pour un profile donné et que l'outil doit cacher cet élément à l'utilisateur. Cette proposition n'élimine pas complètement l'élément du langage du modèle de données, ce qui permet de garder la compatibilité des outils. Ce mécanisme est une proposition purement théorique et indépendante de la méthode.

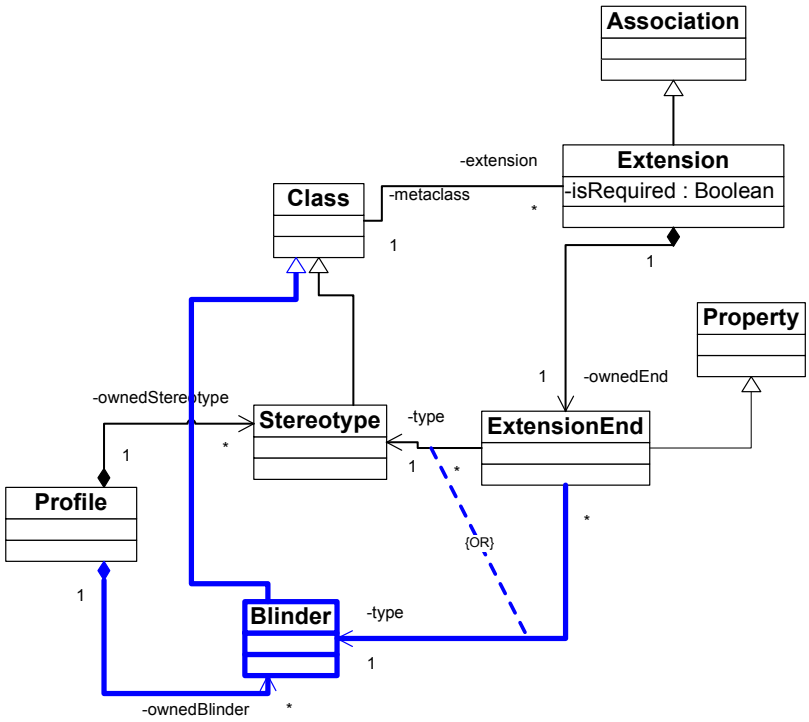


Figure 3.8. Mécanisme pour cacher des éléments obsolètes du langage.
(Les parties en trait fort sont notre contribution)

3.3 Extensions d'UML pour systèmes hétérogènes et Profiles existants

3.3.1 UML et les systèmes hétérogènes

UML a été appliqué à plusieurs types de systèmes hétérogènes. Dans **[Nguyen2004]** les auteurs présentent une extension au diagramme de classe et au diagramme d'états d'UML permettant de concevoir un système temps-réel et de générer le code SystemC[13] correspondant. Les diagrammes de classe permettent une décomposition hiérarchique du système ainsi que l'établissement du squelette du code SystemC qui sera généré en aval. Les diagrammes d'états décrivent le comportement d'une classe. Les actions, qui sont exécutées pendant chaque état sont décrites en code C++.

Dans **[Caraa2004]** les auteurs utilisent UML pour modéliser des circuits analogiques en empruntant une approche systémique. Ils réalisent un mappage des concepts de VHDL-AMS vers des éléments de modélisation UML pour permettre la description du matériel et du logiciel sur une même plateforme. Ils utilisent le diagramme de classes pour décrire la structure du système, les composants et leurs interconnexions. Les entrées, les sorties et les valeurs génériques sont représentés par des attributs de classe et une fonction de transfert est utilisée pour décrire le comportement dans la partie opérations de chaque classe. Ceci est valable pour les composants qui peuvent être représentés par un modèle mathématique. Une classe nommée « system » est finalement utilisée pour décrire le système global et connecter les différents composants, cette connexion est utilisée pour réaliser le mappage entre les ports dans le code VHDL-AMS.

Dans **[Dudra2003]** l'auteur expose une application d'UML pour concevoir un système d'amortisseurs d'un camion. Il utilise pour cela dans des phases d'analyse le diagramme Use-case pour définir les liaisons entre les fonctions du système avec les acteurs du monde extérieur au système. Ensuite il décrit un diagramme de collaboration qui lui a permis de mettre en évidence un certain nombre de classes (représentant des

[13] SystemC est une librairie écrite en C++ largement utilisée pour écrire du logiciel temps-réel.

modules du système) et l'échange séquentiel des messages entre elles tout en faisant intervenir les acteurs extérieurs au système. Il expose ensuite un diagramme des classes pour définir l'ensemble des classes nécessaires pour construire le système et les relations qui les lient en détaillant les fonctions (méthodes) de chaque classe. Cette approche est connue par l'approche orientée par les Use Cases (Use Case Driven Design).

D'autres travaux ont introduit dans UML des modèles pour la vérification tel que les réseaux de Petri **[Paludetto2004].** Dans un projet antérieur **[TURKI2004]** nous avons montré comment un PSM décrit en UML pouvait être utilisé pour décrire graphiquement un système à comportement hybride puis être simulé sur la plate-forme de simulation OpenMask[14].

D'autres qualités d'UML ont été démontrées par d'autres travaux. Dans **[Bahill2003],** par exemple, les auteurs utilisent UML dans toute la démarche de conception en partant de l'expression des besoins vers l'analyse du système et finalement la conception du système. Cet article montre les qualités d'UML pour supporter les tâches de l'ingénierie système. Les auteurs déclarent que les outils classiques de l'ingénierie systèmes (RDD-100, DOORS, Slate, RTM, Excel etc.) sont dépassés et ne permettent pas de faire communiquer les ingénieurs du logiciel et les ingénieurs électroniciens. Les ingénieurs système utilisent encore le processus « waterfall » à cause de ces outils. Bahill et Daniels **[Bahill2003]** expriment la nécessité aux ingénieurs système d'évoluer vers les outils/méthodes utilisés dans le génie logiciel (UML, RUP etc.). Ils illustrent leur utilisation d'UML par la conception d'un système de chauffage ventilation et air conditionné en mettant en œuvre le processus unifié pour l'ingénierie système (Unified Systems Engineering Process) **[Bahill1998]** qui est une évolution du « Unified software development process » **[Jacobson1999].**

[14] OpenMask est une plateforme de réalité virtuelle Open Source développée par l'INRIA : http://www.irisa.fr/bunraku/OpenMASK/

Ainsi UML est tout aussi adapté pour décrire des modèles exécutables de systèmes mécatroniques mais aussi pour tout le processus de conception de systèmes mécatroniques. L'ensemble de ces expériences ont montré les possibilités d'UML et son utilité en tant que langage de base pour le MBSE des systèmes complexes et/ou hétérogènes.

3.3.2 Lacunes d'UML par rapport aux systèmes physiques

Par contre, ces expériences ont montré que certaines faiblesses d'UML devaient être comblées pour en faire un langage efficace à l'ingénieur système. Nous en citons :

- Besoin de décrire les exigences dans le modèle UML, et d'en assurer la traçabilité vers la conception.
- Besoin de représenter des éléments non-logiciels et d'en spécifier le type (mécanique, circuit, hydraulique, câblage, capteur…).
- Besoin de représenter des attributs de performance, des attributs physiques et non comportementaux.
- Types de données pour des éléments physiques.
- Éléments de modélisation explicites pour représenter des entrées/sorties physiques.
- Représentation des attributs continus (fluides, énergie..).
- Sémantiques limitées pour spécifier des événements et coupler l'événement avec l'action.

Beaucoup de tentatives ont été développées durant ces quinze dernières années pour répondre à ces besoins comme UML RT, UML/PNO **[Paludetto2004Petri]** et des profiles UML ont été définis tels que MARTE **[MARTE2007]**, OMEGA **[UMLOmega]** TURTLE **[Apvrille2004]** et SysML **[SysML2007]**.

3.3.3 Le profile MARTE

MARTE **[MARTE2007]** est le profile UML2 pour la *M*odélisation et l'*A*nalyse de systèmes Temps-*R*éel et *E*mbarqués. Il succède à *UML Profile for Schedulability,*

Performance and Time **[UML SPT 2005]**. MARTE est conçu de manière à permettre l'utilisation de techniques d'analyse quantitative variées. La figure 3.9 suivante décrit le méta-modèle de description structurelle des composants dans MARTE.

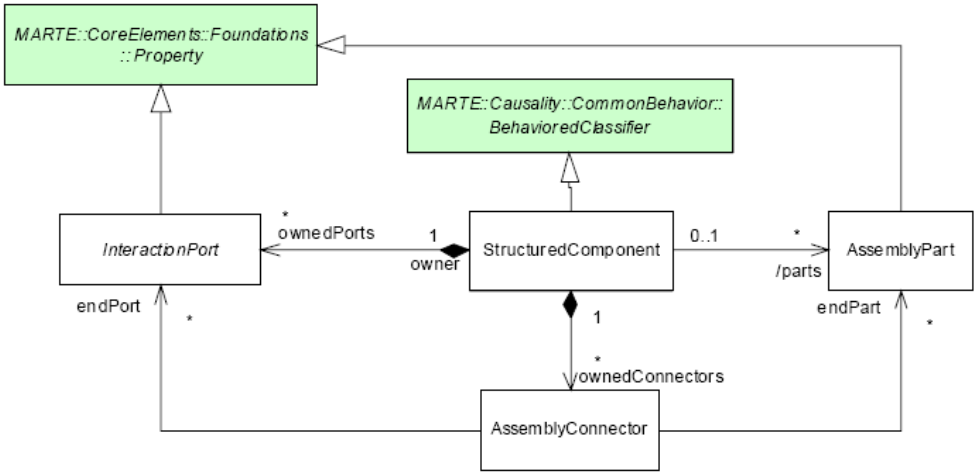


Figure 3.9. Méta-modèle de description structurelle
des composants en MARTE **[MARTE2007]**.

Un *StructuredComponent* définit dans MARTE une entité encapsulée contenant des données structurées et une description comportementale. Ce composant peut contenir des *InteractionPort* qui définissent des points d'interaction à travers lesquels le composant peut être relié par des *AssemblyConnector*. Le composant peut aussi être connecté directement sans passer par les ports. Les ports sont étendus pour supporter les échanges orientés-messages et ceux orientés-flux. Dans les aspects structurels, ce profile reprend les mêmes concepts que nous retrouverons dans SysML dans le chapitre suivant.

Au-dessus de cette couche de description de l'architecture, une couche de description des exigences non-fonctionnelles et temporelles est définie. Le package MARTE *design model* contient ces éléments. Il définit des mécanismes décrivant le temps tels que les événements temporels et les horloges. Ainsi MARTE supporte trois modèles temporels différents ; le temps réel (chronométrique), le temps logique, et le temps logique synchrone.

Une autre couche représentée par le package MARTE *analysis model* est destinée à fournir des éléments d'annotation des modèles permettant l'analyse des propriétés du système décrit **[Gérard2007]**. MARTE propose aussi un modèle de description des plateformes d'exécution, qui est un élément essentiel en systèmes temps-réels embarqués. Ainsi le *General Ressource Model* ou GRM présente des éléments de modélisation de ressources à un très haut niveau comme les sémaphores ou les processus par exemple.

Le langage AADL **[AADL2006]** (Architecture Analysis and Design Language) est un langage de conception d'architectures standardisé par la SAE (Society of Automotive Engineers). Il est destiné à la conception et l'analyse de systèmes embarqués complexes et temps-réel. Le langage AADL est orienté composants, il en expose dix catégories **[Delanote2007]**, il y a des composants de description de la plateforme d'exécution, des composants de description du logiciel applicatif et un composant de description de la composition. Les composants de description de la plateforme d'exécution sont Processor, Memory, Device et Bus. Les composants de description du logiciel applicatif sont Process, Thread, Thread Group, Subprogram et Data category. Finalement, le composant permettant la composition est System. AADL définit pour un composant un type et une implémentation. Le type représente son interface. L'implémentation représente sa structure interne en termes de sous-composants et leurs interconnexions. AADL est spécifique aux systèmes embarqués et qui couvre l'architecture seulement, une annexe comportementale a été définie pour lui apporter la possibilité de décrire les communications inter-process par exemple. Le profile MARTE peut être utilisé comme un PIM (Platform Independent Model) pour AADL. AADL peut être ainsi utilisé comme un pont vers SystemC.

3.3.4 Un profile pour SDL

SDL (Specification and Description Language) **[SDL]** est un langage graphique formel qui a été standardisé par l'ITU (International Telecommunication Union). Il est destiné à la description des systèmes temps-réels événementiels complexes. Les vues décrites par SDL sont :

- L’architecture : L’architecture d’un système est décrite en SDL par l’interconnexion d’un ensemble de blocs. Chaque bloc est décomposable en un ensemble d’autres blocs.
- Le comportement : Description de processus qui contiennent des procédures.
- La communication : Canaux de communication, signaux.
- Les données : Données typées, données partagées.

En SDL un block est défini par « Le bloc est un agent contenant un automate à états et éventuellement, un ou plusieurs processus ou blocks. L’automate à états d’un block est interprété en parallèle avec ses agents contenus. » **[SDL2000]**

SDL a une sémantique formelle contrairement à UML qui définit des *semantic variation points* (pour un élément de langage, dire qu’il peut être interprété différemment). Par contre SDL ne couvre pas les exigences et ne peut être intégré à d’autres langages que de manière propriétaire, pour cette raison la création du profile UML for SDL (standard Z.109) **[SDL]** permet l’utilisation de tous les diagrammes UML et de profiter de la sémantique formelle de la conception SDL.

3.3.5 Le profile TURTLE

Le profile TURTLE **[Apvrille2004]** a été développé pour combler certaines lacunes d’UML, en l’occurrence l’expression de la communication et de la synchronisation des logiciels embarqués temps-réel. Ce profile introduit le concept de Gate (ou porte) semblable au MessagePort dans MARTE à travers lequel les classes communiquent. La description du comportement d’une classe TURTLE ou TClass est fait par un diagramme d’activités étendu par des opérateurs exprimant des contraintes temporelles et d’autres opérateurs de synchronisation, cette partie du langage est comparable à l’annexe comportementale (Behavioral Annex) du langage AADL. Le profile TURTLE-P **[Apvrille2006]** introduit en plus de cela la possibilité de décrire le déploiement physique des composants logiciels. L’aspect formel de TURTLE lui confère la possibilité de réaliser des tests de vérification tout au long de la conception et des tests de validation par rapport aux exigences.

3.3.6 Le profile OMEGA

Le profile UML Omega **[UMLOmega]** a été développé pour la conception des systèmes embarqués temps-réels. Il isole un sous-ensemble d'UML et lui définit une sémantique formelle. Les exigences sont décrites par des uses cases, des diagrammes de séquences étendus, des contraintes OCL et des machines d'états. Les composants, ports et interfaces permettent la conception de l'architecture. Le profile permet aussi de décrire les politiques d'ordonnancement, l'utilisation des ressources et autres aspects temps-réels.

3.3.7 Le profile SysML

La conception orientée objet a de plus en plus intéressé les ingénieurs système, et le langage UML (Unified Modeling Language) **[UML2007]** en particulier grâce à la place qu'il a conquise dans le monde du logiciel et aussi grâce aux possibilités d'extension qu'il offre. SysML (Systems Modeling Language) **[SysML2007]** a ainsi vu le jour en tant qu'extension du langage orienté-objet UML pour couvrir toutes les étapes de conception de systèmes complexes et hétérogènes. SysML résout principalement les lacunes des autres profiles quant aux phases amont de l'ingénierie système (exigences) et la traçabilité de ces exigences à la conception. Il est présenté avec plus de détails dans le chapitre suivant.

3.3.8 Conclusion

UML n'est donc pas suffisant pour supporter un processus d'ingénierie système de systèmes mécatroniques. Par contre ces profiles permettent de combler certaines lacunes d'UML. Ainsi, pour obtenir un ensemble complet de modèles permettant de supporter tout le processus de conception, nous pouvons articuler certains de ces profiles autour d'un processus d'ingénierie système couvrant toutes les activités de conception. Vu le chevauchement de ces profiles, et le fait qu'ils sont incomplets par rapport aux systèmes mécatroniques, nous avons préféré ne pas nous restreindre à ces modèles mais nous en proposons d'autres que nous trouvons appropriés à la description de l'architecture de systèmes hétérogènes tels que les systèmes mécatroniques.

Chapitre 4

IV - Le langage SysML

4.1 Le langage SysML : Une approche systémique

L'OMG ou *Object Management Group* **[OMG]** qui s'occupe de la standardisation de certaines technologies objet telles qu'UML **[UML_OMG]** et CORBA **[CORBA_OMG]** a émis une demande de propositions (Request For Proposal ou RFP) **[UMLSE RFP 2003]** pour un profil UML destiné à couvrir les tâches de l'ingénierie système. Suite à cette RFP, SysML a été proposé par le International Council on Systems Engineering (INCOSE) **[INCOSE]**. Dans cette RFP, le besoin d'un langage standard destiné aux activités d'ingénierie système a été exprimé. Ce langage devait être facilement intégrable dans les équipes d'ingénierie système et dans les outils existants. Un autre besoin était de faciliter la communication entre des équipes hétérogènes. Dans cette optique, SysML a été conçu comme une extension à UML pour minimiser les difficultés de son implémentation dans les outils UML existants. Sa spécification a été écrite de manière à être abordable contrairement à celle d'UML pour ne pas constituer un frein à son adoption dans le monde de l'ingénierie système. SysML bénéficie donc des mécanismes d'extension d'UML qui sont les stéréotypes, les tagged values (ou méta-propriétés) et les contraintes. Ces mécanismes peuvent encore être utilisés pour spécialiser davantage SysML et en faire une boite à outils de description de systèmes de domaines particuliers (Aéronautique, automobile, énergie, etc.). La première spécification finalisée de SysML a vu le jour en juillet 2006 et l'adoption de SysML a été accomplie en septembre 2007 **[SysML2007]**. Les premiers outils CASE à proposer le support de SysML sont Artisan Software, No Magic, Telelogic, et en open-source TOPCASED **[TOPCASED]** et PAPYRUS **[PAPYRUS]**. D'un autre côté, l'adoption par des ingénieurs systèmes de cette nouvelle technologie est la question clé pour l'avenir de SysML. Nous pensons que SysML doit avoir de sérieux arguments pour réussir cela. Et nous pensons que la solution réside dans l'ingénierie orientée par les modèles ou MDD (Model Driven Development). Cette nouvelle technologie a aussi été adoptée par l'OMG sous le nom de MDA (Model Driven Architecture). Cette architecture MDA permet d'une part de supporter une approche systémique de conception et d'autre part d'intégrer l'utilisation de l'outil CASE avec les outils des experts de domaines en créant une

forte liaison entre ces deux classes d'outils et ceci en intégrant les modèles (par les profiles) spécifiques aux plates-formes (ou PSM Platform Specific Model) au niveau des outils CASE (figure 4.1).

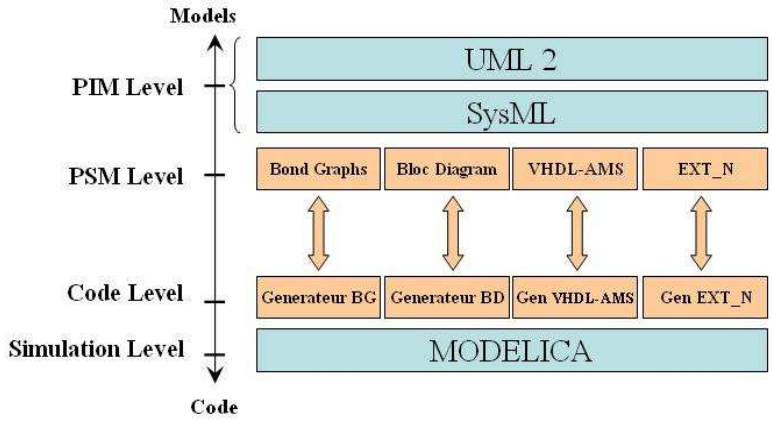


Figure 4.1. Environnement de conception orienté objet de systèmes dits mécatroniques basé sur SysML et ses extensions.

4.2 Construction de SysML

SysML **[SysML2007]** est un langage défini comme une extension à UML2 **[UML2007]** La partie d'UML2 qui est réutilisée par SysML est appelée UML4SysML (Figure 4.2). Les diagrammes UML2 ont été étendus pour subvenir à des besoins spécifiques à l'ingénierie système. En évaluant SysML par rapport aux critères énumérés par Friedenthal et Bukkart **[Friedenthal2003]** on peut dire ce qui suit : par héritage d'UML, SysML bénéficie de mécanismes d'extension qui lui permettront d'être adaptatif par rapport au domaine et évolutif. Il hérite aussi du format d'échange XMI pour l'échange entre outils. Il est aussi facile d'utilisation d'autant plus qu'un large public et experts sont déjà familiarisés avec UML. Les autres points restent à être vérifiés au fur et à mesure que SysML est construit mais ce sont les mêmes problèmes qui ont été posés pour la construction d'UML2, ce qui veut dire que des efforts peuvent être épargnés pour SysML en profitant de l'héritage d'un langage qui soit non-ambigu, précis, adaptatif par rapport à la taille du système étudié et indépendant du processus. La complétude devra être vérifiée par les extensions apportées à UML2 et les besoins des ingénieurs système.

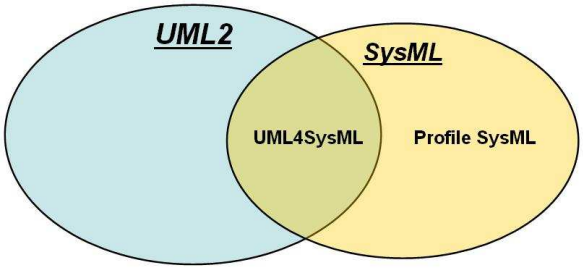


Figure 4.2. Relations entre UML2 et SysML **[SysML2007]**

Les besoins suivants ont été exprimés par les ingénieurs système représentés par l'INCOSE **[INCOSE]** au cours d'une étude préalable à la spécification de SysML. Ces besoins l'ont motivée à développer sa propre méthode d'ingénierie système OOSEM (Object Oriented Systems Engineering Method) **[OOSEM2001]** :

- Incompatibilités et problèmes de communication entre les ingénieurs système et logiciel.
- Besoin d'un meilleur moyen pour représenter les informations relatives à l'analyse et la conception des systèmes et d'un meilleur moyen pour la communication des besoins.

De plus, l'INCOSE a exprimé les besoins ressentis par ses membres quant à la représentation des informations, et a présenté en parallèle les solutions qu'ils ont apportées à travers la méthode OOSEM :

- Besoin de représenter les éléments non-logiciels :
  - ❖ Création de stéréotypes pour désigner un composant hardware générique.
  - ❖ Création de stéréotypes pour différencier un utilisateur et un système externe (acteurs de stéréotypes différents).
  - ❖ Différencier un système et un sous-système.
- Besoin de représenter des attributs de performance, des attributs physiques et non comportementaux :
  - ❖ Attributs stéréotypés : Required performance et Actual performance.

- ❖ Attributs stéréotypés : Caractéristiques physiques, non comportementales.
- ❖ Distributions probabilistes de ces attributs.

- Attributs de données pour des éléments physiques :
  - ❖ Attributs de données pour représenter des stocks de système, extension du concept de stock persistant pour supporter des entités physiques (énergie, fluides…).
- Eléments de modélisation explicites pour représenter des entrées/sorties :
  - ❖ Les entrées sorties peuvent inclure des données ou des échanges physiques.
  - ❖ Un diagramme spécifique est utilisé (Elaborated Context Diagram) pour représenter les flux entre les acteurs et le système ou encore entre composants du système.
- Liens entre couloirs (swimlanes) et classes, et entre activités et opérations :
  - ❖ Les scenarii sont utilisés pour dériver des besoins fonctionnels de manière à faire correspondre les couloirs des diagrammes d'activités à des classes et les activités à des opérations.
- Représentation du temps continu :
  - ❖ Utilisation d'un diagramme temporel similaire à un diagramme de GANTT.
- Sémantiques limitées pour spécifier des événements et coupler l'événement avec l'action :
  - ❖ Définition des événements de manière explicite pour supporter des entrées/sorties physiques par exemple définition d'un seuil de franchissement et couplage avec l'action résultante.

4.3 Les apports de SysML

La figure 4.3 présente l'ensemble des diagrammes adoptés par SysML :

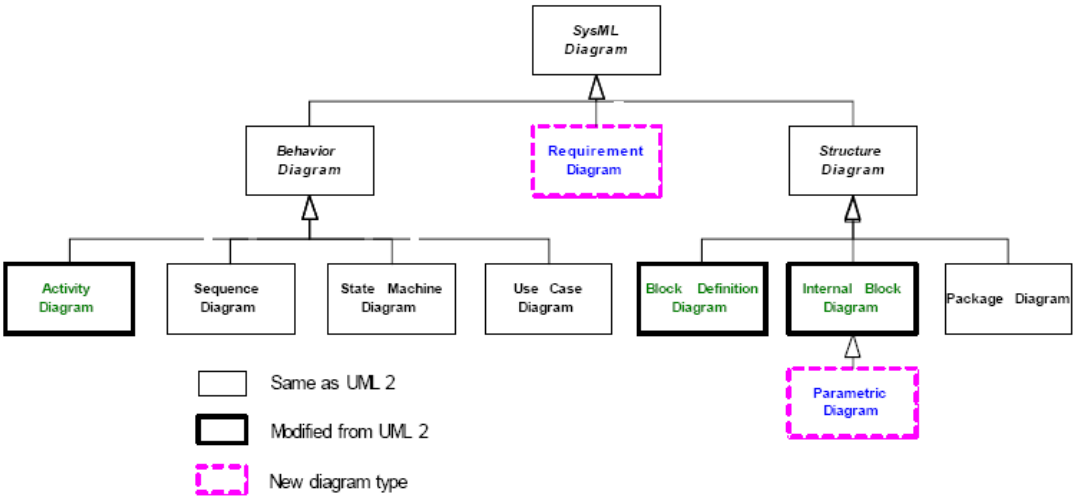


Figure 4.3. Les diagrammes de SysML **[SysML2007]**

En SysML, seul le diagramme de séquences est retenu parmi les diagrammes d'interactions ; les diagrammes de timing, de collaborations et de vue globale d'interactions (Interactions overview) qui sont définis dans UML2 sont exclus.

Le diagramme des exigences est la principale extension apportée par SysML (figure 4.4), il est défini en tant qu'extension du diagramme de classes. Il permet d'analyser les exigences en gardant une trace des réflexions des ingénieurs. Il permet de structurer les besoins fonctionnels et non fonctionnels. Il définit de nouveaux stéréotypes décrits dans le tableau 4.1.

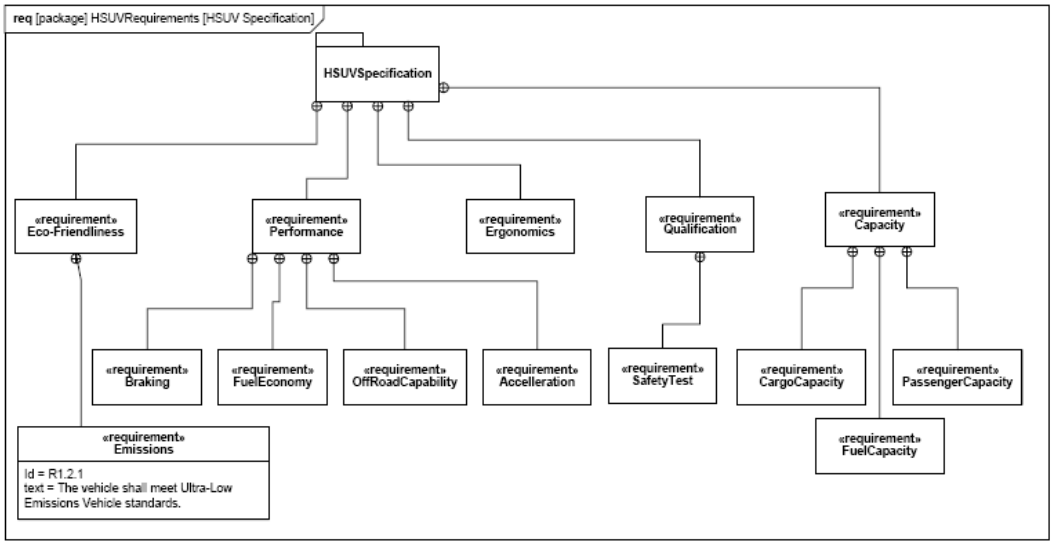


Figure 4.4. Diagramme des exigences **[SysML2007]**.

Tableau 4.1. Stéréotypes utilisables dans un diagramme des exigences.

<<requirement>>	Est une extension abstraite de "Classe" affichant deux méta-propriétés spécifiques. La méta-propriété "id" est destinée à la numérotation hiérarchisée des besoins. "text" permet de décrire le texte du besoin.
<<deriveReqt>>	permet d'expliciter un besoin qui était implicite dans un besoin de niveau hiérarchique supérieur de la décomposition
<<Verify>>	permet de lier des jeux de test à un besoin
<<copy>>	permet la réutilisation d'un besoin dans un autre diagramme de besoins
<<satisfy>>	est une sous-classe du stéréotype <<realization>> qui permet de dire qu'un ensemble d'éléments implémentent un ensemble de besoins

Une deuxième extension importante est le Block qui est défini en tant que stéréotype de la classe définie dans UML2 dans le package StructuredClasses **[UML2007]** (figure 4.5). Cette classe étend la classe UML2 en y ajoutant la possibilité d'avoir une structure interne et des ports. Un Block est une unité modulaire. Il peut contenir des éléments structurels et comportementaux.

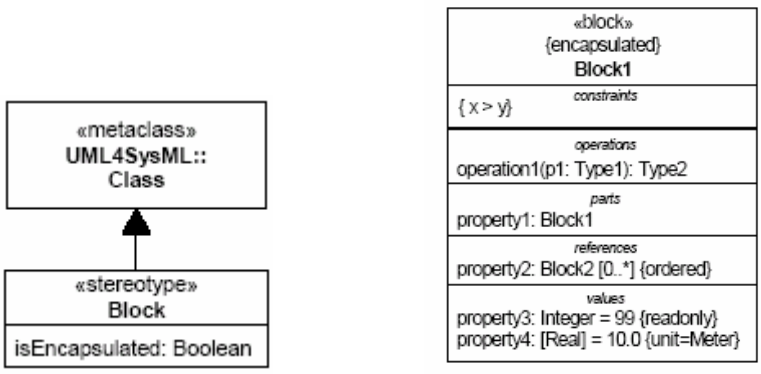


Figure 4.5. Le stéréotype de classe : <<Block>> **[SysML2007]**

Le diagramme de définition de blocks est basé sur le diagramme des classes d'UML2. Il permet de décrire des relations de composition, des agrégations, des dépendances, des généralisations de blocks.

Le diagramme des blocks internes est aussi une extension apportée par SysML. Il est basé sur le diagramme des structures composites d'UML2 **[UML2007]** (figure 4.6). Il décrit des structures internes de blocks par des ***interconnexions*** de blocks ou des parties de blocks (<<parts>>) à travers des ports. Des connexions peuvent être reliées directement à des propriétés internes d'un block sans passer par un port. Le diagramme IBD permet de décomposer le système, de représenter les composants (physiques ou logiques) et les interfaces qui les lient (figure 4.6). Il est utilisé pour représenter les flux d'information et les flux continus ou discrets entre les composants.

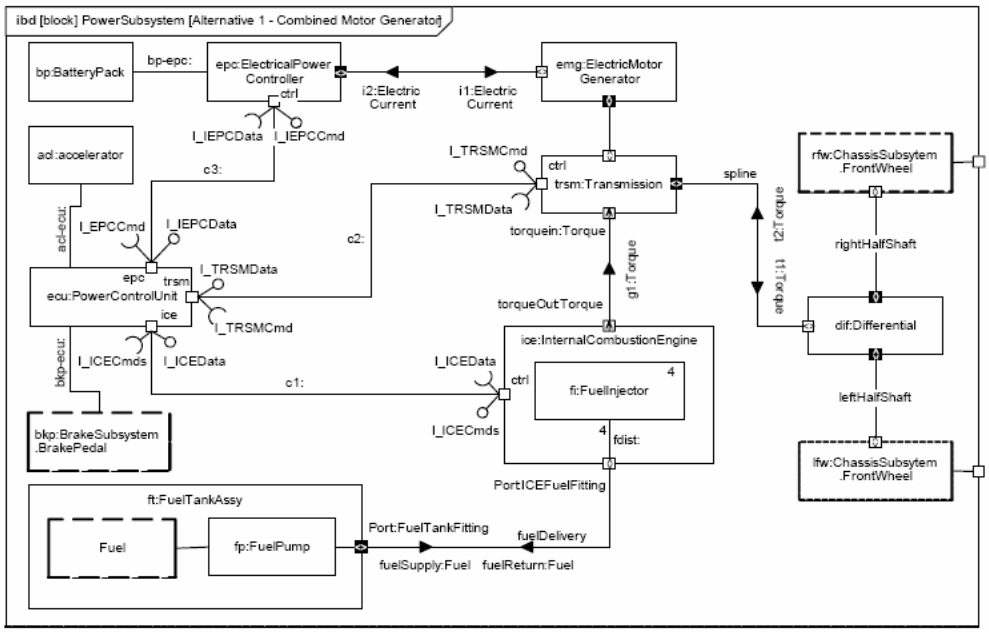


Figure 4.6. Diagramme IBD **[SysML2007]**.

Le diagramme des contraintes paramétriques est un nouveau type de diagrammes qui est une extension au diagramme de blocks internes. Ainsi dans un diagramme paramétrique un block sera stéréotypé <<ConstraintBlock>> et une propriété de block sera stéréotypée <<ConstraintProperty>> (figure 4.7). Il permet de spécifier des grandeurs physiques. Il est utilisé pour décrire :

- Des caractéristiques quantitatives d'un système.
- Des expressions mathématiques (F=m*a..) reliant des propriétés physiques du système (figure 4.15).
- Des performances critiques.
- Des contraintes temporelles.

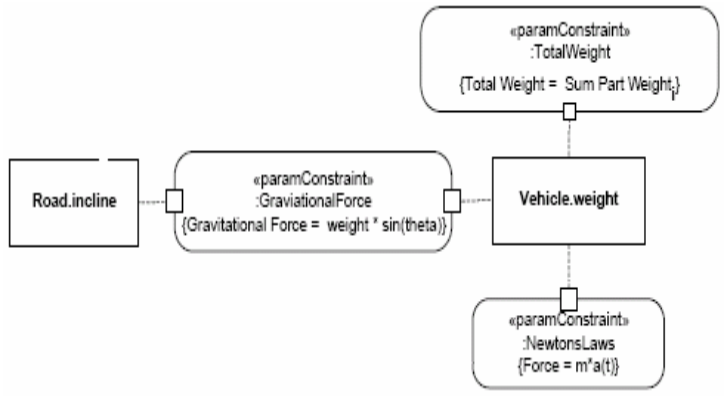


Figure 4.7. Diagramme des contraintes paramétriques **[SysML2007]**.

Quant au diagramme des activités, dans SysML le support des modèles markoviens est introduit en permettant l'affectation d'une distribution de probabilités des transitions entre actions. Dans UML2 on peut dire que le paramètre d'une activité peut se répéter pendant l'exécution de l'activité (propriété {isStream}), en SysML on peut définir en plus de cela la fréquence de cette répétition ({Rate}) et si ce flux est continu ou discret. Il permet de décrire :

- Représenter la dynamique des sous-systèmes similaires aux EFFBD[15].
- Entrées/sorties, séquencement, coordination.
- Flux physiques, continus, activités interruptibles.
- Permet l'expression des distributions probabilistes.

D'autres éléments inter-diagrammes sont introduits par SysML. La notation d'appel externe (*callout notation*) est un mécanisme fourni dans SysML pour décrire des relations entre éléments appartenant à différents diagrammes. On peut ainsi rappeler une machine d'états en vue par exemple de l'allouer à un block défini dans un diagramme de blocks internes IBD.

Le commentaire <<rationale>> peut être utilisé pour illustrer le raisonnement qui a conduit à un choix donné. Cette extension n'est réalisable que si des modifications sont apportées au méta-modèle UML2. En effet "comment" ne peut pas être stéréotypé en UML2. Le commentaire <<Problem>> peut être attaché à n'importe

[15] EFFBD est le Enhanced Functional Flow Block Diagram.

quel élément du modèle SysML pour décrire une limitation détectée par le concepteur. <<View>> et <<Viewpoint>> sont introduites pour décrire des perspectives particulières d'un système. Nous pouvons ainsi définir les points de vue que nous jugeons utiles tels que les points de vue de la qualité de service, de la performance, de l'ergonomie etc.

SysML propose aussi le mécanisme d'allocation. Contrairement au profile MARTE **[MARTE2007]** qui limite l'utilisation de l'allocation ; l'allocation d'un élément applicatif à un autre de la plateforme, SysML permet d'allouer n'importe quel sous-ensemble d'éléments à n'importe quel autre sous-ensemble d'éléments **[Gérard2007]**. Mickael Latta **[LattaHomepage],** qui est un des reviewers de SysML, critique vivement cette approche de SysML en disant que les ingénieurs du logiciel affectent dés le début de la conception les comportements aux structures statiques, ils créent des classes dans lesquelles ils créent des propriétés et des méthodes, plus tard dans la conception ils font du « refactoring » pour modifier ces classes. Par contre, les ingénieurs système séparent les structures des comportements puis allouent ces derniers aux structures. SysML permet de décrire cette allocation de manière à documenter une décision de conception qui pourra être détaillée plus tard, Latta critique alors cette approche en ajoutant qu'au-delà de la documentation, les allocations devraient être en plus des éléments de conception sémantiquement actifs et que l'allocation d'un comportement à une structure doit être équivalente à l'implémentation d'une méthode pour une classe ; ce qui est le cas avec les allocations dans MARTE.

4.4 Critiques de SysML

SysML présente certaines lacunes que nous présentons dans ce qui suit :

- Chevauchement sémantique entre uses cases et diagrammes des exigences.

- Liens entre diagrammes : Les allocations ne font que documenter, pas de sémantique définie.
- Diagramme des contraintes paramétriques pas assez intuitif.
- Le chevauchement sémantique des différents diagrammes donne naissance à un problème : Un changement dans l'un des diagrammes doit pouvoir se répercuter dans un autre. Il y a donc besoin dans SysML de limiter les chevauchements au minimum et à l'outil d'implémentation de supporter la mise à jour d'une vue lorsqu'un modèle est modifié.

Mickael Latta **[LattaHomepage]** critique aussi vivement les « nested connectors » qui permettent de relier un élément à un élément contenu dans un autre sans passer par une frontière. Il déclare dans sa relecture de la spécification de SysML qu'en ingénierie système, il est fortement recommandé de décrire rigoureusement les frontières des composants. Il ajoute que même si dans leurs apparences les systèmes physiques peuvent avoir des « fils » qui rentrent directement dans leurs voûtes, il ne serait pas judicieux de modéliser cela par des « nested connectors » car en ingénierie système, ce ne sont pas les connections physiques qui comptent mais plutôt les connexions logiques. En réalité la frontière du composant n'est pas définie par sa structure physique externe mais plutôt par son interface. Ainsi, il déclare que respecter le plus rigoureusement possible l'encapsulation permet de mieux modéliser et mieux appréhender les composants tout en améliorant la réutilisation. Mickael Latta reproche aussi à UML2 cette innovation nuisible qu'est la « conformité élémentaire » qui permet à un éditeur d'outils CASE d'être en conformité avec un ou plusieurs packages de la spécification sans pour autant être en conformité par rapport aux autres packages, ce qui permet à ces éditeurs d'outils de prétendre être conformes à UML2 sans que cela ne soit rigoureusement vrai. On obtient alors des dizaines d'outils implémentant des dizaines de langages différents, ne permettant pas l'utilisation commune des modèles (Model Interchange). Un projet est donc bloqué avec un outil de départ et l'apprentissage du langage dépendra de l'outil choisi. Ce qui brise le caractère « standard » d'UML. Il fait remarquer ensuite que SysML continue avec cette même approche de conformité élémentaire, il hérite donc des mêmes problèmes.

Un autre point doit être noté concernant la portée de SysML. En effet, l'utilisation de SysML (le langage tel que décrit dans le standard **[SysML2007]**) s'arrête à la conception indépendante des plateformes et des technologies. Ensuite, nous avons besoin pour la conception spécifique plus détaillée de mettre en place d'autres profiles UML ou des extensions de SysML (MARTE, MecaML, ElecML) comme le montre le processus décrit par la figure 4.8 :

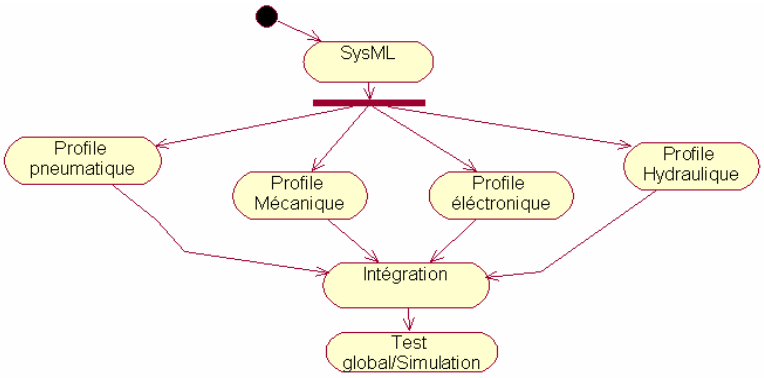


Figure 4.8. Portée de SysML et son extension.

Une autre problématique se pose par rapport aux niveaux d'abstraction du BDD et de l'IBD autrement dit du block et des propriétés du block. Dans la figure 4.9 on voit que le block, qui est un stéréotype de classe n'a pas de correspondant dans le langage SysML au niveau des instances. Tim Weilkiens dans **[Weilkiens2008]** explique la nuance entre le niveau des instances (M0) et le niveau d'abstraction dans les IBD qui est le niveau des rôles et qu'il décrit comme un niveau entre le niveau M1(classes) et M0(objets).

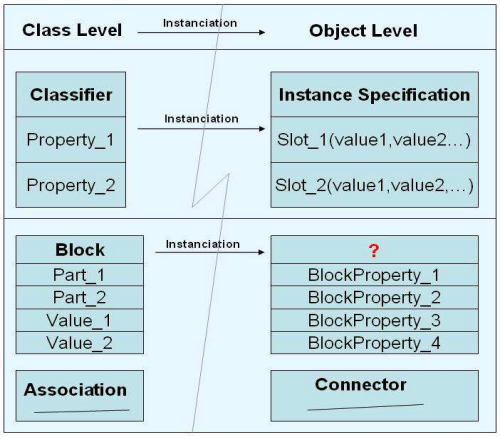


Figure 4.9. Correspondances des éléments SysML
du niveau classes avec ceux du niveau objets.

Le block est utilisé aussi pour désigner un classifieur non une instance. On peut ainsi se demander à quoi correspondrait une classe pour un mécanicien. A-t-il vraiment besoin de passer par ces niveaux d'abstraction ? Ou bien est-il superflu pour lui de définir des classifieurs englobant toute une série d'instances ? Nous pensons que les deux niveaux sont indispensables mais il est très utile de permettre aux non informaticiens de ne pas confondre les niveaux d'abstraction. En effet, la conception orientée objet a pour avantages de permettre un haut degré de réutilisation. On peut donc faire correspondre les classes aux éléments d'une bibliothèque commune à l'ensemble des projets d'une même entreprise. Lorsqu'un concepteur définit une classe, il peut la publier dans la bibliothèque de classes. Il peut aussi l'utiliser dans un diagramme de définition de blocks BDD pour concevoir d'autres classes.

L'un des grands avantages de SysML est qu'il soit basé sur UML, il peut ainsi profiter de tous les outils UML existants sur le marché et les fournisseurs d'outils CASE peuvent donc créer un profil SysML pour supporter ce nouveau langage. Sauf que dans la situation actuelle, SysML n'est pas exactement un profile UML. Chaque fournisseur implémentera donc une version qui sera intégrable dans son outil selon les mécanismes d'extension dont son outil dispose. L'unification de SysML passe donc forcément par le simple fait qu'il doit être un profile d'UML2 et non pas une union d'un profile UML2 avec un nouvel ensemble d'éléments.

Conclusion

Nous avons choisi d'utiliser SysML comme langage de base des modèles. Nous avons présenté le langage UML et le langage SysML, nous avons mis l'accent sur les mécanismes d'extension qui nous permettront de compléter les modèles utilisés par la méthodologie proposée. Nous avons dû présenter avec un certain niveau de détail ces technologies car nos apports reposent sur ces technologies, il était alors indispensable de présenter aux lecteurs l'ensemble de ces technologies.

Notre méthodologie sera construite à partir de ces technologies. Ainsi, après avoir présenté les modèles que nous avons choisi de mettre en œuvre, nous allons

maintenant présenter comment nous articulerons ces modèles (SysML et autres : Bond Graphs, DSM, etc.). Ces modèles seront articulés de manière à supporter le processus d'ingénierie système IEEE15288. Pour ce faire, nous présentons dans le chapitre suivant l'ingénierie guidée par les modèles et, en particulier, l'architecture de modèles MDA qui va permettre cette articulation.

Chapitre 5

V - L’ingénierie guidée par les modèles ou MDE

« Using models to design complex systems
is de rigeur in traditional engineering disciplines. »
Bran Selic, IBM Rational
Software

5.1 Introduction

Le MDE « model-driven engineering » (ingénierie guidée par les modèles) consiste en l'utilisation des modèles en tant que produit réel. C'est à dire que ces modèles ne sont plus seulement des moyens de communication entre les personnes impliquées dans le projet. Dans ce cas là, le code développé représente le produit réel (en génie logiciel). Deux approches opposées ont proposé des solutions au développement guidé par les modèles ou MDD. La première étant MDA **[MDA2003]** (model-driven Architecture supportée par l'OMG[16] , et la deuxième celle des Software Factories encouragée et supportée par Microsoft **[DSL]**. Une discussion comparant ces deux approches a été réalisée au paragraphe 3.3.7 (*étendre UML* Vs *créer un DSL*).

5.2 Introduction à MDA

L'approche MDA introduit une approche globale pour le développement des systèmes logiciels. MDA est basée sur l'idée que toute entreprise doit pouvoir bénéficier de ses expériences en accumulant son savoir faire dans un ensemble de modèles. Le savoir faire visé s'étend sur toutes les étapes de la construction d'un système. Cela découle de la remarque qu'une entreprise évolue généralement dans un domaine d'application spécifique en mettant en œuvre des technologies qui évoluent avec une certaine fréquence plus grande que pour le changement des techniques métiers. MDA propose donc d'accumuler ce savoir-faire dans des modèles qui représentent des schémas d'action face à une situation donnée. Par exemple, dans une situation où nous voulons fournir des données météorologiques à partir d'un centre de surveillance vers des unités mobiles, l'expérience montre que la meilleure solution est décrite par le « Remote Method Call Pattern » **[Douglass2002]** page 363-364 dans le modèle suivant (figure 5.1) :

[16] organisme de standardisation des technologies objet, ayant standardisé CORBA, UML, MDA etc.

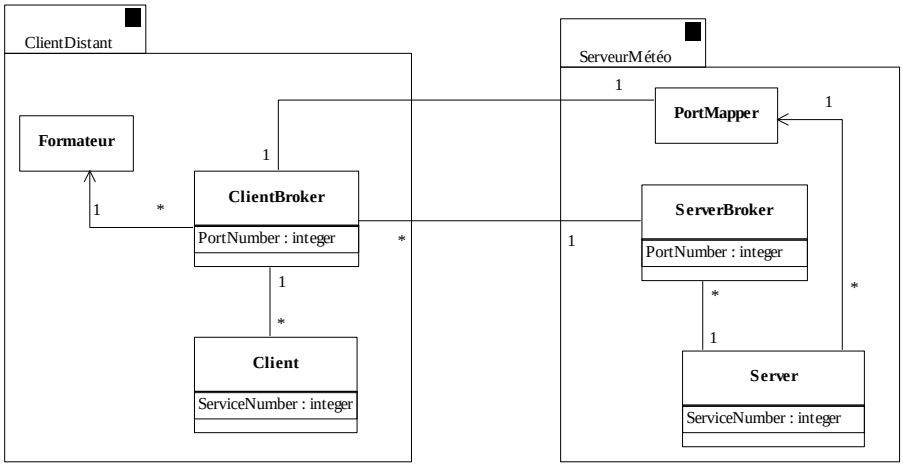


Figure 5.1. Pattern d'appel de méthodes distantes de systèmes pouvant s'élargir.

MDA étend ce principe utilisé pour les Design Patterns pour mettre en place des modèles de haut niveau (ou méta modèles) de systèmes logiciels entiers. Ces méta modèles contiennent donc des éléments de description de systèmes et peuvent contenir plusieurs Design Patterns. Jean Bézivin **[Bézivin2004]** décrit l'évolution vers le MDD en décrivant les deux relations de base (figure 5.2) qui ont régi la modélisation orientée objet et les deux relations qui régissent la modélisation guidée par les modèles :

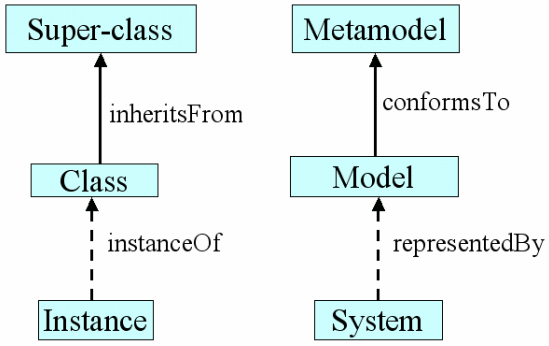


Figure 5.2. Relations fondamentales régissant la modélisation orientée objet et la modélisation guidée par les modèles.

Les modèles dans MDA **[MDA2003]** peuvent être des méta modèles de description ou des méta modèles de transformation. Les méta modèles de description sont le CIM (Computation Independent Model), le PIM (Platform Independant Model) et le PSM(Platform Specific Model) :

- Le CIM : Expression des besoins par un modèle indépendant de la conception.
- Le PIM : En MDA, un PIM (Platform Independent Model) est un modèle d'un haut niveau d'abstraction qui ne prend pas en compte les aspects relatifs aux technologies d'implémentation.
- Le PSM : Le PSM (Platform Specific Model) est un modèle qui décrit la solution technique à réaliser. Il est utilisé pour traduire le PIM en explicitant les technologies mises en œuvre.

Les modèles de transformation: Le passage d'un modèle à l'autre nécessite des procédures de transformations de modèles. Dans MDA, ces procédures sont exprimées par des modèles pour les mêmes raisons qui nous ont amené à utiliser des modèles au lieu du code. En effet on peut considérer une transformation d'un modèle en un autre comme un projet à part entière.

La génération de code et de documentation: Cette génération diffère de la transformation de modèles dans le sens où un document ou du code n'ont pas nécessairement de méta modèle. Ce sont des chaînes de caractères structurées.

L'approche MDA définit alors une chaîne de transformations que l'on veut essentiellement automatisée entre le CIM vers le code et la documentation (figure 5.3). Cette décomposition nous permet de décrire un système en termes de ses fonctionnalités, ce qui encourage la créativité et donne plus de liberté à l'ingénierie. Ceci permet de réutiliser des solutions abstraites (niveau PIM) en changeant le niveau de l'implémentation si la technologie venait à évoluer (par exemple, si une contrainte liée à une source d'énergie venait à changer). On n'aurait alors qu'à modifier la partie relative à cette technologie au niveau PSM et les règles de génération de texte qui suivent.

Toute la difficulté dans la mise en œuvre de MDA et le choix d'un PIM qui puisse non seulement nous amener vers le(s) PSM que nous envisageons d'utiliser mais aussi qui puisse être assez abstrait pour pouvoir être réutilisable et donc pérenne.

Ce sont les experts implémentant MDA qui fournissent le plus grand effort, les développeurs/ingénieurs viennent travailler sur cette plateforme MDA en se concentrant sur la créativité et les aspects conceptuels.

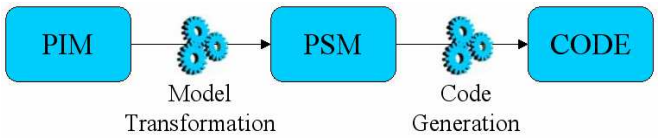


Figure 5.3. MDA process.

Pour décrire nos modèles (CIM/PIM/PSM/…) nous devons disposer de langages standards tels que le langage UML2 qui est préconisé par l'OMG à cette fin ci. Pour les modèles de transformation d'autres langages sont préconisés **[Kleppe2003]** page 32. UML est l'un des langages les plus largement acceptés dans la communauté du génie logiciel et aussi dans d'autres communautés comme celle des systèmes. Il est utilisé pour spécifier des modèles et le nombre d'outils implémentant UML (de manière plus ou moins fidèle) a augmenté de manière fulgurante (plus d'une centaine d'outils). Dans le langage UML, ces modèles (CIM, PIM et PSM) sont décrits par des profiles UML. Dans **[Fernandez2004]** les auteurs présentent des guidelines pour concevoir des profiles UML ainsi que le rôle des profiles dans la mise en œuvre de MDA. Ces profiles sont utilisés pour guider la conception. Ces profils sélectionnent les éléments de langage et les diagrammes utiles pour notre domaine d'application.

En plus d'un outil UML, il y a besoin d'outils de transformation et de langages de transformation pour pouvoir implémenter des transformations de modèles et des générations de texte. Le meilleur outil devra permettre un accès aisé aux modèles ; une liberté maximale de manipulation de ces modèles permet d'avoir un maximum d'automatisation dans le processus global. Ce qui permettrait de mieux maîtriser le processus de développement et d'améliorer la qualité, les coûts et aident à respecter les délais.

Utiliser UML pour implémenter MDA dans un contexte d'ingénierie système n'est pas trivial. En effet, UML a été construit pour le génie logiciel. Les vues/diagrammes qu'il offre sont donc celles utiles à la spécification de logiciels. Pour l'ingénierie système nous utiliserons SysML.

Finalement nous résumons ici les avantages d'une approche de conception basée sur les modèles. Tout d'abord, cela permet d'offrir un meilleur support de communication inter ou intra équipes. Ce qui diminuera les ambiguïtés et donc facilitera l'intégration des sous-systèmes. Ensuite, elle permet d'accumuler le savoir-faire et de le pérenniser, donc de le réutiliser. Grâce à la facilité de manipulation des modèles cela risque d'être la nouvelle tendance pour la conception de systèmes dans les années à venir.

MDA ne précise pas quel formalisme on doit utiliser pour exprimer des méta modèles. Par contre l'OMG préconise l'utilisation du MOF (Meta Object Facility) **[MOF2006]**. Le MOF est donc un formalisme de description de méta modèles. Il est utilisé entre autres à décrire le méta modèle d'UML. Un méta modèle en réalité ne permet de décrire que la forme des langages et non pas leur sémantique, ce qui constitue une faiblesse majeure. Il en résulte qu'un méta modèle, tel qu'il est défini par l'OMG, ne permet finalement pas de décrire des langages, car un langage est constitué d'une syntaxe et d'une sémantique. C'est de cette approche-ci que découlent toutes les faiblesses d'UML tel que sa non formalité. Ce que l'OMG appelle « precise natural language » dans **[UMLinfra2004]** reste toujours un langage naturel pouvant être interprété différemment par des personnes différentes et dans des contextes différents :

"The specification uses a combination of languages - a subset of UML, an object constraint language, and a precise natural language to describe the abstract syntax and semantics of the full UML." **[UMLInfra2004]** page 21.

Quant aux modèles de transformations de modèles (QVT[17] et ATL[18]), rappelons que dans MDA tout est modèle. Même les transformations des PIM en PSM sont décrites par des modèles. L'OMG propose le langage QVT pour décrire ces modèles de transformations, mais d'autres langages sont développés par d'autres groupes de recherche tels que ATL qui est développé par l'INRIA **[ATL2006]**.

5.3 L'application de Design Patterns en ingénierie système

Les diagrammes de classes en UML sont des modèles de systèmes, les Design Patterns ou patrons de conception sont des modèles de solutions à des problèmes récurrents. C'est une manière efficace de capitaliser le savoir-faire. Ils sont très utilisés en génie logiciel mais très peu en ingénierie système. Les design patterns sont aussi un outil efficace pouvant être intégré dans une ingénierie guidée par les modèles. L'application d'un pattern fournit un point de départ pour le concepteur, une solution globale à affiner et, si besoin est, à modifier. Les design patterns définis évoluent avec l'expérience. Le choix du nom d'un design pattern doit être fait de sorte à expliciter brièvement le problème résolu. Dans le domaine du génie logiciel, il existe beaucoup de design patterns dont les plus connus sont les « GoF[19] Patterns » **[Gamma1994]**.

5.4 Le choix de Modelica comme langage pour la simulation

Plusieurs travaux ont été réalisés en MDE (Model-Driven Engineering), en particulier pour générer du code à partir de modèles UML. Dans **[Andersson2007]** et **[Kreku2007]** les auteurs présentent un générateur de code SystemC à partir d'un modèle UML en décrivant la mise en correspondance entre les éléments des deux langages en utilisant un profile UML pour SystemC comme une étape intermédiaire dans le processus de génération de code. Dans **[Cano2007]**, la plateforme cible est OSGi, et l'auteur décrit une application de MDA pour implémenter des applications

[17] QVT (Query/View/Transformation) est un langage standardisé par l'OMG pour permettre la transformation de modèles.
[18] ATL (ATLAS Transformation Language) est un langage de transformation de modèles développé par le groupe ATLAS INRIA, http://www.eclipse.org/m2m/atl/.
[19] GoF pour The Gang Of Four ou le gang des quatre qui sont les inventeurs du concept de design patterns.

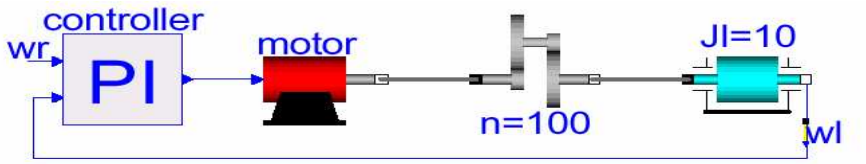
embarquées temps-réel sur OSGi. Le choix du langage cible dépend de l'utilisation que nous voulons en faire. Dans cette thèse, le choix s'est porté sur le langage Modelica **[Modelica]** en tant que langage pouvant être simulé par les outils spécifiques, ce choix a été fait pour plusieurs raisons :

- Langage orienté-objet donc mise en correspondance directe entre un modèle UML/SysML et une classe/modèle Modelica. Dans **[Dudra2003]** Dudra simule son système d'amortisseurs en utilisant Modelica. Ce dernier étant orienté-objet, il est plus naturel de passer d'un diagramme de classe qui détaille les différents composants du système vers un modèle en Modelica. Le processus de décomposition hiérarchique lui a permis d'aller vers le détail de fonctionnement du système.
- Langage orienté-objet donc extensible par la définition de librairies réutilisables. Beaucoup de librairies existent déjà, dont un grand nombre libre, quelques rares librairies sont payantes.
- Langage standardisé. De plus, de grandes compagnies s'investissent de plus en plus pour développer les outils Modelica. Dassault Systèmes a acheté l'éditeur Modelica appelé Dymola **[Dymola]** pour l'intégrer avec son outil CATIA.
- Une bibliothéque BondLib **[BondLib]** a été développée pour y intégrer la possibilité de décrire des Bond Graphs et les simuler.

Pour intégrer le support de la simulation à travers le langage MODELICA il faut utiliser un profile de MODELICA (profile existant ModelicaML **[Pop2007])** conjointement avec SysML en décrivant un cycle de conception adapté. Cette solution est applicable dans tout outil UML CASE supportant l'édition de profiles UML et la création de générateurs de code personnalisés. L'utilisation de l'éditeur dédié de modèles MODELICA tels que Dymola ou OpenModelica **[OpenModelica]** aurait pour avantage de modéliser directement avec les objets des bibliothèques. Mais l'intégration avec tout le cycle de conception se trouve ainsi rompue et on se trouve face à un problème de vérification de conformité de deux modèles disjoints. L'utilisation des profiles n'empêche pas de poursuivre le raffinement du modèle

généré en MODELICA par un éditeur externe et dans ce cas se pose aussi la question du retour d'informations vers le modèle principal qui doit rester unique et complet.

MODELICA offre plusieurs bibliothèques dans les différents domaines physiques telle que la mécanique, l'électricité, la pneumatique etc. Ces composants peuvent être assemblés et connectés comme peut le faire manuellement un concepteur qui fabrique son prototype (figure 5.5). MODELICA se prête très bien à une méthode de conception par décomposition.




```
model MotorDrive
  PI controller;
  Motor motor;
  Gearbox gearbox(n=100);
  Shaft Jl(J=10);
  Tachometer wl;
equation
  connect(controller.out, motor.inp);
  connect(motor.flange , gearbox.a);
  connect(gearbox.b , Jl.a);
  connect(Jl.b , wl.a);
  connect(wl.w , controller.inp);
end MotorDrive;
```

Figure 5.5. Exemple de code MODELICA.

Pour définir un composant dans MODELICA, nous entreprenons une approche objet (figure 5.5). Dans l'exemple qui suit, nous montrons comment est décrite une résistance R : La résistance est un élément électrique qui possède deux Pins comme plusieurs autres éléments électriques, on définit alors une interface OnePort :

```
partial model OnePort "Superclass d'éléments avec deux pins"
Pin p, n;
Voltage v;
Current i;
equation
v = p.v - n.v;
0 = p.i + n.i;
i = p.i;
end OnePort;
```

Le mot clé « Partial » équivaut au mot clé « Abstract » en C++. Les équations traduisent le comportement interne du composant. Par exemple que la somme de l'intensité entrante et de l'intensité sortante est nulle. Ensuite on définit l'élément R :

```
model Resistor "Résistance électrique idéale"
extends OnePort;
```

```
parameter Real R(unit="Ohm") "Résistance";
equation
R*i = v;
end Resistor;
```

Le mot clé « Parameter » implique que R est constant pendant une simulation mais peut changer d'une simulation à une autre. « Real » est l'un des types prédéfinis dans la librairie standard. Il possède un ensemble d'attributs tel que l'unité de mesure, valeur minimale, maximale ou initiale. Nous pouvons alors réutiliser la classe OnePort pour décrire le modèle d'une capacité :

```
model Capacitor "Capacité idéale"
extends OnePort;
parameter Real C(unit="F") "Capacité";
equation
C*der(v) = i;
end Capacitor;
```

L'avantage majeur de Modelica est, donc, le mapping un-à-un que l'on peut établir entre ses éléments de langage et d'autres langages (figure 5.6):

Modelica Code	IBD DG	Bond GR	Activity DG
Class model	Blocx	BG node	Action node
Connector	UML:port	Energy port	Pin class
Connection	UML:connector	Bond	Activity edge
Variable (connector)	UML:Port.Portname, UML:Port.Typename	(effort,flow)	Pin property (OutputPin)
Variable (class model)	Blocx property	Node property	Action property

Figure 5.6. Mapping entre
MODELICA / IBD SysML / BOND GRAPHS / ACTIVITY DIAGRAM

La bibliothèque BondLib **[BondLib]** permet de décrire des Bond Graphs sous MODELICA. La figure 5.7 en montre un exemple :

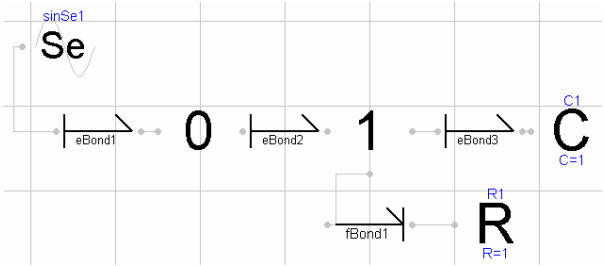


Figure 5.7. Bond graph d'un circuit RC.

Dans l'outil Dymola qui implémente le langage Modelica, le concepteur assemble graphiquement des éléments instanciés à partir de la BondLib et l'outil génère le code Modelica (figure 5.8) :

```
model test_BGs
  a;
  BondLib.Bonds.eBond eBond1 a;
  BondLib.Passive.R R1 a;
  BondLib.Junctions.J0p2 J0p2_1 a;
  BondLib.Junctions.J1p3 J1p3_1 a;
  BondLib.Passive.C C1 a;
  BondLib.Sources.sinSe sinSe1 a;
  BondLib.Bonds.eBond eBond2 a;
  BondLib.Bonds.eBond eBond3 a;
  BondLib.Bonds.fBond fBond1 a;
equation
  connect(eBond1.eBondCon1, J0p2_1.BondCon1) a;
  connect(sinSe1.BondCon1, eBond1.fBondCon1) a;
  connect(J0p2_1.BondCon2, eBond2.fBondCon1) a;
  connect(eBond2.eBondCon1, J1p3_1.BondCon1) a;
  connect(J1p3_1.BondCon2, eBond3.fBondCon1)
    a;
  connect(eBond3.eBondCon1, C1.BondCon1)
    a;
  connect(J1p3_1.BondCon3, fBond1.eBondCon1) a;
  connect(fBond1.fBondCon1, R1.BondCon1)
    a;
end test_BGs;
```

Figure 5.8. Code Modelica généré à partir du Bond Graph.

L'outil Dymola permet aussi la simulation du Bond Graph (figure 5.9):

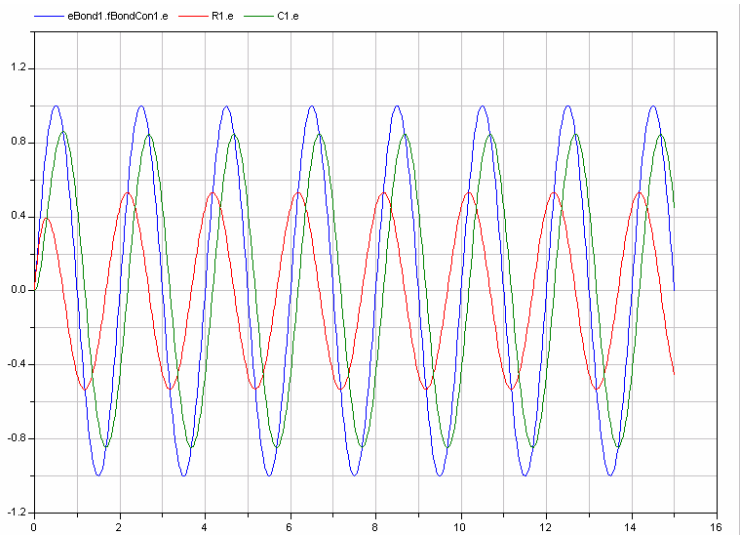


Figure 5.9. Résultat de la simulation du Bond Graph.

Conclusion

Nous avons dans cet état de l'art présenté les technologies qui composent la méthodologie proposée dans cette thèse (IEEE15288, MDA, SysML, mécanismes

d'extensions d'UML/SysML, Modelica, Bond Graphs et DSM). Vu le nombre de technologies concourantes à l'élaboration de la méthodologie, il était nécessaire de les présenter dans un niveau de détail permettant de montrer le cadre général de la réflexion menée dans ce travail.

Nous allons, à présent, présenter la méthodologie proposée. Nous commencerons par présenter l'articulation globale des modèles selon l'architecture MDA. Ensuite, nous présenterons les extensions apportées aux modèles choisis. Nous présenterons d'abord les extensions liées à l'adoption du standard IEEE 15288, puis les extensions spécifiques aux systèmes mécatroniques, en particulier l'intégration des DSM et du profile supportant la modélisation des Bond Graphs. Finalement, nous présenterons l'application du standard IEEE 15288 et la mise en correspondance des modèles aux processus de ce standard. Dans **[Fontan2008],** l'auteur présente sa méthodologie de conception utilisant SysML, destine aux systèmes temps-réel et permettant la vérification formelle des contraintes temps réel. Pour notre part, nous présentons une méthode de vérification de la qualité de l'architecture conçue moyennant des métriques et des transformations de modèle utilisant les DSM.

Dans **[Kadima2005]**, l'auteur adapte un cycle de vie en Y à l'architecture MDA (page 180). Il associe les modèles de la branche gauche du cycle en Y aux PIM de haut niveau d'abstraction (UML), il associe les modèles de la branche droite du cycle Y aux PIM décrivant les architectures d'implémentation à un haut niveau d'abstraction (indépendamment de la plateforme), puis il associe aux modèles de réalisation les PSM qui vont mapper les concepts de haut niveau aux plateformes. Nous avons privilégié d'utiliser un cycle itératif permettant le prototypage et la simulation qui sont indispensables en mécatronique, nous présenterons dans le chapitre suivant le processus adopté dans notre méthodologie et les modèles (PIM et PSM) associés.

Chapitre 6

VI - Approche et extensions adoptées pour la méthodologie de conception proposée : ***MISSyM***

"…What I am suggesting is that the underlying language definition must deal with the differences in some way explicitly. If the language does not differentiate between two types of computation … it by necessity results in a system that treats them the same."
Michael Latta
Sowtware engineer
UML2 contributor
SysML Reviewer

6.1 Introduction

Nous avons choisi d'intituler notre méthodologie proposée **MISSyM** pour ***M***éthodologie d'***I***ngénierie ***S***ystème basée sur ***Sy***sML, ***M***DA et le standard IEEE 15288.

Dans ce chapitre nous allons présenter l'ensemble de la solution MISSyM proposée. Nous rappelons d'abord les 3 types d'approches de conception d'après Clarkson **[Clarkson2005]** ; Les approches abstraites, les approches procédurales et les approches analytiques (voir paragraphe 2.3). Le standard IEEE15288 est une approche abstraite d'après la définition donnée par Clarkson**.** Nous avons choisi de le combiner à l'architecture MDA, qui définit de manière abstraite un ensemble de représentations, et nous avons choisi en tant que représentations le langage SysML pour mettre en place une approche analytique (toujours d'après la définition de Clarkson).

La méthodologie proposée (Figure 6.1) comporte un ensemble de modèles décrits par l'extension de SysML destinés à l'ingénierie des systèmes mécatroniques selon l'architecture MDA appliquée au standard IEEE15288 (processus techniques seulement). Elle comporte aussi un ensemble de patrons de conception permettant de fournir au concepteur des modèles solution utilisés comme base de son travail de conception. De plus, nous définissons une procédure de vérification de la qualité des modèles moyennant des métriques (indicateurs calculés sur ces modèles) et un algorithme d'amélioration du modèle qui met en œuvre la matrice de dépendances DSM (Design-Dependency Structure Matrix). Cette méthodologie comporte aussi des générateurs de code MODELICA permettant de simuler certains sous-ensembles du modèle global. Et finalement nous proposons, pour cette méthodologie, un cycle de conception itératif et systématique décliné du standard 15288 **[IEEE15288]** en associant les différentes vues du modèle aux différentes activités des différents processus de ce cycle de conception.

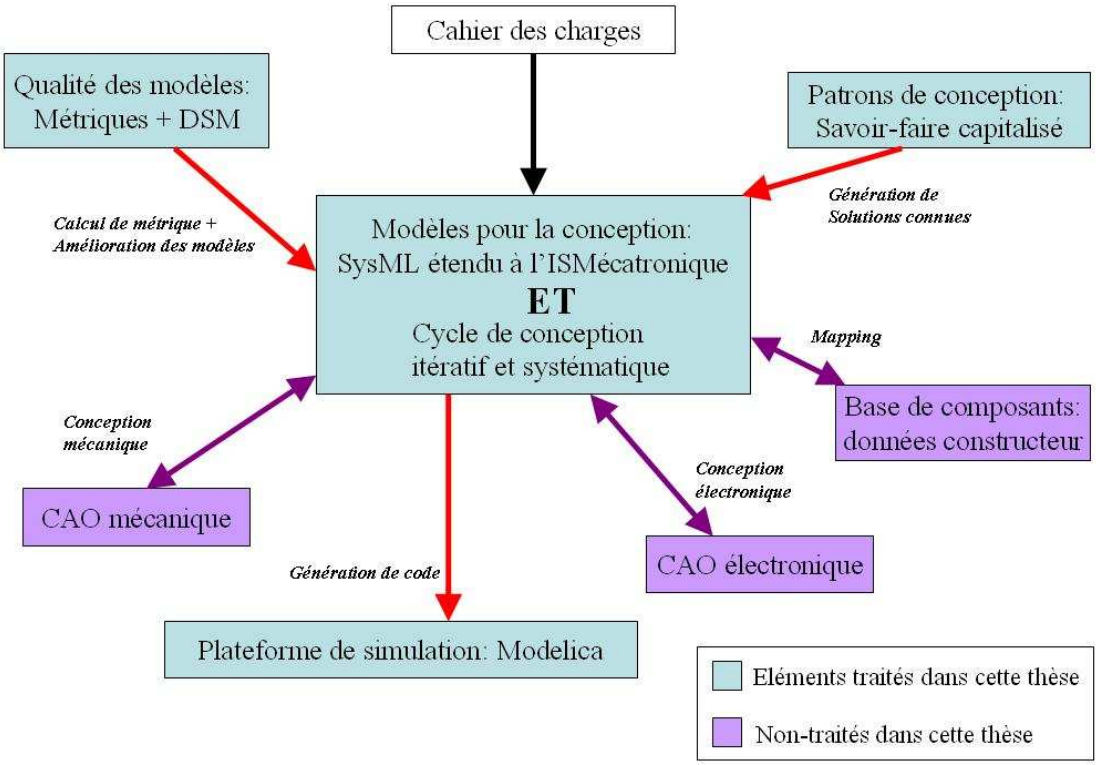


Figure 6.1. Eléments de la méthodologie globale proposée.

Nous commencerons par proposer le modèle global par application de l'architecture MDA. En particulier, nous présenterons les extensions élémentaires que nous apportons à SysML. Puis nous présenterons les éléments satellites que sont la qualité des modèles et les patrons de conception. Et finalement nous détaillerons le cycle de conception et la mise en correspondance des vues avec les activités du cycle.

6.2 Application de l'architecture MDA

6.2.1 Introduction

Dans **[Oliver2005]** l'auteur rappelle que malgré que MDA fournisse une « structure méthodologique », il reste à savoir comment les modèles s'insèrent dans cette structure. Nous nous proposons de répondre à cette question fondamentale par rapport à MDA. Nous présentons dans ce qui suit l'application de l'architecture MDA en définissant les modèles choisis de la méthodologie de conception. Notre plateforme d'aide à la conception des systèmes mécatroniques doit avoir les spécifications suivantes :

- Elle doit reposer sur un méta-modèle supportant SysML pour la partie Analyse et conception.
- Pour les tâches de réalisation, elle doit supporter les langages utiles pour chaque domaine technologique. En permettant l'intégration de nouveaux langages.
- Elle doit supporter des cycles de vie mettant en œuvre des démarches de décomposition et itératives.
- Elle doit permettre la capitalisation du savoir-faire et la réutilisation.
- Elle doit offrir des fonctionnalités de contrôle de la qualité des modèles produits.
- Elle doit permettre la simulation de la réalisation.

6.2.2 Les éléments du modèle de la méthodologie proposée

Le PIM (ou Platform Independant Model) que nous proposons est basé sur UML2. Ce qui a pour avantage de fournir des modèles Orientés-objet et standards, et de bénéficier de la grande panoplie d'outils disponibles sur le marché. Il est composé de SysML muni de deux profiles que nous appelons « SysML15288 » et « SysMLMecatronics ». Le premier est une extension de SysML permettant de supporter les besoins d'expressivité décrits dans le standard d'ingénierie système IEEE15288 **[IEEE15288]**. Et le deuxième est une extension supportant une description fonctionnelle liée aux systèmes mécatroniques. Ensuite, nous définissons un PSM que nous appelons « BlockBondGraphs » permettant de décrire des Bond Graphs dans un diagramme SysML (le Internal Block Diagram). Nous proposons aussi d'utiliser le profile « ModelicaML » réalisé par **[Pop2007]** et permettant de décrire des simulations pour générer le code Modelica correspondant. Et, finalement, nous avons choisi d'utiliser Dymola (implémentation de Modelica) comme plateforme de simulation du code généré à partir des PSM cités sachant que la bibliothéque BondLib permet la simulation des Bond Graphs sur les outils Modelica (figure 6.2).

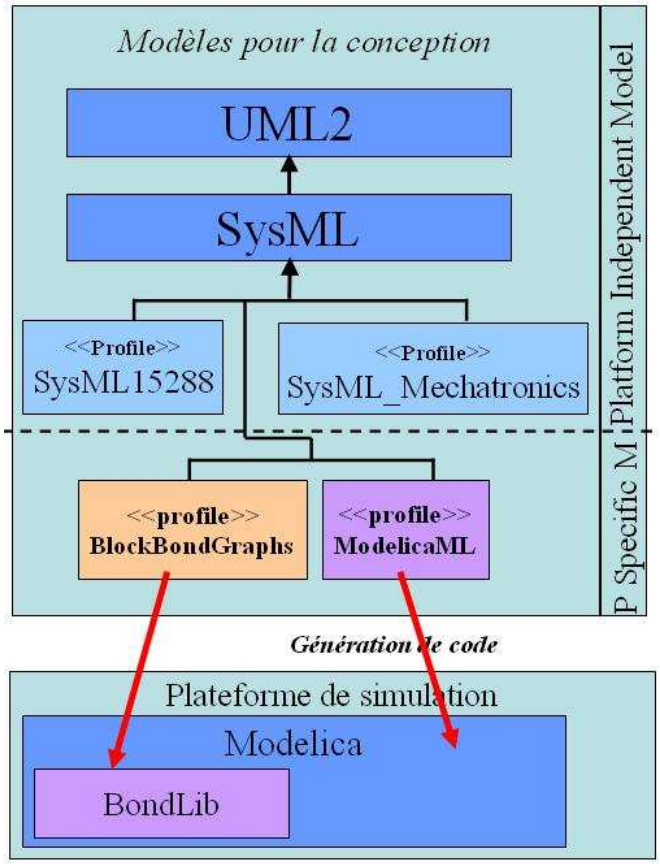


Figure 6.2. Application de l'architecture MDA pour une solution d'ingénierie mécatronique guidée par les modèles.

6.3 Extensions de SysML liées à l'adoption du standard 15288

6.3.1 Adoption du standard 15288

La mise en œuvre efficace du langage SysML nécessite l'utilisation d'un cycle de conception systématique. Nous proposons d'adapter le standard 15288 **[IEEE15288]** pour constituer ce cycle. Ainsi nous reprenons les étapes de définition du concept et de développement décrites dans le standard. L'application du standard 15288 nécessite que des extensions soient apportées à SysML pour supporter la modélisation des informations induites par les processus élémentaires du standard.

6.3.2 Les extensions induites par l'adoption du standard 15288

A – Extensions générales

Pour différencier entre les systèmes considérés, nous proposons {System-Of-Interest} une méta-propriété de type *String* de la méta-classe *package* permettant de spécifier que les éléments appartenant au package sont liés au cycle de conception du système considéré.

B – Extensions liées au processus de définition des besoins des parties prenantes

Au cours du processus de définition des besoins des parties prenantes réutilisé du standard IEEE 15288, nous avons besoin de décrire, pour un service donné, le contexte idéal de fonctionnement. Ceci est réalisable sous SysML en utilisant une note attachée au service (ou au use case). Pour alléger le diagramme des cas d'utilisations, nous proposons une méta-propriété de use case dédiée intitulée {RequiredContext} et de type *String*.

Nous avons besoin de décrire la qualité souhaitée d'un service donné, que ce soit une qualité de service d'ordre général, de sécurité ou d'ordre environnemental. Pour cela, et pour les mêmes raisons que l'extension précédente, nous proposons trois méta-propriétés de use case dédiées intitulées {QoS}, {SecurityQos} et {EnvironmentalQoS} qui sont de type *String*.

Nous proposons aussi la méta-propriété {Priority} d'un « requirement » de type PriorityLevel (que nous définissons par une énumération {1, 2, .., 10}) permettant de noter la priorité d'une exigence et permettra en cas de conflit entre un groupe d'exigences de retrouver les priorités. Nous intégrons ainsi « Ambiguity », stéréotype d'Association[20] UML permettant de noter des ambiguïtés entre « requirement »'s. Sachant que ceci est l'exemple typique de la relation n-aire, l'association n-aire stéréotypée « Ambiguity » peut regrouper ainsi un ensemble d'exigences.

Nous proposons par la suite {Stakeholder}, méta-propriété de type String d'un « requirement » qui permet de conserver la trace de la partie prenante concernée par

[20] Définition UML de l'association : Une association décrit un ensemble de tuples [UML superstructure] page 41.

cette exigence. Aussi {ConformToNeed}, méta-propriété booléenne de « requirement » permettant de faire valider les exigences par les parties prenantes.

c – Extensions liées au processus d'analyse des exigences

Pour le processus d'analyse des exigences, le diagramme des exigences (requirements diagram) en SysML nécessite plusieurs extensions pour supporter les activités du standard IEEE 15288. Quelques unes de ces extensions sont déjà décrites dans l'annexe C (non-normative extensions) de la spécification SysML **[SysML2007]**. Ces extensions représentent des propositions ne faisant pas partie de SysML mais pouvant être intégrées si besoin est. Le diagramme suivant (figure 6.3) présente ces extensions :

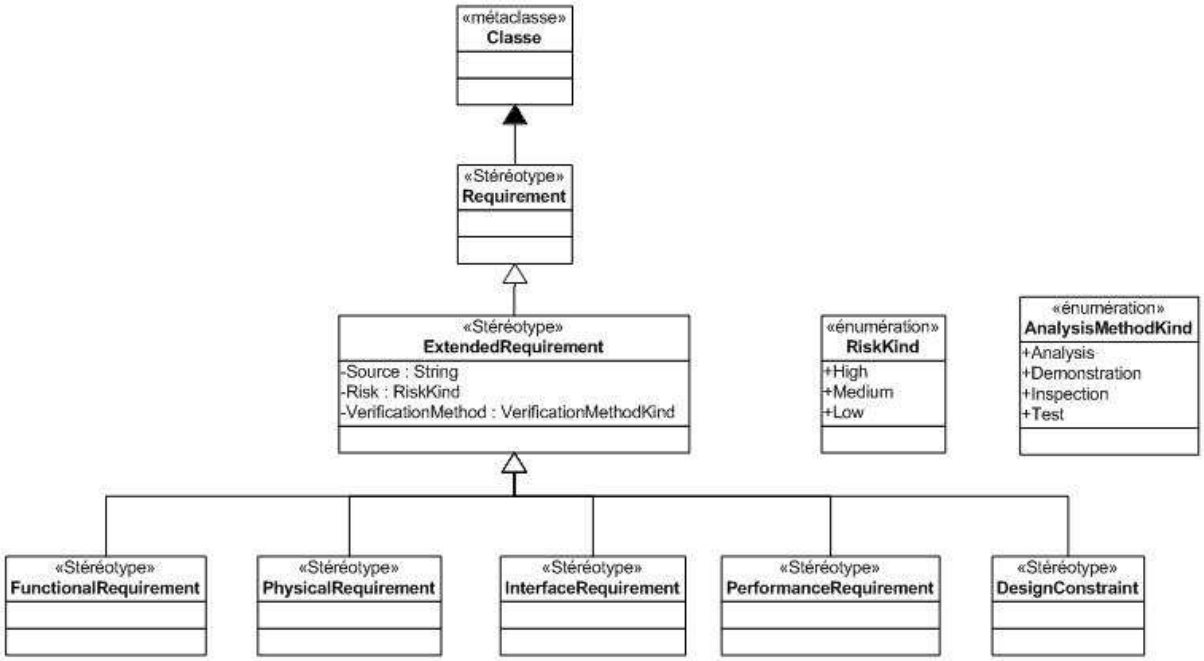


Figure 6.3. Extensions de la métaclasse « requirement »
dans l'annexe C non normatif de SysML.

Dans le standard IEEE **[IEEE15288]**, les exigences sur le système et les exigences sur les services sont traitées dans deux activités différentes et séquentielles, il y a donc nécessité de différencier entre ces deux niveaux d'exigences, d'où l'utilité de ces stéréotypes. Nous reprenons toutes les extensions proposées dans l'annexe C de SysML pour pouvoir différencier entre les exigences. Nous proposons aussi une extension permettant de décrire pour chaque exigence la méthode de vérification

prévue. Par contre, nous ne reprenons pas la méta-propriété {Risk} telle qu'elle est décrite dans la spécification de SysML, mais nous proposons une métapropriété {Risk} de type *RiskDataType* que nous définissons par *(RiskLevel : RiskKind, Risk :String)* permettant de noter les risques identifiés sur une exigence donnée et son niveau de gravité en reprenant l'énumération *RiskKind* de la spécification. Nous ne gardons pas la méta-propriété {source} définie dans l'annexe C, nous la remplaçons par {Stakeholder}, présentée avec les extensions du processus de définition des parties prenantes.

Pour les activités du standard IEEE 15288, nous complétons par la proposition de {RequiredContext}, méta-propriété de type string des stéréotypes « FunctionalRequirement » et « PhysicalRequirement » permettant de décrire les conditions nécessaires de fonctionnement. Nous proposons aussi, « SecurityRequirement » et «EnvironmentalRequirement » stéréotypes de « ExtendedRequirement » permettant de différencier les exigences concernant la qualité de service relative à la sécurité ou à l'environnement. Nous proposons aussi « ImplementationConstraint » qui est un stéréotype de « ExtendedRequirement » permettant de noter les contraintes de réalisation inévitables.

D – Extensions liées au processus de conception de l'architecture

Pour supporter les activités du processus de conception de l'architecture, nous proposons d'abord « AllocationRequirement », stéréotype de « requirement » permettant de dire qu'un « requirement » a été généré par l'allocation d'une fonction à un élément. Nous proposons ensuite l'énumération *ObtentionManners* {à acheter, à réutiliser, à construire} et {ObtentionManner}, la méta-propriété de « Block » qui, utilisés conjointement permettent de noter comment ce block sera obtenu.

Pour distinguer entre une architecture logique de haut niveau et une architecture physique, nous proposons « LogicalArchitecture » et « PhysicalArchitecture », tous

deux des *DiagramUsage*[21] (voir figure 6.4) du diagramme de description interne de blocs (IBD). Nous proposons aussi « LogicalArchitecture » et « PhysicalArchitecture », des *DiagramUsage* du diagramme de définition de blocs (BDD).

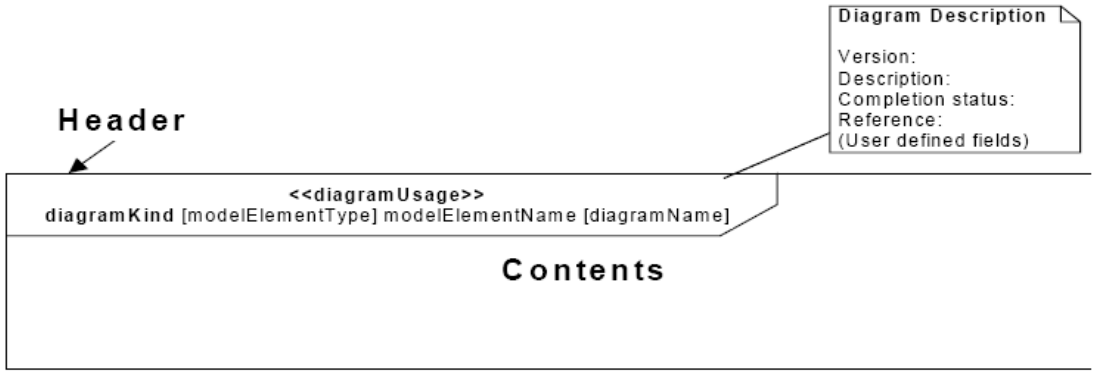


Figure 6.4. Description d'un diagramme SysML.

Nous introduisons aussi le stéréotype de block « Operator » (défini par héritage car block est déjà un stéréotype) qui permet de distinguer entre un opérateur humain et un sous-système ou un élément du système. Ce stéréotype est muni de la méta-propriété {MinHumanCapabilities} de type *string* permettant de noter les limites minimales d'un opérateur utilisant une interface du système.

Nous intégrons ensuite, une méta-propriété {Availability} du « block » SysML de type *AvailabilityType* que l'on définit par une énumération {Make, Buy, Reuse}. Ces deux extensions permettront de spécifier si l'on décide de construire un élément à partir de zéro, de l'acheter ou de réutiliser un élément existant (probablement en y apportant des modifications).

Pour permettre au concepteur de décrire plusieurs architectures alternatives pour un même block SysML, nous étendons à la note *DiagramDescription* (figure 6.4) du diagramme de description interne de blocs (IBD) et du diagramme de définition de blocs (BDD) la propriété *AlternativeIdentifier* de type *Integer*. De plus, à la note

[21] *DiagramUsage* est le terme utilisé pour étendre un diagramme, en effet l'utilisation du terme *stéréotype* est réservée à l'extension des méta-classes UML ou SysML, et *diagram* n'est est pas une, voir [SysML] page 173.

DiagramDescription nous proposons la propriété *AlternativeEvaluation* de type *string* permettant de noter les critiques que le concepteur peut avoir concernant une conception donnée.

Le choix d'une architecture physique peut induire des contraintes de réalisation inévitables, nous proposons ainsi une méta-propriété de « block » SysML intitulée {ImplementationConstraint} de type *String* permettant de noter ces contraintes.

E – Les autres processus de l'étape de développement

Les autres processus de l'étape de développement sont au delà de la portée de SysML. Ces processus doivent être confiés à des outils dédiés et sont hors du champs d'étude dans le cadre de cette thèse.

6.3.3 Tableau récapitulatif des extensions de SysML liées à l'IEEE 15288

Nous récapitulons dans les deux tableaux suivants les extensions présentées ci-dessus:

Tableau 6.1. Extensions induites par le standard IEEE 15288

Processus	Extensions
Cycle de conception global	✓ « System-Of-Interest » : String // Méta-propriété de type *String* de la méta-classe *package.*
Le processus de définition des besoins des parties prenantes.	✓ {RequiredContext} : String // Méta-Propriété d'un use case. ✓ {QoS},{SecurityQos},{EnvironmentalQoS} : String // Méta-propriétés d'un use case. ✓ PriorityLevel : Enumération = {1, 2, .. 10} ✓ {Priority} : PriorityLevel // Méta-propriété d'un « requirement ». ✓ « Ambiguity » : Stéréotype d'Association.

	✓ {ConformToNeed} : Boolean // Méta-propriété de « requirement ». ✓ {Stakeholder} : String // Méta-propriété d'un « requirement » qui permet de conserver la trace de la partie prenante concernée par cette exigence.
Le processus d'analyse des exigences	✓ {RequiredContext} : Métapropriété de type string des stéréotypes « FunctionalRequirement » et « PhysicalRequirement ». ✓ « SecurityRequirement », «EnvironmentalRequirement » : Stéréotypes de « ExtendedRequirement ». ✓ « ImplementationConstraint » : Stéréotype de « ExtendedRequirement ». ✓ {Risks} : *RiskDataType(RiskLevel : RiskKind, Risk :String)* // Méta-propriété de « extendedRequirement ».
Le processus de conception de l'architecture	✓ « AllocationRequirement » : Stéréotype de « requirement ». ✓ *ObtentionManners : Enumération* = {à acheter, à réutiliser, à construire}. ✓ {ObtentionManner} : ObtentionManner // Méta-propriété de « Block ». ✓ « InternalLogicalArchitecture » et « InternalPhysicalArchitecture », deux *DiagramUsage* du diagramme de description interne de blocs (IBD). ✓ « LogicalArchitectureComposition » et « PhysicalArchitectureComposition », deux *DiagramUsage* du diagramme de définition de

	blocs (BDD). ✓ « Operator » : stéréotype de block. ✓ {MinHumanCapabilities} *: string* // Méta-propriété de « Operator ». ✓ *AvailabilityType :* Enumération = {Make, Buy, Reuse}. ✓ {Availability} *: AvailabilityType* // Méta-propriété de « block ». ✓ {AlternativeIdentifier} *: Integer* // Propriété de la note *DiagramDescription* de l'IBD et du BDD. ✓ {AlternativeEvaluation} *: string* // Propriété de la note *DiagramDescription* de l'IBD et du BDD. ✓ {ImplementationConstraint} : *String* // Méta-propriété de « block ».

6.4 Extension de SysML pour les systèmes mécatroniques

Nous présentons notre méta-modèle qui sera basé sur SysML. Pour chaque vue que le processus de conception nécessitera nous réutiliserons une des vues fournies par SysML, sinon nous inclurons des profiles d'UML2 qui compléteront notre méta-modèle global. Quand nous construirons nos profiles complémentaires, ils devront être complètement indépendants du reste du modèle pour privilégier la réutilisation sur la redondance. En effet si un élément se répète dans deux profiles différents on choisira de les garder tous les deux pour avoir deux profiles complètement réutilisables, la différenciation entre les deux éléments se fait par les espaces de noms (namespace).

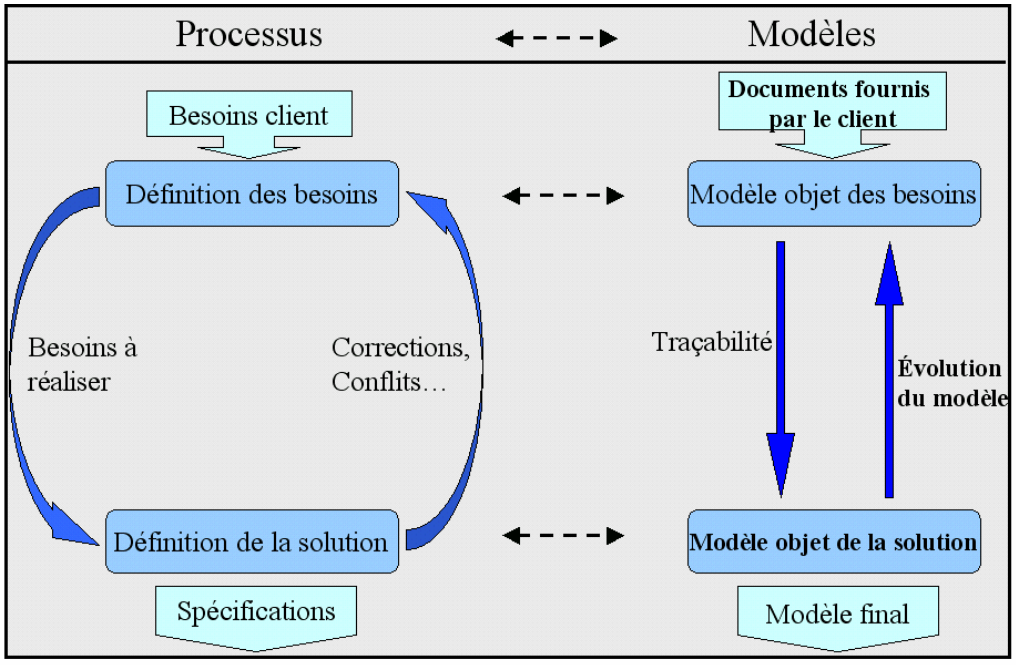


Figure 6.5. Association des modèles au processus de conception.

Nous avons adopté le langage SysML et nous l'avons étendu pour supporter la conception de systèmes mécatroniques. Nous avons défini un élément intitulé « Composant mécatronique » ou MC. Nous avons intégré dans ce composant MC une matrice DSM qui affiche les interdépendances internes du composant. Nous avons aussi intégré une DSM au système pour avoir une vue globale des interconnexions de ses composants. Nous avons ensuite intégré l'utilisation des Bond Graphs comme outil d'analyse. Puis nous avons défini des métriques et des procédures de contrôle de la qualité des modèles. Finalement nous avons décrit un processus systématique pour la mise en œuvre de ces éléments au sein de l'outil que nous décrirons dans la partie faisabilité et expérimentation. Nous allons détailler ces points un par un dans ce qui suit.

6.4.1 Extensions apportées aux connecteurs

Les diagrammes de blocks internes IBD (ou de structure interne de blocks) permettent de décrire l'interconnexion des composants d'un block. En mécatronique nous avons besoin de différencier entre ces connexions. En effet, dans un IBD un connecteur peut représenter une liaison mécanique, une connexion électrique, un

échange d'informations, etc. La nature de la connexion a une incidence sur le degré d'intégration du composant et permet de décider si une séparation des composants interconnectés est souhaitable. Pour permettre d'obtenir une meilleure conception que celle proposée par le concepteur, la différentiation des connecteurs doit permettre de détecter la pertinence de l'intégration du block. Y'a-t-il des composants qui peuvent être isolés à l'extérieur du block? Cette isolation permet d'obtenir des blocks hautement intégrés (augmenter la cohésion, voir chapitre métriques) pouvant être réutilisés. La constitution d'une base de composants réutilisables est un outil pouvant accélérer fortement la production.

Les connecteurs peuvent être utilisés conjointement avec les ports ou sans utilisation de ports (figure 6.6). Mais dans SysML seuls les ports sont différenciés par le stéréotype « FlowPort ». Par contre les connecteurs sont les mêmes pour les flux, les événements ou les signaux. L'utilisation d'un connecteur sans spécification des ports peut donc présenter une sous-spécification du connecteur. Pour remédier à cela nous proposons le stéréotype de connecteurs « FlowConnector » qui ne peut être utilisé qu'avec des « FlowPorts », nous représentons des « FlowConnector » par des flèches en trait gras pour éliminer les annotations textuelles (: « nom_de_stereotype ») qui peuvent surcharger le diagramme.

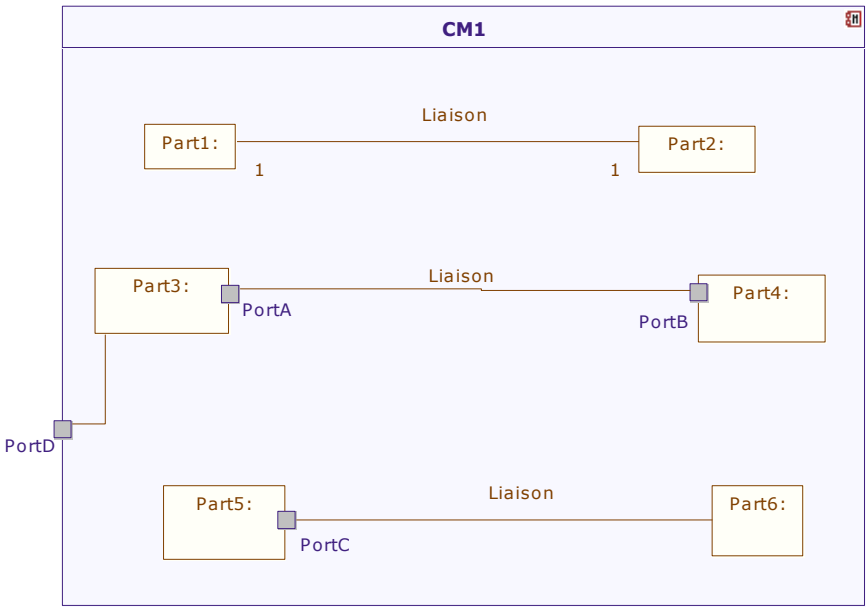


Figure 6.6. Parts, ports et connecteurs dans un IBD

Puis nous proposons aussi des tagged values (méta-propriétés) qui nous permettront de préciser si le connecteur participe à une fonction critique du block dans le cas d'un « FlowConnector » et de préciser si le connecteur participe à un échange temps-réel dans le cas d'un connecteur classique. Ainsi, nous intégrons aux connecteurs des diagrammes de blocks internes les méta-propriétés suivantes (figure 6.7):

- ✓ « RealTime » : Cette méta-propriété permet de distinguer entre un connecteur temps-réel et un connecteur normal.
- ✓ « Criticity » : Cette méta-propriété permet d'annoter qu'un lien <<FlowConnector>> intervient dans une fonctionnalité critique du système.

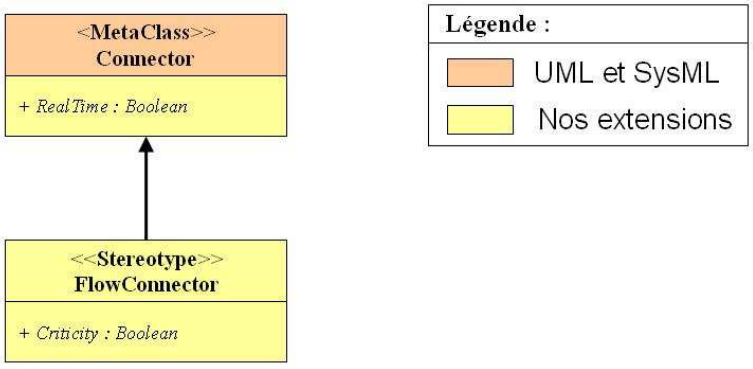


Figure 6.7. Les extensions apportées à *connector*.

6.4.2 Extensions apportées aux ports

Les ports tels que décrits dans SysML (voir présentation dans le chapitre **IV**) permettent de faire une description physique de l'échange avec l'extérieur du block. Pour plus d'expressivité nous avons besoin de rajouter sur cette couche physique du langage, une couche fonctionnelle. Cette couche fonctionnelle permettra de distinguer entre une interface de fixation et une interface de transmission d'énergie ou encore entre une alimentation électrique et un échange d'énergie électrique. Nous établissons alors les stéréotypes suivants (voir figure 6.8) :

- <<StructuralFlowPort>> : Ce stéréotype permet de décrire que la fonctionnalité dans laquelle intervient le port est de type purement structurel (fixation, assemblage, etc).
- <<FunctionnalFlowPort>> : Ce stéréotype permet de spécifier que le port intervient dans une fonction du système (transmission d'énergie, communication, etc).
- <<SupportFlowPort>> : Ce stéréotype permet de spécifier un port qui permet de fournir au système les supports lui permettant de fonctionner (fourniture d'énergie).
- <<ObservationFlowPort>> : Ce port permet de dire qu'un port est utilisé pour observer le système (observation de la température, de la vitesse, etc.).

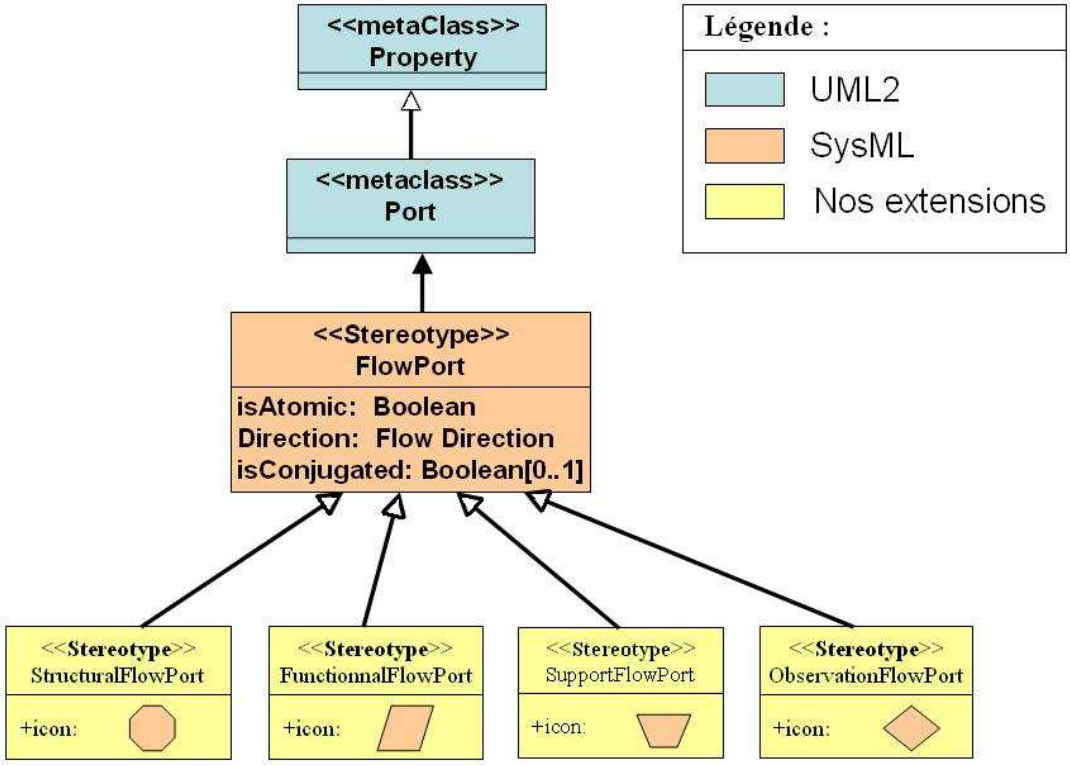


Figure 6.8. Stéréotypes du FlowPort SysML.

6.4.3 Intégration de la vue DSM au modèle SysML

6.4.3.1 Pourquoi intégrer la vue DSM au modèle ?

Pendant la construction de l'architecture du système, une décomposition est effectuée. Les sous-systèmes (composants) sont reliés par différents types de relations qui peuvent représenter des services (événements, messages, control) ou des flux pouvant être continus ou discrets (échanges énergétiques, de matière, de données etc.). L'architecture du système et les interdépendances entre ses sous-systèmes peuvent nous donner des indications sur des caractéristiques du système. Nous pouvons ainsi penser à la manière d'améliorer ces caractéristiques. Par exemple, nous pouvons obtenir un système où les sous-systèmes sont très couplés ce qui implique que la modularité interne du système est faible. Ce qui n'est pas une bonne manière de concevoir et il devient difficile d'appréhender les effets de petites modifications dans un des sous-systèmes.

La DSM est un modèle de dépendance donnant une vue globale de l'architecture d'un système de manière compacte. C'est une vue particulière d'un système qui est bien adaptée aux systèmes complexes. Les dépendances représentées peuvent être de types différents et les systèmes peuvent être hétérogènes et de nature différente (systèmes physiques, processus, etc.). Il est donc important de définir quelles informations seront représentées dans la DSM et un codage de ces informations doit ainsi être défini.

Les DSMs peuvent compléter une description UML d'un système en donnant une vue globale des interdépendances des sous-systèmes. En effet, la représentation graphique dans UML (boîtes et flèches) devient difficile à appréhender dès que le nombre d'éléments devient important. Les DSM offrent une vue compacte et globale des dépendances entre sous-systèmes. **[Browning2001]** décrit quatre applications utilisant les quatre types de DSM (Architecturale, organisationnelle, d'ordonnancement et paramétrée) et montre que les DSM répondent bien à ce besoin que l'on rencontre avec les systèmes complexes.

De plus, cette représentation matricielle est plus adaptée pour dégager des caractéristiques globales du système. Les diagrammes UML continuent à donner plus de détails sur des régions du système.

6.4.3.2 Codage des différents types de liens dans la DSM

Nous affectons à chaque lien un poids qui est calculé en fonction de deux critères qui sont le type du connecteur (ou lien) et le type des ports utilisés. Nous affectons aussi à chaque connecteur une couleur selon qu'il est standard ou de type flux :

Tableau 6.2. Codage établi pour la DSM.

Connector (standard)	**Port**	**FlowPort**
Sans contrainte temps-réel *RealTime = False*	4	*Combinaison interdite*
Avec contrainte temps-réel *RealTime = True*	8	

FlowConnector	**Port**	**FlowPort**			
		Structural	*Functionnal*	*PoweSupply*	*Observation*
Sans contrainte de criticité *Criticity = False*	*Combinaison interdite*	2	5	3	1
Avec contrainte de criticité *Criticity = True*		6	10	9	7

A travers ce codage nous cherchons à retrouver les composants mécatroniques hautement intégrés pour des soucis de réutilisation, en améliorant la cohésion et en minimisant le couplage (voir chapitre métriques). **[Enders2003]** explique que la cohésion dans un système peut être de nature diverse (logique, temporelle, fonctionnelle, communicationnelle, séquentielle) et précise que la cohésion

fonctionnelle doit être la plus recherchée pour obtenir une bonne conception. Ainsi l'affectation des poids des connecteurs est choisie de manière à privilégier les connecteurs participant à la fonction du bloc, de privilégier les connecteurs ayant une contrainte de criticité ou temps-réel. Ensuite nous avons supposé qu'un « FlowConnector » est plus important dans une architecture qu'un connecteur standard. En effet, pour les informations par exemple, une information continue est moins importante qu'une information événementielle. Nous avons aussi supposé qu'une contrainte temps-réel était moins importante qu'une contrainte de criticité.

6.4.4 Le profile ModelicaML

Pendant la construction de notre plateforme et ayant fait le choix de l'utilisation de Modelica en tant que langage de simulation, la question de l'intégration d'un profile Modelica s'est posée. Fritzson et Pop **[Pop2007]** ont crée le profile ModelicaML pour intégrer UML/SysML et Modelica en vue de supporter la modélisation et la simulation des comportements continus. Ce profile ModelicaML est un PSM (modèle spécifique à une plate-forme) permettant de décrire le modèle graphique d'une simulation Modelica. Leurs principales extensions aux langages consiste en de nouveaux diagrammes qui sont le diagramme de classes Modelica, le diagramme d'équations et le diagramme de simulation.

La création du profile ModelicaML permet d'intégrer la conception détaillée dans la plateforme SysML. Ensuite la description de la simulation qui permettra de vérifier et valider le système/composant.

Quant à la simulation prenant en compte les domaines technologiques (mécanique, électrique, Bond Graphs etc.), certaines considérations sont à prendre en compte. Tout d'abord les outils pouvant compiler et simuler du code Modelica tels que Dymola ou MathModelica fournissent un éditeur graphique permettant de concevoir des systèmes physiques et mettent à disposition du concepteur une panoplie de bibliothèques relatives à des domaines technologiques aussi variés que la mécanique,

l'électronique, l'hydraulique, les Bond Graphs, etc. L'utilisation de ces interfaces est plus adaptée que l'utilisation d'un outil UML muni d'un profile ModelicaML car l'utilisation des outils cités ci-dessus permet de travailler sur un modèle qu'on peut simuler et améliorer au fur et à mesure. Nous considérons que ce genre d'outils doit être utilisé comme l'atelier de réalisation virtuelle et de simulation. Après la conception générale utilisant SysML, nous générons le modèle Modelica du système ou d'une partie du système pour pouvoir utiliser ce modèle comme un premier prototype et une base de travail pour l'atelier de réalisation virtuelle et de simulation. Cette approche permet d'utiliser les points forts de l'outil SysML pour la conception générale et de ne pas perdre les avantages qu'offrent les outils de simulation.

Nous proposons donc d'utiliser le profile ModelicaML pour les simulations ne prenant pas en compte le domaine technologique. Pour les simulations prenant en compte le domaine technologique nous devons construire le profile adéquat, ce profile sera un modèle spécifique à la plate-forme PSM.

6.4.5 Intégration des Bond Graphs dans SysML

Dans cette thèse nous avons pris pour exemple d'extensibilité, l'intégration de Bond Graphs pour l'analyse énergétique des systèmes. Nous avons donc crée le profile *BlockBondGraphs*.

Les modèles utilisés pour l'analyse et la conception des systèmes sont aujourd'hui très nombreux (réseaux de Petri, automates hybrides, diagrammes en blocs, etc.). Un ces modèles est par ailleurs introduit en tant que profile dans SysML et est décrit dans la spécification de SysML, ce sont les EFFBD (Enhanced Functional Flow Block Diagrams). Cette extension a pour but d'enrichir l'expressivité de SysML sans en compliquer l'utilisation et l'apprentissage car les diagrammes et les éléments de langage utilisés sont les mêmes que ceux de SysML lui-même. De plus, l'utilisation de ces modèles conjointement avec SysML permet de créer ce lien entre la phase de conception et celle de l'analyse ou du prototypage virtuel Pour aller vers cette architecture, nous intégrons les Bond Graphs dans le langage SysML en tant que

profile. Dans **[Borutzky1999]** Borutzky fait une mise en correspondance entre les Bond Graphs et la modélisation orientée objet. Nous constatons aussi cette correspondance et nous la décrivons par un profile Bond Graphs pour SysML.

Les Bond Graphs sont un modèle de conception destiné à représenter les échanges énergétiques entre composants d'un système multi-physique. Nous préconisons donc l'intégration des Bond Graphs dans la boite à outils SysML en vue de son utilisation pour la description des parties physiques du système. Cette description vise à mieux comprendre le comportement dynamique des parties physiques d'un système mécatronique pendant la phase d'analyse du système.

Les systèmes mécatroniques intègrent des technologies mécaniques, hydrauliques, électroniques, etc. Dans une approche systémique de conception on vise à étudier le système dans sa globalité, tout en s'assurant que les composants du système se comportent conformément aux besoins. L'intégration des Bond Graphs dans SysML ne vise donc pas à utiliser une approche qui séparerait l'étude d'une partie opérative de manière complètement découplée de celle d'une partie commande par contre, cette intégration vise à obtenir une plate-forme de conception de systèmes mécatroniques permettant à des ingénieurs de différents domaines d'intégrer leurs conceptions dans un même outil tout en leur permettant d'utiliser leurs propres outils spécifiques pour les analyses techniques spécialisées. Les Bond Graphs peuvent en effet être simulés en utilisant des outils tels que Dymola ou OpenModelica munis de la bibliothèque de BondGraphs BondLib.

Pour des soucis de clarté et d'organisation, nous avons jugé utile de consacrer un chapitre indépendant dédié à la présentation de ce profile *BlockBondGraphs* de manière détaillée, chapitre intitulé « Le profile SysML Bond Graphs ».

6.4.6 Le composant mécatronique

Le block SysML représente un composant modulaire. Pour l'aide à la conception des systèmes mécatroniques nous créons une extension de ce composant en

y intégrant des éléments nécessaires pour décrire un composant mécatronique. Nous appelons cette extension « composant mécatronique » ou CM qui est un composant hautement intégré contenant des parties opératives, des parties commandes, des actionneurs et des capteurs. Nous définissons donc ce CM comme un stéréotype de block SysML. Un CM est décrit par une méta-propriété (*Fonctionnality : string*) définissant ses fonctions dans le système et une méta-propriété (*CorrectConditions : string)* spécifiant ses contraintes de fonctionnement correct. Un composant mécatronique contient une DSM qui décrit les interconnexions de ses composants. Nous définissons des compartiments pour séparer les différents *parts* qui forment le composant mécatronique. La figure 6.9 suivante représente le stéréotype « MechatronicComponent » tel que nous le définissons :

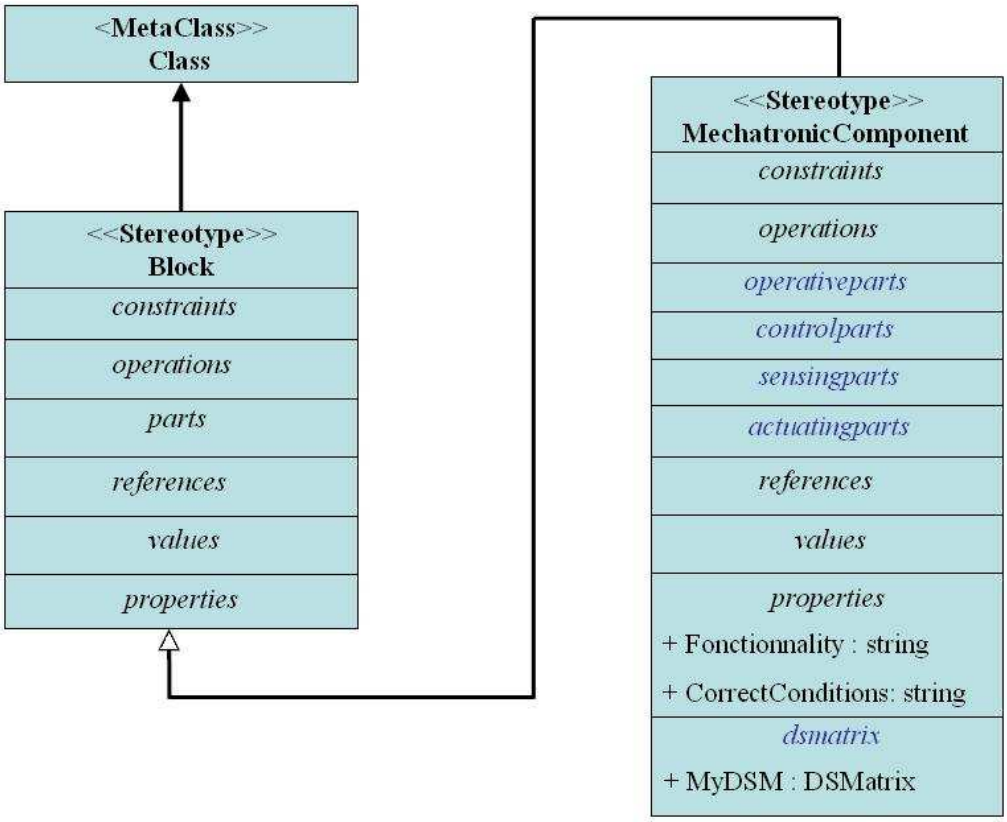


Figure 6.9. Le stéréotype « MechatronicComponent ».

Un système peut contenir un ensemble de composants mécatroniques. Un système ou un composant mécatronique sont généralement décomposés en un

ensemble de parties contrôle, de parties opérative, de capteurs, d'actionneurs et, récursivement, d'autres composants mécatroniques.

Nous décrivons dans le schéma suivant les stéréotypes nécessaires pour le diagramme de définition de blocks (figure 6.10) ainsi que pour le diagramme de structure interne de blocks IBD (figure 6.11). Les couleurs représentant chaque stéréotype permettra d'avoir une représentation graphique du stéréotype en allégeant les diagrammes.

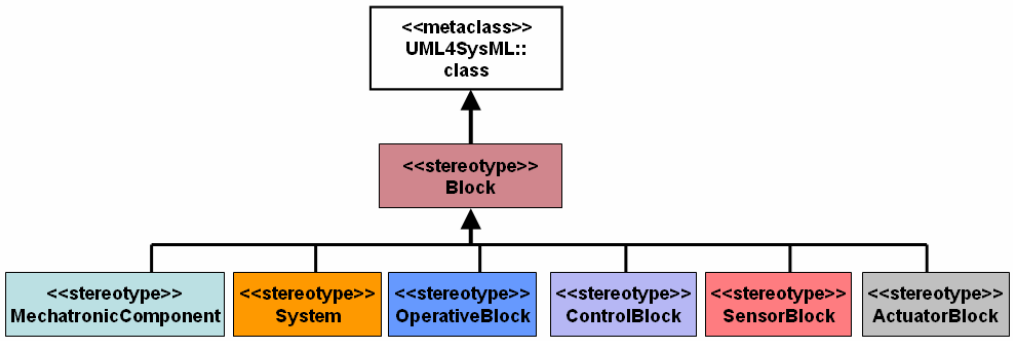


Figure 6.10. Stéréotypes du Block SysML.

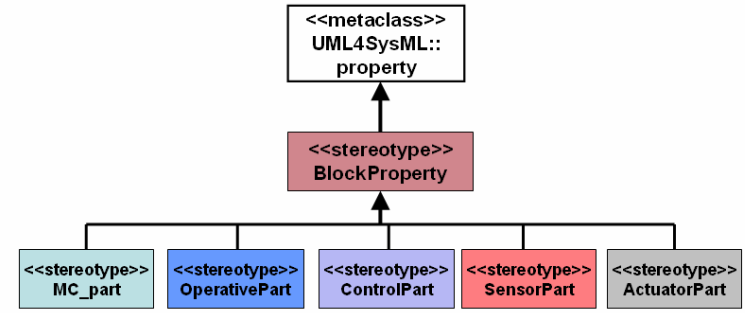


Figure 6.11. Stéréotypes de la BlockProperty SysML.

Une question importante se pose ici quant à la relation entre un Block et une BlockProperty. Est-ce le même rapport qu'on trouve entre une classe et ses attributs ? En réalité, le diagramme IBD n'est pas un diagramme décrivant des instances. Un IBD est un diagramme de structure composite tel que décrit dans la spécification d'UML2 **[UML2007].** C'est donc une structure interne qui représente des connections entre des rôles et non entre des instances **[Weilkiens2008]**. Cette nuance peut générer des difficultés de compréhension et donc de communication entre des utilisateurs qui n'ont pas forcément de connaissances en UML2. Certains ingénieurs habitués à travailler

avec des langages tels qu'AADL **[AADL2006]** pourraient comprendre qu'un Block représente le système, alors que ce n'est qu'un plan du système. En effet en AADL les deux niveaux Component*Type* et Component*Implementation* correspondent bien à un système et sa composition non à un plan du système et à une structure possible de ce système.

6.4.7 Description et allocation du comportement hybride du composant mécatronique

La description du comportement hybride du composant mécatronique se fait en **allouant** (allocation SysML)aux blocks des statemachines pour la commutation entre états et en **allouant** des diagrammes de blocks paramétriques pour détailler les comportements continus (figure 6.12) :

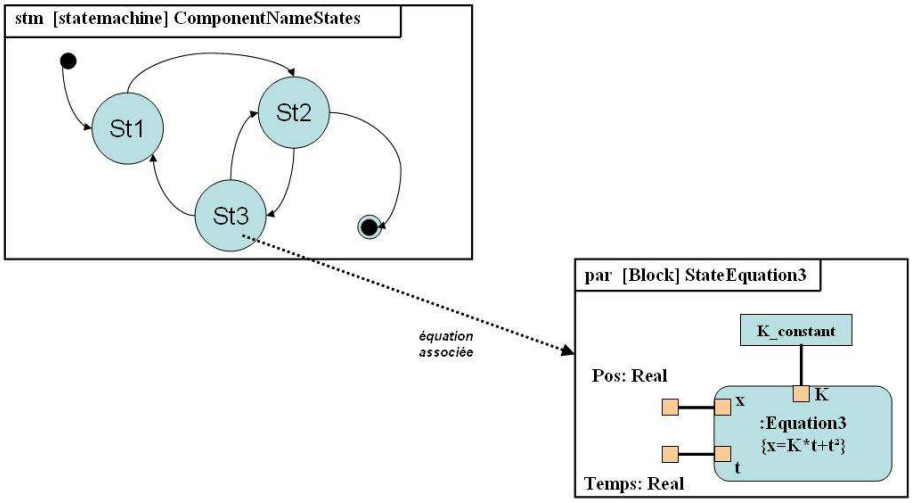


Figure 6.12. Description du comportement hybride
du composant mécatronique.

Nous avions présenté au paragraphe 4.3, les allocations en SysML et nous avions rapporté les critiques faites à ces allocations. Il est vrai que ces allocations sont très abstraites et peuvent être interprétées différemment, sémantiquement elles n'imposent rien. Par contre, SysML étant un langage destiné à l'ingénierie système, il est important de pouvoir disposer d'un tel élément général, d'autre part, pour exprimer l'allocation physique dont parle Mickael Latta (dans le même chapitre 4.3), nous proposons une extension à l'allocation qui exprime l'allocation physique et nous

l'appelons « deploiment ». Le concept de déploiement est nécessaire pour allouer physiquement un comportement à une structure.

6.4.8 Patterns ou patrons de conception pour la mécatronique

6.4.8.1 Le pattern de décomposition du composant mécatronique

La décomposition CM est décrite par un diagramme de définition de block et d'un diagramme de structure interne de block. Le Pattern de décomposition du système mécatronique permet de fournir une base de travail pour tout CM. Nous définissons ce Pattern par ce BDD (figure 6.13):

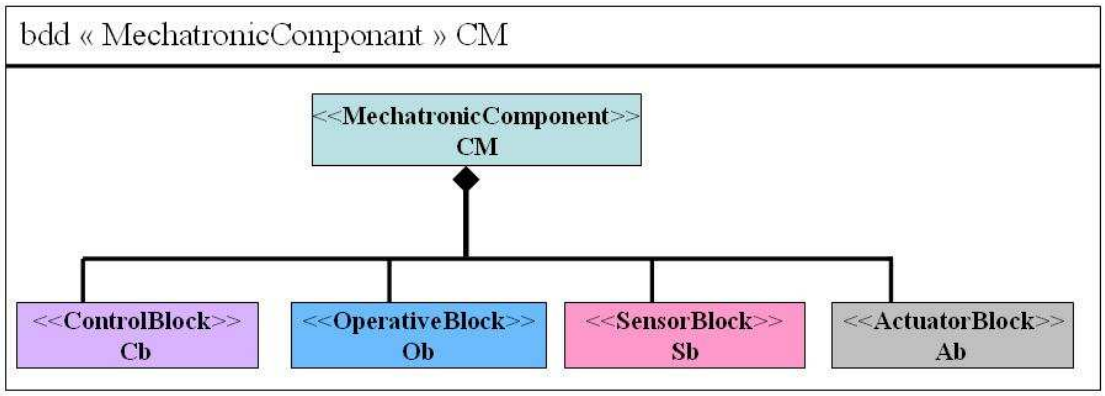


Figure 6.13. Pattern de décomposition du composant mécatronique.

6.4.8.2 Le pattern de création d'un IBD pour le composant mécatronique muni des extensions apportées

Un système mécatronique est un système intégré multi-technologique, **[Isermann2000]** décrit l'architecture globale d'un système mécatronique par le schéma suivant (figure 6.14) :

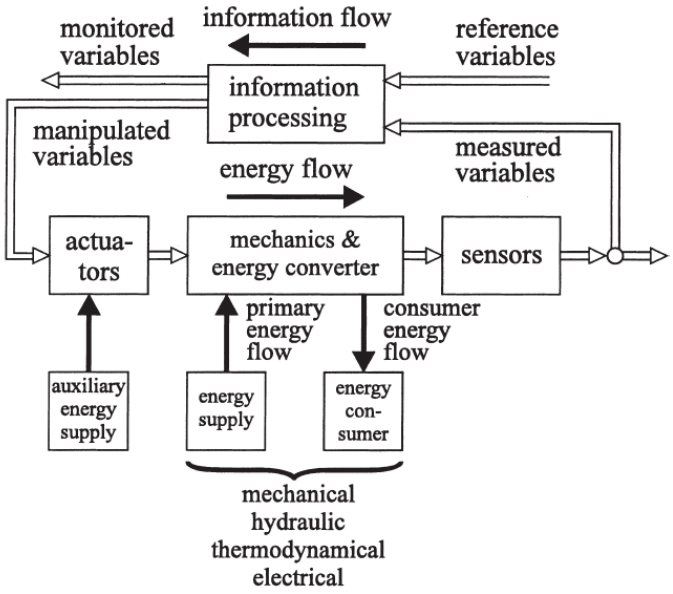


Figure 6.14. Architecture globale d'un système mécatronique **[Isermann2000]**.

Nous proposons ainsi un pattern de création de composant mécatronique pour l'aide à la conception en respectant l'architecture de base proposée par Isermann dans la figure ci-dessus. Nous obtenons le diagramme IBD suivant (figure 6.15):

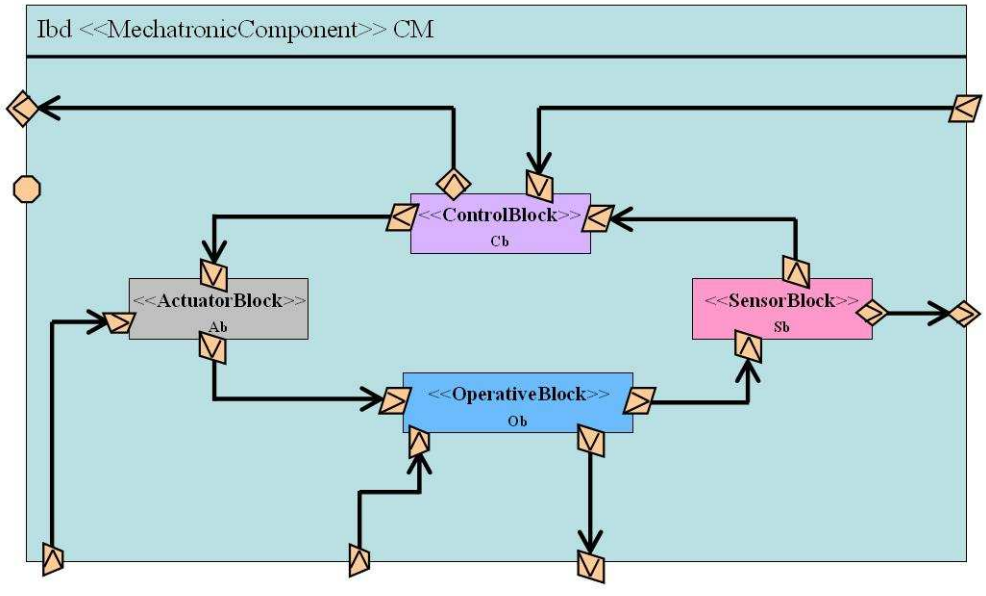


Figure 6.15. Pattern de création d'un IBD pour un composant mécatronique[22].

[22] Les connecteurs en trait fort sont des « FlowConnector ». Le port circulaire est un « StructuralFlowPort » et il représente un point de fixation du CM.

6.4.8.3 Le Pattern du système à boucle fermée dédié au bloc de contrôle

Un système à boucle fermée peut être décrit par le diagramme suivant (figure 6.16):

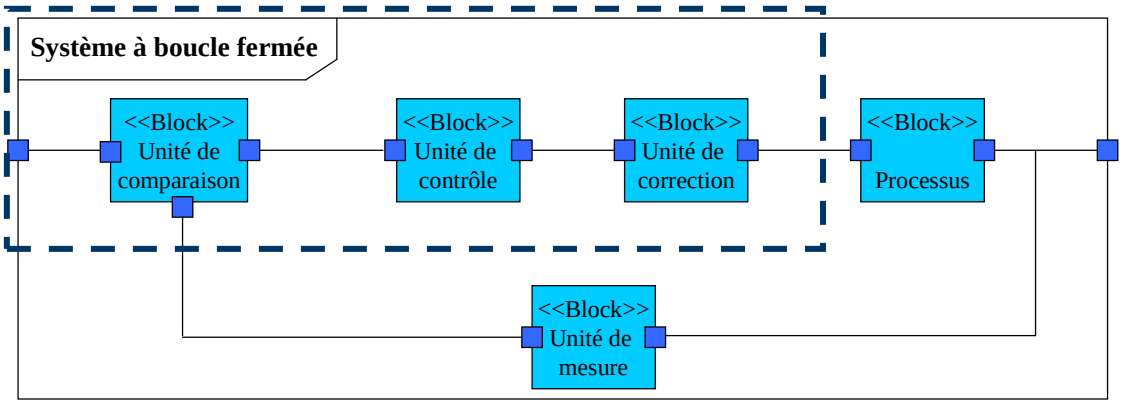


Figure 6.16. Système à boucle fermée.

Nous décrivons le pattern correspondant au système à boucle fermée pour un block de contrôle. Les blocks ‘Processus’ et ‘Unité de mesure’ ne font pas partie de cette partie contrôle. Nous obtenons le pattern suivant (figure 6.17):

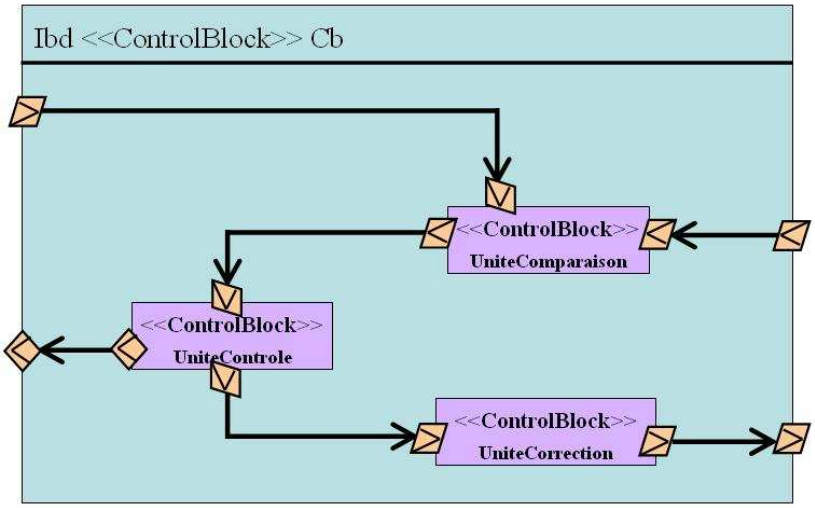


Figure 6.17. Pattern dédié à la création
d’un contrôle à boucle fermée.

6.5 Cycle de conception 15288 appliqué aux modèles adoptés

Dans cette partie nous présentons notre application du standard IEEE 15288 en tenant compte des modèles induits par l'application de l'architecture MDA. Ce qui résulte en l'instanciation d'une étape de conception du système contenant une sous-étape de développement du prototype virtuel, suivie d'une étape de développement du système. Dans la figure 6.18 suivante le terme système considéré (System-of-Interest) est le système ayant pour frontières les limites des intérêts et des responsabilités de la personne qui l'appréhende. Dans le processus décrit ci-dessous, nous lions, par extension, le système considéré à l'étape du cycle de conception, réalisée par la ou les personne(s) concernée(s). Le système objectif est la perception que cette (ces) personne(s) a (ont) du système réel.

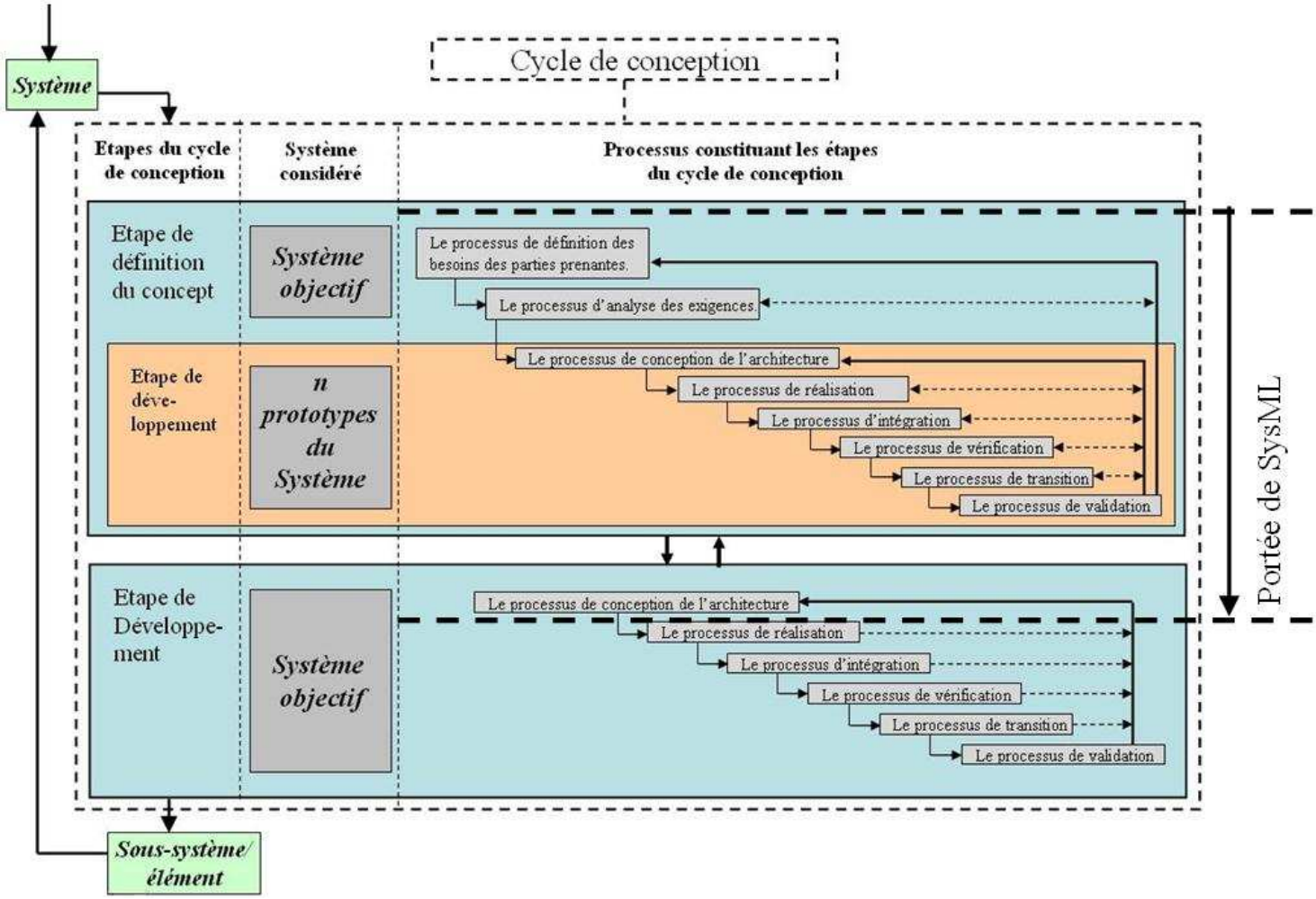


Figure 6.18. Les étapes du cycle de conception,
adaptation du standard IEEE 15288 **[IEEE15288]**.

Le cycle démarre en ayant pour objectif la conception du système. Au départ, le système considéré est donc le système objectif. La définition du concept est une phase d'identification de nouveaux concepts, d'évaluation de la faisabilité de solutions et d'identification des besoins des parties prenantes et exigences préliminaires sur le système. Cette phase commence par un processus de définition des besoins puis une phase d'analyse de ces besoins en considérant le système objectif, se poursuit, ensuite, par une étape de développement d'un prototype du système, le système considéré dans cette partie est donc un prototype du système. Cette étape de développement du prototype est réitérée jusqu'à obtention d'un prototype satisfaisant. La deuxième phase est la phase de développement du système objectif qui commence par un processus de conception de l'architecture.

Le cycle peut être réitéré récursivement pour des sous-systèmes ou des éléments du système considéré. Dans cette thèse nous nous penchons sur les étapes concernées par SysML. Dans la figure précédente nous pouvons voir que la portée de SysML s'arrête au processus de conception de l'architecture du système. Nous allons décrire l'application de ce cycle de conception à SysML muni des extensions décrites précédemment. La figure 6.19 représente le cycle associé à l'étape de définition du concept en faisant la mise en correspondance entre les activités des processus élémentaires et les modèles SysML (étendu).

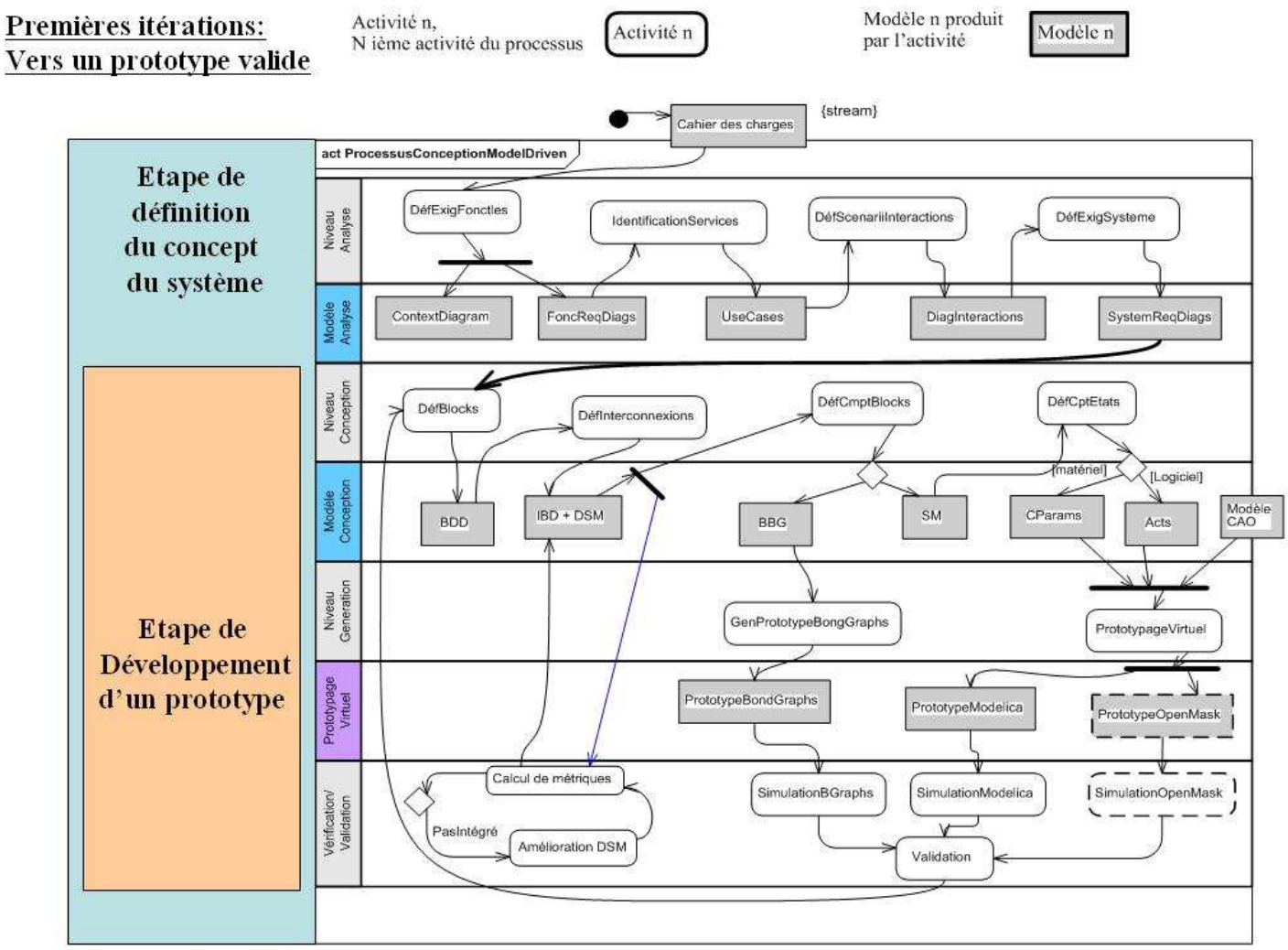


Figure 6.19. Processus associé à l'étape de définition du concept du système appliquée à SysML.

Le cycle continue dans la figue 6.20 avec l'étape de développement du système, en faisant encore la mise en correspondance des activités avec les modèles.

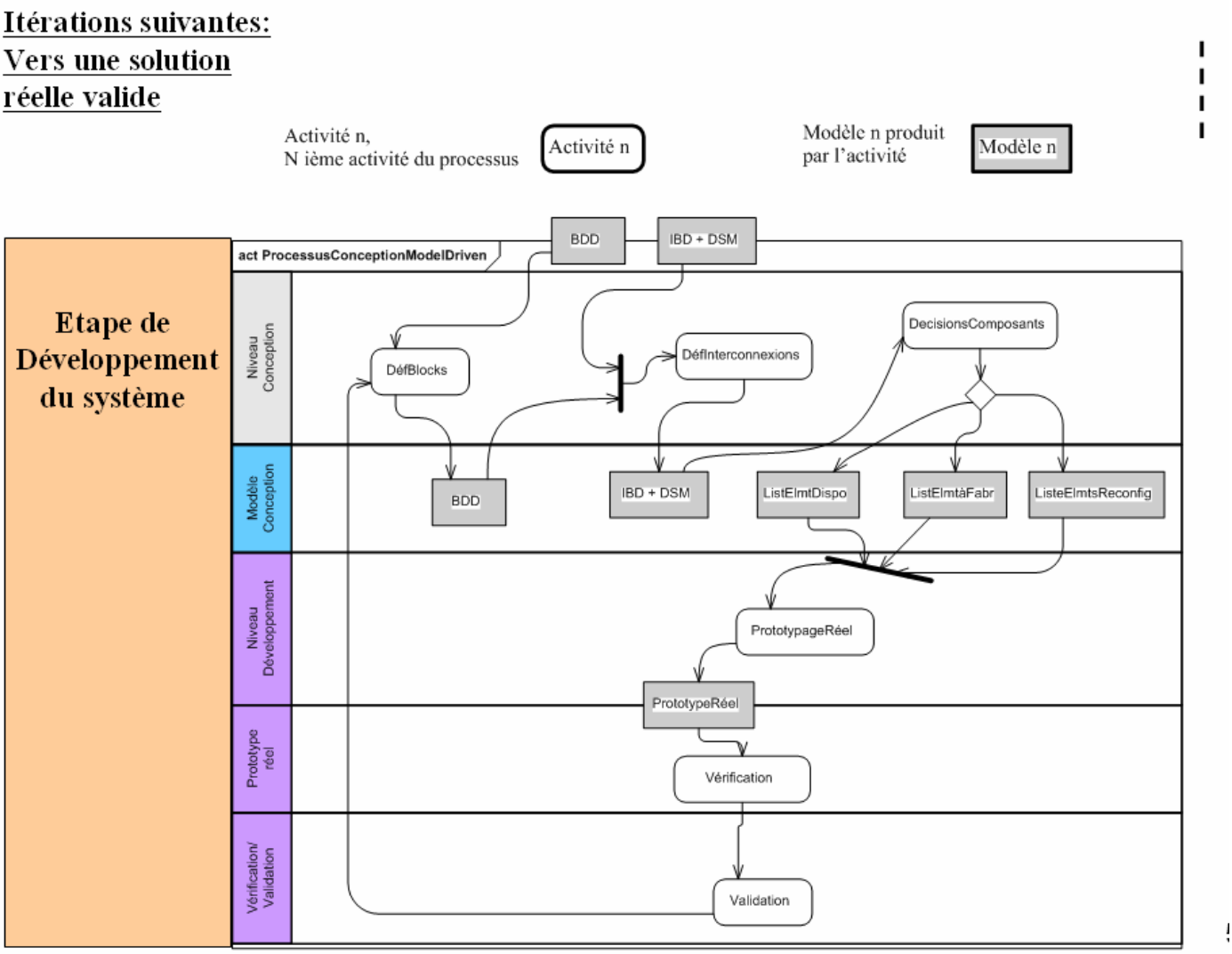


Figure 6.20. L'étape de développement du système appliquée à SysML.

6.6 Mise en correspondance des processus avec les modèles SysML

Dans ce qui suit, nous décrivons les mises en correspondances entre le cycle décrit précédemment et les modèles adoptés.

6.6.1 Le processus de définition des besoins des parties prenantes

Ainsi, le processus de définition des besoins des parties prenantes du cycle 15288 appliqué correspond aux activités de (figure 6.21):

i. définition des exigences sur les fonctions moyennant le diagramme de contexte et le diagramme des exigences fonctionnelles (diagramme des exigences muni du *DiagramUsage « FunctionalRequirements »*).

ii. identification des services moyennant le diagramme des use cases.

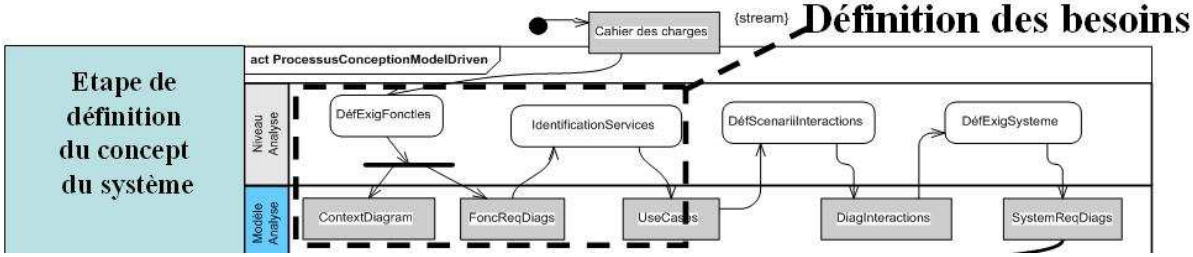


Figure 6.21. Le processus de définition des besoins appliqué à SysML étendu.

6.6.2 Le processus d'analyse des exigences

Le processus d'analyse des exigences correspond aux activités de (figure 6.22):

i. Définition des scenarii et des interactions moyennant le diagramme des interactions.

ii. Définition des exigences sur le système moyennant le diagramme des exigences sur le système (diagramme des exigences muni du *DiagramUsage « SystemRequirements »*).

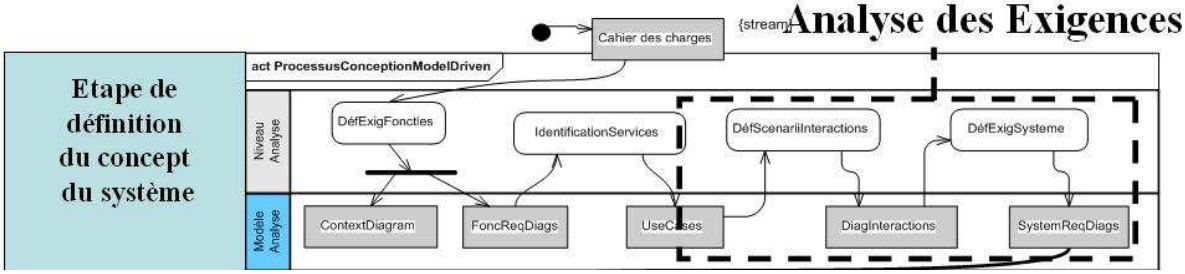


Figure 6.22. Le processus d'analyse des exigences appliqué à SysML étendu.

6.6.3 Le processus de conception de l'architecture d'un prototype

Le processus de conception de l'architecture d'un prototype correspond aux activités de (figure 6.23):

i. Décomposition du système moyennant le diagramme de définition des blocks BDD.

ii. Définition des interconnexions entre composants du système moyennant le diagramme interne des blocks IBD.

iii. Puis génération des DSM correspondantes à chaque composant et au système.

iv. Calcul des métriques de vérification de la qualité des modèles. Et proposition d'améliorations à partir de l'algorithme de réorganisation de la structure interne d'un block qui utilise la DSM de ce block en cas de métriques faibles.

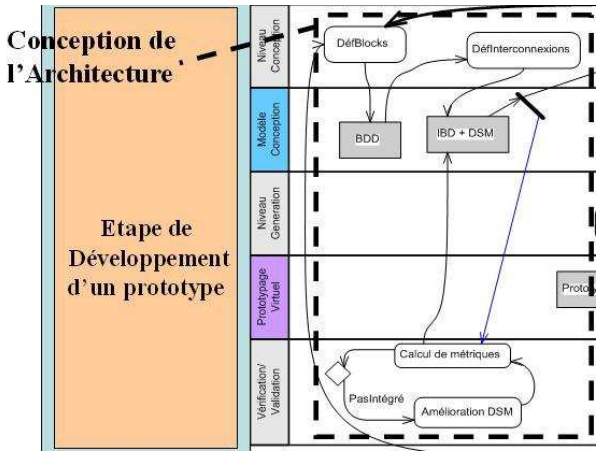


Figure 6.23. Le processus de conception de l'architecture d'un prototype appliquée à SysML étendu.

6.6.4 Le processus de réalisation et d'intégration

Le processus de réalisation et d'intégration correspond aux activités de (figure 6.24):

i. Définition du comportement des blocks moyennant les Statemachines.

ii. Définition du Bond Graphs d'un block en cas d'une analyse énergétique.

iii. Définition des comportements continus moyennant le diagramme des contraintes paramétriques pour un composant matériel et le diagramme des activités pour un composant logiciel.

iv. Liaison du modèle CAO du composant (chemin du fichier).

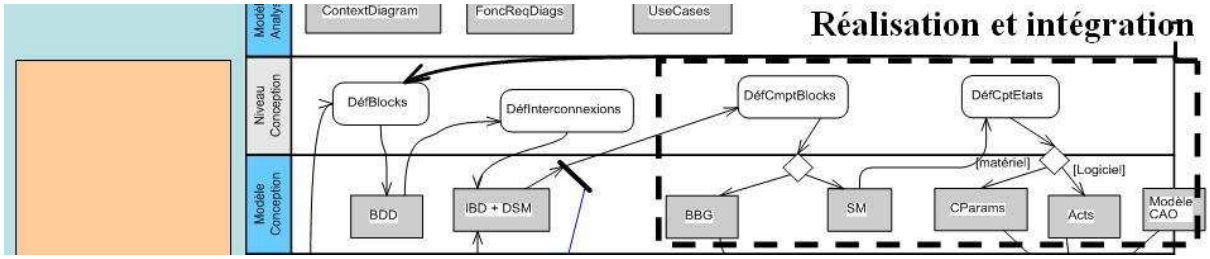


Figure 6.24. Le processus de réalisation et d'intégration appliqué à SysML étendu.

6.6.5 Le processus de vérification et de transition et le processus de validation

Le processus de vérification et de transition et le processus de validation correspondent aux activités de (figure 6.25) :

i. Génération du prototype Bond Graph en terme de code Modelica utilisant la bibliothèque BondLib et simulation du prototype en vue de la validation du composant.

ii. Génération d'un prototype virtuel en terme de code Modelica.

iii. Simulation du prototype virtuel en intégrant le modèle CAO géré par la plateforme de simulation OpenMask. Cette partie du cycle de conception est en dehors du champ d'étude de cette thèse, elle est le sujet d'une thèse intitulée « Simulation distribuée des systèmes mécatroniques » réalisée par Hadj Amor Hassen au sein du LISMMA **[HadjAmor2008]**.

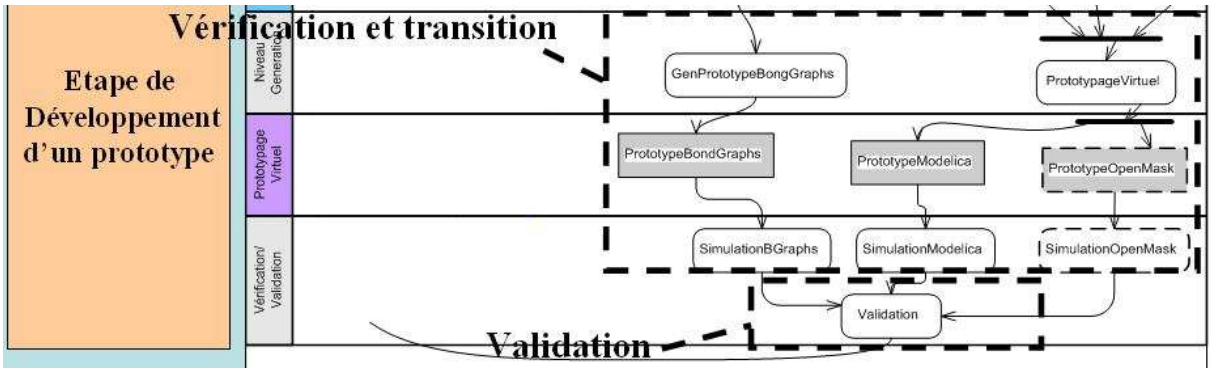


Figure 6.25. Le processus vérification et de transition et
le processus de validation appliqués à SysML

6.6.6 Le processus de conception de l'architecture du système

Le premier processus de l'étape de développement du système est le processus de conception de l'architecture qui est la limite de la portée de SysML. Le processus de conception de l'architecture du système correspond aux activités de (figure 6.26):

i. Décomposition du système moyennant le diagramme de définition des blocks BDD.
ii. Définition des interconnexions entre composants du système moyennant le diagramme interne des blocks IBD.
iii. Puis génération des DSM correspondantes à chaque composant et au système.
iv. Décisions sur les composants (Achat, Fabrication, Réutilisation) moyennant les extensions proposées dans le paragraphe (*6.3.2-D*).

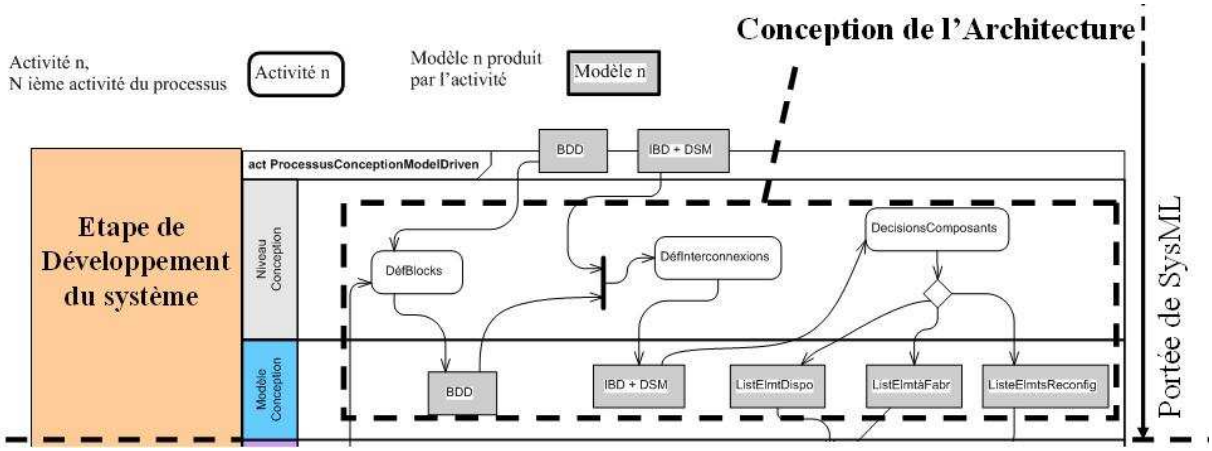


Figure 6.26. Le processus de conception de l'architecture du système appliqué à SysML étendu.

6.7 Conclusion

Nous avons présenté la méthodologie proposée intitulée MISSyM. Nous avons décrit le processus IEEE 15288 adapté à l'architecture de modèles MDA. Ensuite, nous avons présenté les modèles en question de manière détaillée, en définissant les extensions utiles aux activités de l'ingénierie système (processus techniques seulement) et celles utiles à la conception des systèmes mécatroniques. Nous avons présenté l'intégration des DSM et de métriques pour fournir un outil de vérification de la qualité des modèles et d'aide à la re-conception. De plus, nous avons présenté

l’intégration des Bond Graphs pour l’analyse énergétique et pour mettre en relief les capacités d’extension de cette méthodologie (capacités héritées de SysML).

Nous allons, maintenant, détailler ce profile BlockBondGraphs et les métriques proposées que nous avons séparés dans des chapitres à part entière pour faciliter la lecture de ce document.

Chapitre 7

VII - Le profile SysML Bond Graphs

7.1 Introduction

Les Bond Graphs sont un outil efficace pour l'analyse énergétique de systèmes physiques hétérogènes. Ainsi cette intégration des Bond Graphs permettra d'offrir un outil d'analyse pour le concepteur. Dans ce chapitre nous présentons le profile pour SysML qui intègre l'utilisation des Bond Graphs. Nous utilisons une extension des diagrammes de blocks internes (ou Internal Block Diagram IBD) pour ce que ce diagramme a en commun avec la sémantique d'un Bond Graph. En effet, ces deux diagrammes proposent une vue statique d'un système. Nous développons cette argumentation dans le paragraphe introductif. Puis nous détaillons notre extension et nous illustrons son application par un exemple.

7.2 Le Profile SysML Bond Graphs

Nous avons intitulé ce profile Block Bond Graphs BBG car dans cette proposition nous associons un Bond Graph à un block qui est l'élément de base dans SysML. Pour donner une sémantique formelle à un profile UML, une approche consiste à avoir une étape de traduction du profile vers un langage dont la sémantique formelle est ensuite établie. Cette approche est utilisée par W. Damm et al dans **[Damm2005]** pour donner une sémantique formelle à rtUML à travers la traduction d'un modèle rtUML dans le langage krtUML, ensuite ils établissent la sémantique formelle de krtUML en utilisant le formalisme des systèmes de transitions symboliques. De même, la sémantique formelle du profile présenté ici est établie par traduction du profile vers le langage Modelica muni de la bibliothèque BondLib qui est un modèle mathématique.

7.2.1 La création d'un langage graphique adapté à un domaine

Pour créer un langage graphique orienté objet adapté à un domaine particulier nous pouvons citer quatre possibilités différentes (A, B, C et D ; voir figure 7.1).

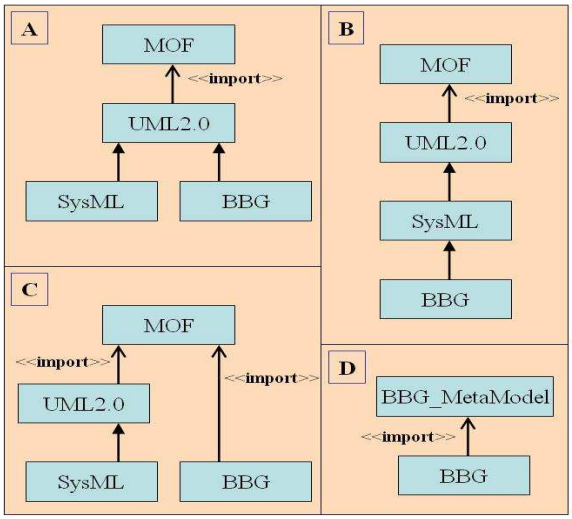


Figure 7.1. Les 4 Possibilités pour construire un langage

- Méthode A : Nous créons un profile d'un langage graphique existant tel qu'UML qui est basé sur un méta-modèle, MOF **[MOF2007]** en l'occurrence et qui dispose de mécanismes d'extension. Cette méthode est celle utilisée pour la construction de SysML ainsi que plusieurs autres profiles tels que le « UML Profile For CORBA », Le « UML Profile For System On chip » (UML SoC, 2006), Le « UML Testing Profile » et MARTE (UML Profile for Modeling and Analysis of Real-time and Embedded Systems **[MARTE2007]**), qui sont des profiles standardisés ou en cours de standardisation par l'OMG ou par d'autres organismes. Beaucoup d'autres profiles propriétaires existent aussi tels que le « Rational UML profile for business modeling ».
- Méthode B : Nous créons une extension d'un profile existant d'un langage graphique. Dans cette situation on peut envisager d'étendre SysML pour des domaines particuliers. Cette solution a l'avantage d'être directement réalisable sur tout outil implémentant SysML et les mécanismes classiques d'extension d'UML.
- Méthode C : Nous créons un nouveau langage basé sur le méta-modèle d'UML, et dans ce cas on profite du méta-modèle MOF mais nous ne pouvons plus bénéficier des outils CASE existants sur le marché.

- Méthode D : Nous créons un nouveau méta-modèle à partir duquel on construit un nouveau langage qui aura l'avantage de correspondre complètement au domaine visé. Cette solution n'est avantageuse que lorsque l'on ne peut pas utiliser l'une des autres méthodes car l'outil doit être construit à partir de zéro même si des outils commencent à proposer des « usines de logiciels » (ou software factories) qui génèrent des éditeurs graphiques munis de générateurs de codes, tel que les DSL Tools de Microsoft **[DSL_Tools].**

L'un des avantages majeurs de SysML est qu'il bénéficie des outils UML déjà sur le marché depuis plus d'une quinzaine d'années. Ainsi, si la spécification de SysML introduit quelques incompatibilités avec UML, nous n'en ajoutons pas. En d'autres termes, si SysML n'est pas exactement un profile UML, notre proposition doit être, formellement, un profile SysML. Nous ne créerons donc que des extensions aux éléments du langage SysML.

Nous avons envisagé deux solutions :

- utiliser les diagrammes d'activités pour supporter notre diagramme Bond Graphs,
- utiliser les IBD (internal block diagram).

Dans l'approche utilisant les diagrammes d'activités **[TURKI2005]**. Nous avons construit un profile indépendant de SysML mais pouvant être utilisé conjointement avec SysML. Nous avions donc la configuration A de la figure 7.1.

Nous choisissons d'appliquer la méthode A ou B car ces deux méthodes assurent l'intégration directe de notre profile dans tout outil implémentant SysML (avec la méthode A permet de l'intégrer aux outils implémentant UML et pas nécessairement SysML), ce qui permet d'intégrer en même temps ce nouvel outil avec les autres outils déjà disponibles dans SysML et ainsi enrichir toute la plate-forme.

7.2.2 Choix du diagramme

Dans ce profile BBG, nous privilégions l'aspect sémantique du diagramme contrairement à l'approche utilisant les diagrammes d'activités **[TURKI2005]**. En effet, le diagramme d'activités a une différence sémantique majeure car c'est un diagramme dynamique qui est associé à un modèle d'exécution, de la même manière qu'un GRAFCET, un réseau de Petri, un automate hybride, etc. Ce diagramme d'activités présente de grands avantages syntaxiques que sont les nœuds et que nous avons utilisés pour décrire les jonctions 0 et 1 des Bond Graphs, mais il y a un risque de confusion sur la sémantique du diagramme d'activités étendu aux Bond Graphs, car un Bond Graphs est un modèle statique (du point de vue du langage pas du système), c'est-à-dire qu'il ne met pas en œuvre un modèle d'exécution ; c'est une interconnexion de composants.

Dans SysML, nous disposons d'un diagramme dédié à ce genre de descriptions qui est le diagramme des IBD. Dans cette proposition nous nous basons donc sur ce diagramme pour fournir un profile SysML pour la description de Bond Graphs. Nous proposons, donc, un profile SysML et non UML. Le profile BBG n'est donc pas indépendant de SysML (configuration B de la figure 7.1). L'avantage de cette nouvelle configuration est de pouvoir hériter des éléments de modèle du diagramme statique IBD de SysML.

Ceci étant, nous ne devons pas négliger les aspects syntaxiques du diagramme afin qu'un BBG puisse être reconnaissable en tant que Bond Graph classique. En effet, cet outil étant destiné à des ingénieurs qui ne sont pas forcément familiers avec UML, ils doivent retrouver dans ce diagramme le maximum de ressemblance avec un Bond Graph tel qu'ils ont eu l'habitude de le décrire sous un éditeur dédié.

7.2.3 Présentation des IBD SysML

Dans un IBD on peut décrire des structures supportant les flux d'objets alors que dans les diagrammes d'activités nous décrivons des flux de contrôle, aspect qui n'existe pas dans un Bond Graph. Nous allons dans ce qui suit détailler le profile BBG en mettant l'accent sur les éléments de langage les plus importants.

7.2.3.1 Les blocks en SysML

L'élément principal introduit en SysML est le "Block" (figure 7.2). Un bloc peut avoir des ports ainsi qu'une structure interne. Il représente un système ou un composant d'un système. Des compartiments peuvent être représentés (constraints, operations, parts..) et d'autres peuvent être définis par le concepteur pour représenter des propriétés particulières.

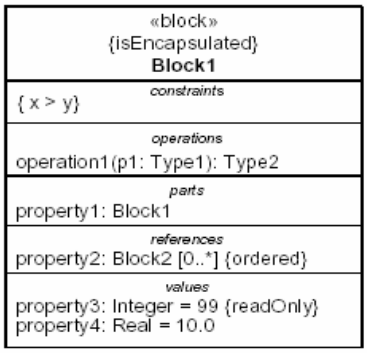


Figure 7.2. Block SysML **[SysML2007]**.

- Le diagramme des blocs internes IBD est basé sur le concept de bloc :
  - Le Block est un stéréotype de classe (figure 7.3).
  - Il permet de décrire l'architecture d'un système; un Système est composé de plusieurs autres sous-systèmes.

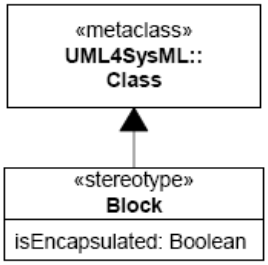


Figure 7.3. Block est un stéréotype de classe **[SysML2007]**.

- Les propriétés d'un bloc sont des stéréotypes de la propriété classique et sont au nombre de trois (figure 7.4):

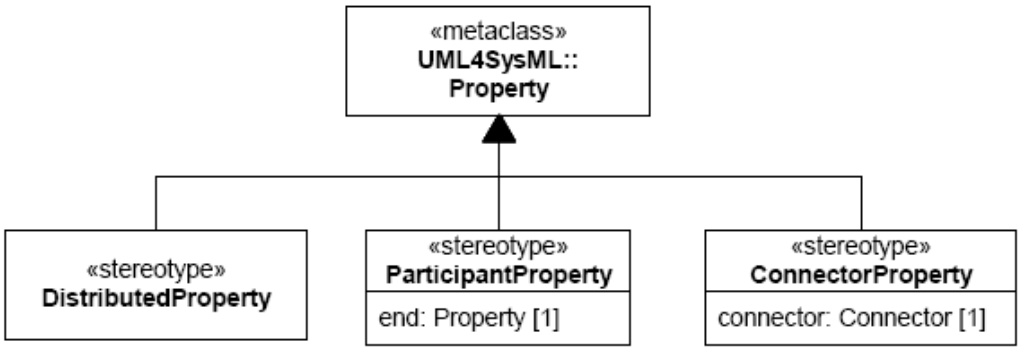


Figure 7.4 Les propriétés utilisées dans un IBD **[SysML2007]**.

7.2.3.2 Les ports en SysML

Les ports ont été étendus en introduisant le concept de « FlowPort » qui permet de représenter un point d'interaction à travers lequel il y a échange continu de données de matériaux ou d'énergie. Un <<FlowPort>> peut être atomique s'il s'agit d'un seul usage de port, d'un type de donnée ou d'un signal. Il peut aussi être non atomique si des éléments de types différents sont relayés à travers ce port (figures 7.5 et 7.6).

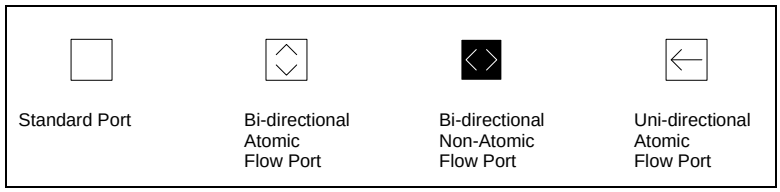


Figure 7.5. Les types de ports dans les IBD **[SysML2007]**.

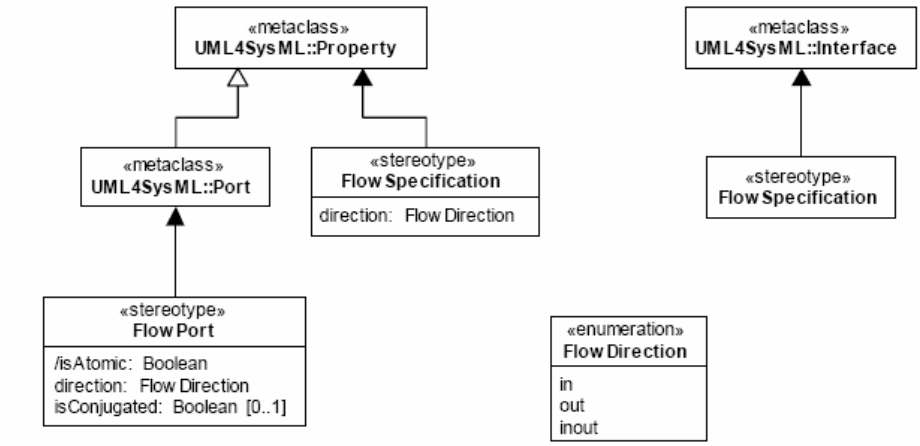


Figure 7.6. Le méta-modèles des ports **[SysML2007]**.

7.2.3.3 Autres extensions

On dispose dans SysML du stéréotype ValueType qui est une extension du DataType classique d'UML et qui contient une unité et une dimension (figure 7.7).

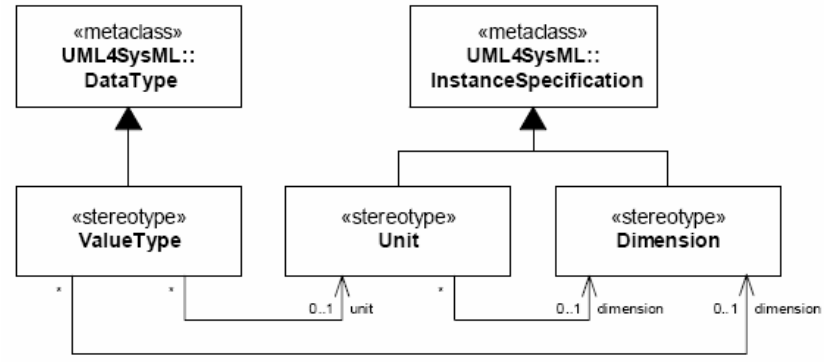


Figure 7.7. Le méta-modèle des DataTypes **[SysML2007]**.

- Le BindingConnector (figure 7.8) permet d'imposer une contrainte sur les valeurs des propriétés reliées entre deux blocs, qui doivent ainsi être égales.
- Et le NestedConnectorEnd (figure 7.8) permet de relier deux blocs qui ne sont pas dans le même niveau hiérarchique de l'architecture du système.

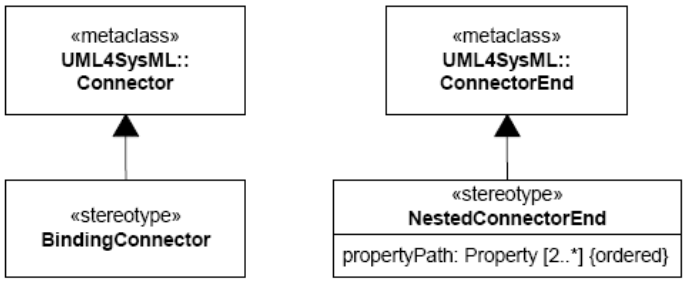


Figure 7.8. Le méta-modèle des connecteurs **[SysML2007]**.

7.2.4 Mise en correspondance des éléments Bond Graphs avec ceux de SysML

7.2.4.1 Les composants élémentaires et les nœuds

L'élément principal d'une description Bond Graph est le noeud qui peut être de deux niveaux de granularité différents ; il peut représenter un sous-système complexe

ou un composant élémentaire pouvant être assimilé à un phénomène énergétique élémentaire.

Le noeud Bond Graph est décrit comme étant un composant multi-port générique par Hales et Rosenberg **[Hales2000]**. On associe à ce nœud multi-port un block SysML. Nous créons un stéréotype global de « Block » représentant un nœud Bond Graph général puis pour chaque composant élémentaire on spécialise un stéréotype spécifique. Le stéréotype du Block SysML « BG_Block » représente le système/sous-système qui sera décrit par un Bond Graph et donc par un IBD étendu. On obtient alors les stéréotypes de la figure 7.9 :

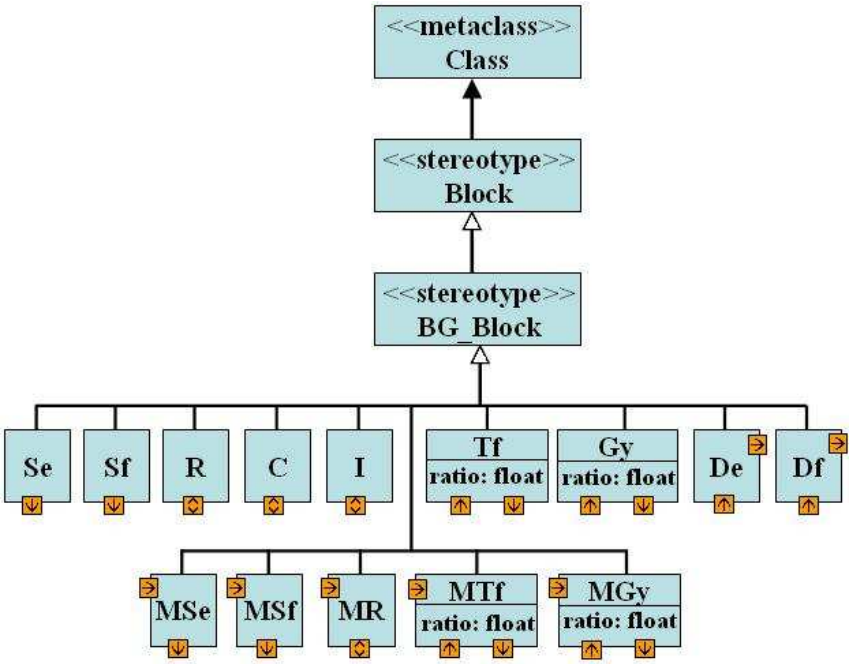


Figure 7.9. Stéréotypes BBG du Block SysML [23]

Les composants élémentaires Se, Sf, R, C et I sont des 1-ports, Tf, Gy, De et Df sont des 2-ports, on définit une FLowSpecification[24] à chacun d'entre eux. La spécification du flux associée à chacun des éléments R, C et I contient un FlowPort bi-directionnel atomique[25]. On peut définir un sens par défaut pour chacun de ces ports, ainsi pour R, C et I la valeur par défaut de la méta-propriété « direction » est « In »

[23]. Noter que UML4SysML (UML for SysML) est un package qui contient un sous-ensemble des éléments de langage d'UML2
[24]. Une FlowSpecification est un ensemble de « FlowPort » d'un block.
[25]. Le FlowPort atomique est un port utilisé dune manière unique dans un block.

(voir définition des FlowPorts page 61 **[SysML2007]**. Le sens de ces ports ne doit pas être contraint au niveau du profile car dans certains cas en Bond Graphs ces éléments peuvent être utilisés différemment. Pour Se et Sf on définit un FlowPort uni-directionnel atomique avec « direction » contrainte à «Out ». Pour De et Df on définit un FlowPort uni-directionnel atomique avec « direction » contrainte à « In » et un deuxième à « Out ». Pour Tf et Gy une entrée et une sortie sont définis par défaut dans leurs spécification de flux. Les éléments modulés sont représentés par une lettre « M » qui préfixe le nom et par un FlowPort en plus dans la spécification du flux et qui est contraint à être une entrée.

7.2.4.2 Les Jonctions Bond Graph

Dans cette proposition nous décrivons les jonctions 0 et 1 des Bond Graphs par des blocs stéréotypés. Les jonctions en Bond Graphs sont des 3-ports, ainsi on introduit trois FlowPorts bidirectionnels atomiques dans la spécification de flux de chacune des jonctions (figure 7.10). Pour l'aspect graphique, cela dépend toujours de l'outil CASE utilisé mais le plus approprié pour se rapprocher de la notation Bond Graphs est d'affecter à un block jonction une forme circulaire ou même transparente ne faisant apparaître que le label 0 ou 1 selon le stéréotype assigné.

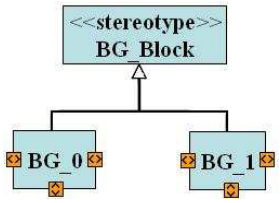


Figure 7.10. Les jonctions Bond Graph dans le BBG

7.2.4.3 Les liens ou "Bonds"

Pour représenter les deux types de liens, informationnel et énergétique, nous définissons les stéréotypes suivants de UML4SysML::Connector (figure 7.11) :

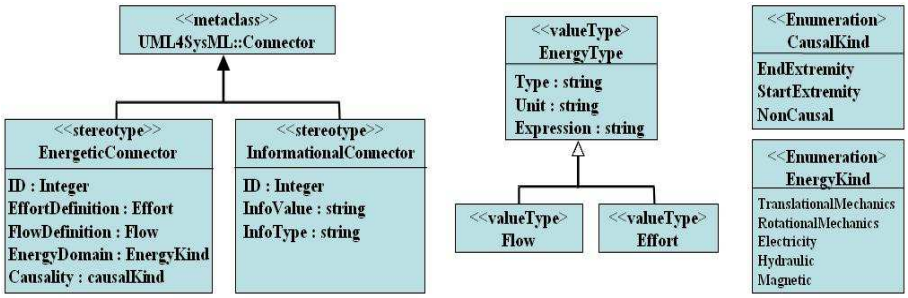


Figure 7.11. Les extensions permettant la représentation des liens

Le connecteur énergétique possède les propriétés suivantes :

ID : Numérotation des liens.

Causality : Est un CausalKind qui est une énumération et représente la causalité associée au lien. L'aspect graphique de la causalité dépend des possibilités offertes par l'outil CASE mais un trait vertical à l'extrémité du connecteur donnerait un résultat plus proche graphiquement des BG.

EnergyDomain : Est un EnergyKind et permet de décrire le domaine énergétique.

EffortDefinition (respectivement FlowDefinition) est, la définition de l'effort (du flux) par un type de donnée dédié contenant le type de l'effort (du flux), son unité et son expression.

7.2.4.4 Contraintes liées aux nouveaux éléments

Nous définissons un ensemble de contraintes pour chaque nouveau stéréotype pour obtenir des éléments de langage sémantiquement équivalents à ceux des Bond Graphs. Nous donnons ici comme exemple les contraintes liées à la causalité :

-Jonction 0: Un seul signe de causalité dans une extrémité d'un connecteur directement liée à la jonction.

-Jonction 1: Un seul flux énergétique ayant le signe de causalité dans une extrémité non adjacente à la jonction.

-De,Df : Pas de causalité donc les deux extrémités des connecteurs énergétiques reliés à De et Df sont de type « NonCausal »..

-Causalité du flux énergétique quittant Se est égale à « EndExtremity ».

-Causalité du flux énergétique quittant Sf est égale à « StartExtremity ».

-Causalité de Tf: Les deux traits doivent être ou adjacents à Tf ou dans l'autre extrémité, comme les deux connecteurs ont un sens contraire l'un à l'autre, alors les deux connecteurs doivent avoir des causalités inverses.
-Causalité de Gy: Les causalités des deux connecteurs sont mises égales.

7.2.4.5 L'extension des diagrammes en block

Les diagrammes en block sont utilisés conjointement avec les Bond Graphs pour exprimer l'aspect commande du système. Pour cette raison nous décrivons ici un profile "BlockDiagramProfile" que nous intégrons au même titre que le BBG dans SysML.

Ainsi, nous intégrons l'élément de transmittance pour les fonctions de transfert, l'élément pour les opérateurs (comparateur, additionneur) et l'élément SetPoint. Nous avons aussi intégré les stéréotypes du port UML standard pour les entrées et leurs signes, ils sont représentés graphiquement par un signe « + » et « - ». Le connecteur utilisé est le même que dans le BBG ; un connecteur unidirectionnel (figure 7.12).

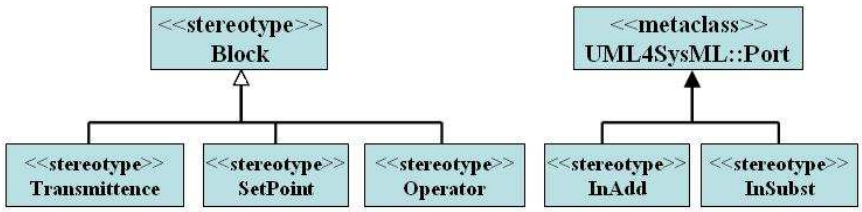


Figure 7.12. Les extensions pour le diagramme en blocks

7.2.4.6 Extension des diagrammes

Chaque diagramme SysML est conforme à une description générique (figure 7.13).

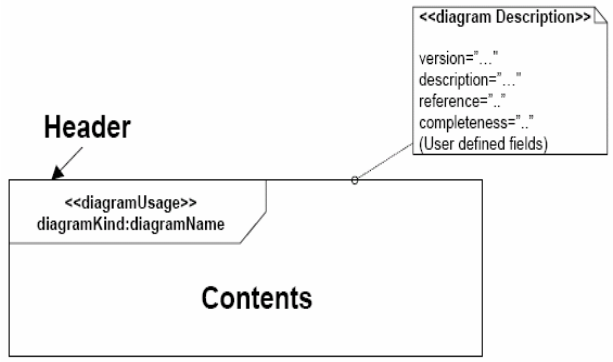


Figure 7.13. Modèle d'un diagramme SysML

Nous ajoutons un stéréotype[26] pour InternalBlockDiagram (figure 7.14). Un stéréotype d'un diagramme décrit un usage spécifique de ce diagramme.

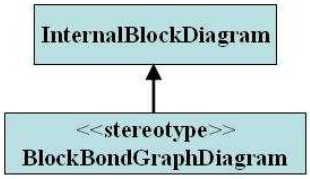


Figure 7.14. Stéréotype BBG du diagramme IBD.

Toutes les extensions décrites auparavant sont empaquetées dans un profile BBG. Un profile est un package livrable.

7.3 Application de ce Formalisme

Dans ce paragraphe nous exposons un exemple d'application de ce profile BBG. Nous utilisons un système asservi pour illustrer tous les éléments décrits dans le profile. Nous décrivons le Bond Graph causal puis le BBG correspondant (figure 7.15).

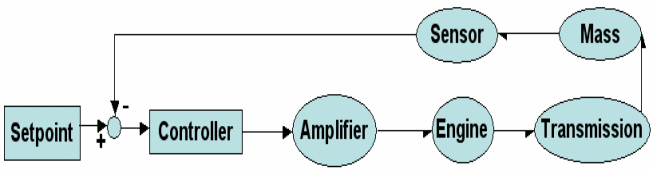


Figure 7.15. Word-Bond Graph combiné

[26]. Noter que pour les diagrammes on peut définir des stéréotypes même si les diagrammes ne font pas partie du méta-modèle UML. Ils ont leur propre méta-modèle.

au diagramme en bloc pour un système asservi.

Le Bond Graph causal résultant est représenté par la figure 7.16:

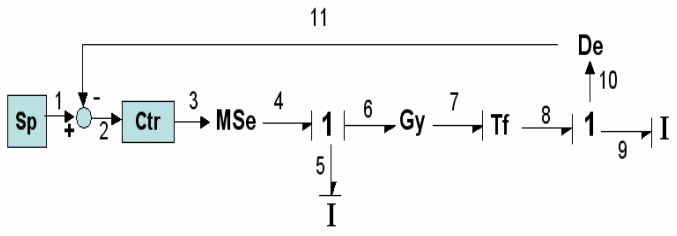


Figure 7.16. Bond Graph combiné associé

On crée un BBG associé à un block donné. En spécifiant son stéréotype comme étant « BlockBondGraphDiagram » (figure 7.17).

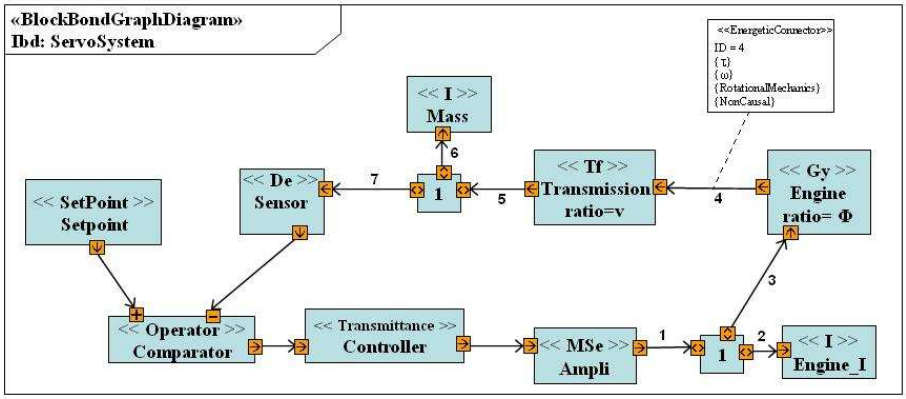


Figure 7.17. Diagramme BBG résultat

7.4 Conclusion

Nous avons proposé une extension pour SysML intégrant les Bond Graphs. Nous sommes convaincus que l'intégration des Bond Graphs sera un atout à SysML qui doit s'imposer en tant que langage standard couvrant les phases d'analyse des besoins, d'analyse du système et sa conception. En effet SysML doit être la base d'un outil de conception qui regroupe les différents domaines technologiques concourrant à la conception d'un système mécatronique, et les Bond Graphs sont un parmi les outils destinés à effectuer l'analyse et la conception basées sur une approche systémique.

Notre proposition n'utilise que des mécanismes d'extension classiques et peut donc être intégrée dans tout outil supportant SysML et permettant la création de profiles ce qui est devenu une fonctionnalité de base pour tout outil de qualité et reconnu à l'échelle industrielle. Avec cette nouvelle proposition, nous avons un diagramme permettant de décrire des Bond Graphs et ayant une sémantique statique au même titre que les BG.

La composition des Bond Graphs avec les diagrammes de Blocks permet d'avoir des systèmes multi-physiques asservis et ainsi fournir les éléments nécessaires à utiliser une approche systémique.

L'une des utilités des Bond Graphs est la simulation du système. La prochaine étape est de générer le code Modelica[27] correspondant au système en ciblant une bibliothèque Modelica **[Broenink1999]**, en l'occurrence BondLib. Nous présenterons cette génération dans la partie « expérimentations ».

[27]. Modelica est un langage orienté objet pour la description textuelle de systèmes physiques et leurs simulation.

Chapitre 8

VIII - Identification de métriques pour les modèles préconisés

8.1 Introduction

Dans ce chapitre nous allons présenter un ensemble de métriques destinées à améliorer la conception. Les métriques que nous avons établies concernent le diagramme de blocks SysML muni des extensions présentées. En ingénierie logicielle les métriques des diagrammes UML ont été développées pour améliorer la qualité des modèles. Des techniques de transformation de modèles sont ensuite appliquées pour améliorer la configuration d'un modèle donné et proposer cette nouvelle configuration, la décision revient toujours au concepteur de garder ou non cette proposition.

8.2 L'utilité des métriques en conception orientée-objet

En UML, l'utilisation des métriques est une pratique en pleine évolution et la recherche des techniques de contrôle de la qualité des modèles est un domaine très actif. Dans **[Bernandez 2005]** les auteurs décrivent des métriques appliquées au diagramme des cas d'utilisations pour prédire des attributs tels que la modificabilité et pour détecter des défauts dans la conception. Ils utilisent pour cela le nombre de fois qu'un cas d'utilisation est inclus ou étendu par d'autres cas d'utilisations. **[Henderson2002]** présente des métriques pour l'estimation de la complexité de cas d'utilisations. Ils extraient de ces métriques des évaluations de maintenabilité des systèmes. Dans **[Bansiya2002]** les auteurs proposent des métriques qui calculent les degrees d'encapsulation, de couplage, de cohesion, de composition et d'héritage dans un diagramme de classes. Et dans **[Marchesi1998]** l'auteur définit une liste de métriques pour les diagrammes de classe qui sont appliqués au niveau du diagramme (associations, héritages, etc.) et au niveau de la définition de la classe elle-même (propriétés, méthodes, etc.).

8.3 Critères d'intégration d'un bloc

Nous supposons qu'un composant mécatronique correctement conçu est un composant mécatronique hautement intégré. Nous définissons alors des critères d'intégration d'un bloc pour pouvoir retrouver les points faibles de la conception. Par

exemple retrouver les composants mécatroniques qui présentent une faible intégration et qui doivent donc être reconçus. Nous fixons aussi une règle ; Le choix doit toujours revenir au concepteur, nous lui proposons un moyen de vérification de son architecture tôt dans le cycle de conception sans lui imposer une solution particulière. Les critères que nous définissons sont les suivants :

- Dans un bloc, un composant de ce bloc qui n'interagit pas avec les autres composants du bloc devrait quitter le bloc.
- Dans un bloc, un composant de ce bloc qui n'a qu'une interaction sortante vers les autres composants du bloc est éligible à quitter le bloc.
- Dans un bloc, un composant de ce bloc qui n'a que des interactions entrantes de composants du bloc mais ne fournissant rien aux autres composants du bloc est éligible à quitter le bloc.

8.4 Définitions de métriques pour améliorer la conception

8.4.1 Que faut-il améliorer?

Concevoir un système mécatronique peut être une tâche très complexe, c'est pour cette raison que nous devons respecter quelques patrons de conception en fin d'obtenir une bonne conception. Une bonne conception est une conception qui permet une bonne réutilisation et une bonne encapsulation. La loi de Constantine **[Stevens1974]** annonce : « Une structure est stable si la *cohésion* est forte et le *couplage* faible ». **[Endres2003]** qui analyse cette loi explique que la cohésion est la communication intra-modèle et que le couplage est l'ensemble des interactions inter-modèles. Il explique qu'une telle structure permet qu'un composant puisse être implémenté, intégré ou échangé en ayant un minimum d'effets sur le système. La stabilité du système veut dire qu'il est tolérant face aux problèmes qui peuvent arriver comme la modification de la conception. Dans le cas d'un système mécatronique nous définissons une bonne conception comme étant un graphe reliant des sous-systèmes hautement intégrés (à haute cohésion). (figure 8.1)

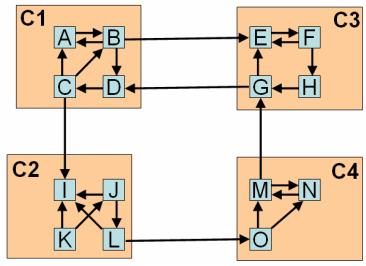


Figure 8.1. Système mécatronique présentant une architecture bien conçue.

Pour pouvoir obtenir cette architecture, nous devons retrouver les sous-systèmes hautement intégrés. Ensuite, nous définissons l'indicateur d'intégration mécatronique IIM, qui sera utilisé dans un diagramme des blocks internes SysML pour estimer le degré d'intégration (ou cohésion) d'un block ou d'un composant mécatronique.

8.4.2 Les métriques proposées

8.4.2.1 Le critère de centralisation CC

Nous définissons le critère de centralisation d'un composant mécatronique MCi comme étant le nombre de ses parties de contrôle (et de capture) divisé par le nombre de ses parties opératives (et d'actionnement) et ses composants mécatroniques.

(1)	$CC_i = \frac{n_i}{m_i + NbMC_i}$

n_i: nombre des parties opératives (et d'actionnement) du composant MC_i.

m_i: nombre des parties contrôle (et de capture) de MC_i.

$NbMC_i$: nombre des composants mécatroniques contenus dans MC_i.

Lorsque l'indicateur CC décroit, nous avons un composant mécatronique ou un block plus distribué. Lorsqu'il croit, nous avons un composant plus centralisé. Un concepteur peut préférer l'une ou l'autre des architectures ; S'il privilégie la sécurité il pourrait préférer la centralisation, s'il privilégie l'autonomie, la performance et la disponibilité il pourrait tendre vers la distribution. En utilisant cet indicateur il peut évaluer l'architecture de ses composants en fonction de ses priorités.

8.4.2.2 L’indicateur de couplage syntaxique

Quand pouvons-nous dire si un ensemble de composants (blocks, parts) est couplé ? Nous considérons qu’une réponse positive minimale à cette question est lorsqu’un composant :

1. dépend d’un autre composant et
2. est dépendant d’un autre composant

Nous définissons cette situation par le couplage syntaxique minimal CSM d’un ensemble de composants. CSM est égal à 2 pour un couple de composants.

Nous définissons l’indicateur de couplage syntaxique ICS d’un block par le nombre de connecteurs existants entre les parties contrôle et les parties opératives NCE, divisé par le nombre minimum de connecteurs possibles entre ces deux sous-ensembles NMCP.

NbCt : Nombre de parties contrôle.

NbOp : Nombre de parties opératives.

(2)	NMCP = NbCt X NbOp X CSM
(3)	$ICS = \frac{NCE}{NMCP}$

Considérons ce diagramme IBD d’un composant mécatronique (figure 8.2) :

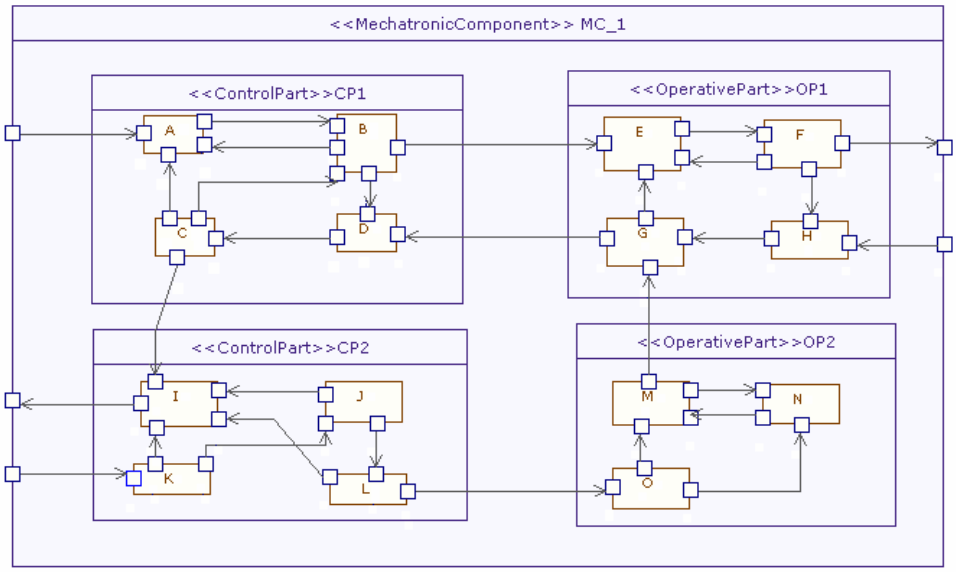


Figure 8.2. IBD exemple.

ICS = 3 / (10 x 9 x2) = 0.0166 (en supposant que A, B, C, D, I, J, K, L sont stéréotypées <<ControlPart>>, et que E, F, G, H, M, N, O sont des <<OperativePart>>).

Dans cet exemple nous obtenons un résultat très bas car le concepteur a utilisé les nested connectors[28] (connecteurs traversant en profondeur les frontières des composants), le calcul confond alors les composants qui ne sont pas dans un même niveau hiérarchique.

Ce résultat ne reflète pas la réalité du couplage fort existant dans la configuration de l'exemple, argument de plus pour appliquer les recommandations de Mickael Latta quant à l'utilisation des nested connectors (voir paragraphe 4.4). Nous devons ainsi réaliser des transformations de modèle sur ce diagramme pour obtenir une description respectant l'encapsulation en éliminant les nested connectors et en créant des ports définissant les frontières des composants. Nous obtenons cette configuration (figure 8.3) :

[28] Le nested connector est un connecteur utilisé avec les boites blanches et ne respectant pas le pattern de l'encapsulation [SysML page 43]

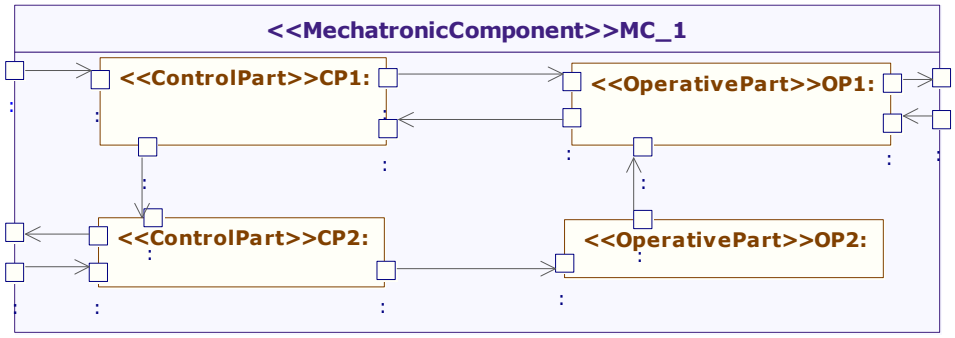


Figure 8.3. IBD transformé, sans nested connectors.

Le calcul de ICS donne maintenant :

ICS = 3 / (2 x 2 x 2) = 0.375

Ce résultat est satisfaisant puisque CP1 est couplée avec OP1 mais pas avec OP2, du moins directement. Il se peut que CP1 soit en réalité couplé avec OP2 dans une boucle passant par OP1 et/ou CP2 mais cette seule vue de l'architecture qu'est le IBD ne permet pas de voir cela, c'est pour cette raison que nous avons appelé cet indicateur l'indicateur de couplage *syntaxique*.

8.4.2.3 L'indicateur de couplage effectif

Nous avons besoin de calculer le couplage effectif du block (pas seulement syntaxique, architectural mais aussi fonctionnel). Pour cela nous devons savoir combien de boucles un block possède. Pour cela, nous utilisons la DSM du block. Le carré de la DSM nous donne les 2-cycles. Le cube de la DSM nous donne les 3-cycles, et ainsi de suite. Nous arrêtons l'élévation de puissance lorsque l'on a :

- $DSM^i = DSM^j$ avec i<j ou
- Lorsque tous les composants participent à au moins une boucle.
- Lorsque l'on atteint une limite maximum prédéfinie.

Nous obtenons pour chaque composant du block, le nombre de cycles syntaxiques auxquels il participe. Un cycle syntaxique est un cycle dans la description de l'architecture, un tel cycle ne représente pas forcément une boucle de contrôle.

Nous pouvons vérifier qu'un cycle syntaxique est aussi logique en utilisant les diagrammes d'états des composants et les diagrammes paramétriques de chaque état. Ainsi, pour obtenir les cycles logiques nous appliquons cet algorithme :

- Pour chaque cycle, nous parcourons ses composants, si les ports de ce connecteur n'interviennent pas dans l'une des équations paramétriques de son diagramme d'états, le cycle est interrompu. Nous concluons que le cycle est seulement syntaxique, sinon le cycle est aussi logique.

Pour calculer l'indicateur de couplage effectif ICE de l'exemple nous devons analyser le cycle CP1-CP2-OP2-OP1-CP1. Nous parcourons ses composants et pour chaque connecteur, si les ports de ce connecteur (variable associée) n'interviennent pas dans une des équations de son diagramme d'états nous pouvons affirmer que le cycle n'est pas un cycle logique, sinon, nous pouvons considérer qu'il l'est et dans ce cas, nous ajoutons deux connecteurs virtuels (utilisés pour le calcul) entre CP1 et OP2 et deux entre CP2 et OP1 puis nous recalculons ICS, nous obtenons :
ICE = 7 / 8 = 0,875

Ce nouveau résultat reflète le fort couplage entre le sous-ensemble des parties contrôles et celui des parties opératives de la figure.

Un couplage fort est un résultat satisfaisant quant à la conception de l'architecture. Si nous trouvons que le couplage est faible, nous pouvons proposer l'algorithme d'amélioration de la cohésion dans un block décrit dans le paragraphe suivant.

8.4.2.4 L'amélioration de la cohésion dans un block

L'amélioration de la cohésion d'un block (que nous appellerons ACB) est une transformation de modèle qui permet d'améliorer le partitionnement d'un block en augmentant la cohésion de ses composants. Cet algorithme est appliqué après avoir appliqué l'algorithme qui élimine les nested connectors. Il s'agit de restructurer un

block de manière à obtenir un block composé de composants hautement intégrés. Soit l'exemple de la figure suivante (figure 8.4) :

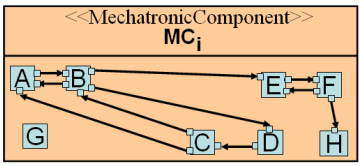


Figure 8.4. Block ayant une faible cohésion.

Dans ce cas nous pouvons améliorer le partitionnement des composants du block en quatre composants hautement intégrés. Nous utilisons pour cela la DSM pour obtenir ce partitionnement. Dans cet exemple, la DSM de départ est (figure 8.5):

	D	E	A	C	G	F	H	B
D	0	0	0	1	0	0	0	0
E	0	0	0	0	0	1	0	0
A	0	0	0	0	0	0	0	1
C	0	0	1	0	0	0	0	1
G	0	0	0	0	0	0	0	0
F	0	1	0	0	0	0	1	0
H	0	0	0	0	0	0	0	0
B	1	1	1	0	0	0	0	0

Figure 8.5. DSM de départ.

En appliquant l'algorithme de partitionnement utilisant la recherche de chemins décrit dans **[DSM_Home]** nous obtenons la matrice triangulaire inférieure suivante (figure 8.6):

	G	H	EF	AB CD
G	0	0	0	0
H	0	0	0	0
EF	0	0	1	0
ABCD	0	1	1	1

Figure 8.6. Matrice partitionnée.

Dans cette matrice triangulaire en blocks, nous obtenons deux ensembles hautement intégrés {A,B,C,D} et {E,F}. Nous pouvons ainsi partitionner le block en quatre composants mécatroniques. Ensuite nous les intégrons dans un même composant mécatronique de haut niveau pour ne pas perdre l'information décrite par le concepteur qui consistait à regrouper tous les composants dans le même block. Ensuite pour chaque block, en partant du block de plus haut niveau et récursivement, si tous

ses composants sont opératifs, nous le transformons en <<OperativePart>> (et même chose pour <<ControlPart>>), s'il contient un seul composant, nous le réduisons à ce composant qu'il contient. Nous obtenons le résultat suivant (figure 8.7):

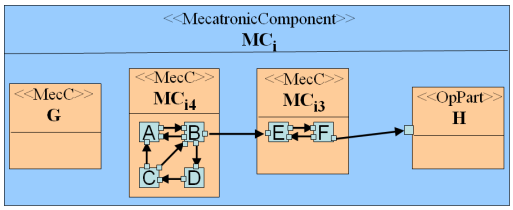


Figure 8.7. Block amélioré par l'application de l'algorithme ACB.

8.4.2.6 L'indicateur d'intégration mécatronique

Nous définissons l'indicateur d'intégration mécatronique IIM d'un block par le nombre de ses connecteurs strictement internes divisé par le nombre de ses composants. Cette métrique nous donne une évaluation de l'intégration des composants du block. Pour l'exemple de la figure 8.3 :

IIM = 5/4 = 1.2

Le résultat montre que le block de la figure 8.3 est hautement intégré. Si nous obtenons un résultat proche de zéro, nous pouvons conclure que le block est très peu intégré et peut être dispatché. Si le résultat est très supérieur à 1, nous pouvons conclure qu'il y a un mauvais partitionnement et qu'une re-conception pourrait aider à réduire le nombre de communications entre les composants du block.

8.4.2.7 L'indicateur d'autonomie d'un block

Nous définissons d'abord les connecteurs strictement internes par les connecteurs ne faisant pas intervenir de ports à la frontière du block. Ensuite, nous définissons l'indicateur d'autonomie d'un block IAB par le nombre de connecteurs strictement internes divisé par le nombre de tous les connecteurs qu'il contient. Cet indicateur ne s'applique que lorsque nous considérons la condition d'interdiction des nested connectors. Ceci veut dire que nous nous intéressons au rapport entre les échanges internes et les échanges avec l'extérieur. Soit alors :

NCSI : Nombre de connecteurs strictement internes.

NCB : Nombre de connecteurs dans le block.

(4)	$AIB = \frac{NCSI}{NCB}$

Cette métrique nous donne une évaluation de l'autonomie d'un block donné, ainsi pour l'exemple de la figure 8.3 :

AIB = 5 / 10 = 0.5

Ce résultat montre que le block de la figure 8.3 n'est pas très autonome. Plus le résultat est proche de 1, plus on peut dire que le block est autonome. L'autonomie peut être un objectif de qualité pour un modèle d'un composant mécatronique, ce qui améliore en plus la modularité et la réutilisation.

8.4.2.8 L'indicateur d'interactivité

L'indicateur d'interactivité *Interact* d'un block est une métrique qui permet de mesurer le degré d'interactivité d'un block, il est calculé par la somme de ses entrées et sorties.

Soir s_i le nombre des entrées d'un block, v_i le nombre de ses sorties et p_i le nombre de ses autres ports, alors :

(5)	$Interact_i = s_i + v_i + p_i$

Cette information peut être utilisée dans le cas où l'on voudrait apporter des modifications à un modèle. Elle nous permet d'évaluer les effets de ce changement.

8.5 Conclusions

Dans ce chapitre, nous avons introduit quelques métriques pour les diagrammes SysML étendus pour identifier des mauvaises décisions de conception potentielles. Nous avons aussi présenté les transformations de modèles qui peuvent être appliqués pour améliorer le modèle. Pour chaque métrique nous avons présenté son objectif et sa formule. Pour les critères du modèle, nous avons décrit un algorithme permettant d'améliorer le modèle. Nous avons aussi illustré chaque indicateur à travers un exemple utilisant un IBD pour montrer l'impact de ces outils sur la qualité des modèles SysML. Ces métriques ont été développées pour fournir un point de départ

pour la transformation de modèles en vue d'améliorer la qualité des modèles en mécatronique.

Ces métriques s'insèrent dans la phase de vérification de l'architecture du processus proposé pour MISSyM dans le chapitre VI.

Chapitre 9

IX - Expérimentations

9.1 Exemple de mise en ouvre de la méthodologie MISSyM

Nous illustrons la mise en œuvre de la méthodologie **MISSyM** proposée (figure 9.1) par l'application de l'exemple académique d'un ascenseur intégrant la motorisation, assimilé à un système mécatronique.

Nous rappelons ici le cycle de conception proposé pour la **MISSyM :**

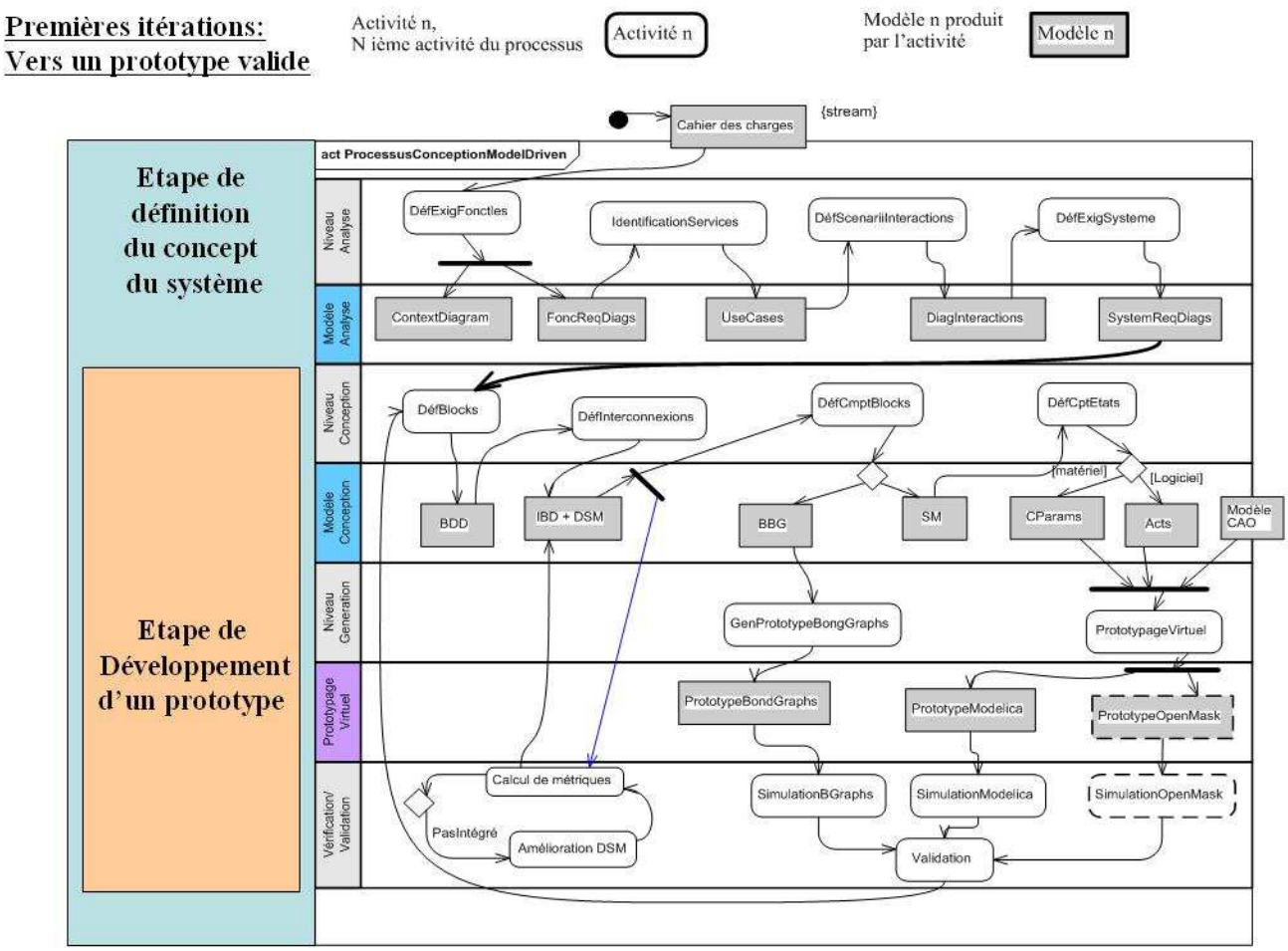


Figure 9.1. (Rappel) L'étape de définition du concept du système appliquée à SysML.

9.1.1 Le processus de définition des besoins des parties prenantes :

La première tâche est de structurer les besoins textuels du cahier des charges pour décrire les besoins des parties prenantes en définissant les exigences sur les fonctions du système.

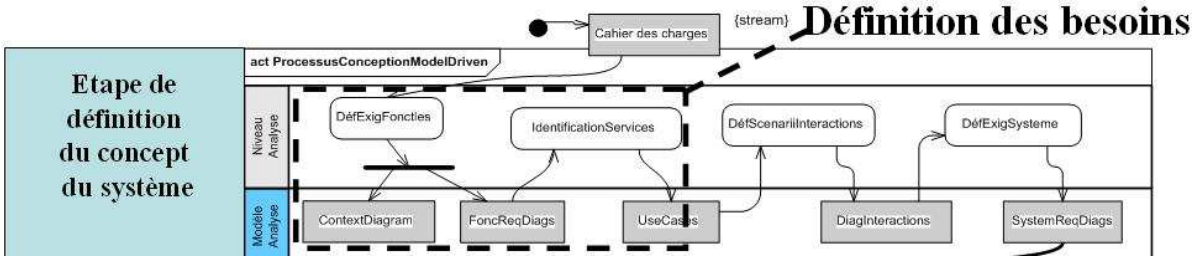


Figure 9.2. (Rappel) Le processus de définition des besoins.

Ce qui correspond (figure 9.2) en termes de modèles à dresser, en premier lieu, les diagrammes d'exigences sur les fonctions du système (figure 9.3).

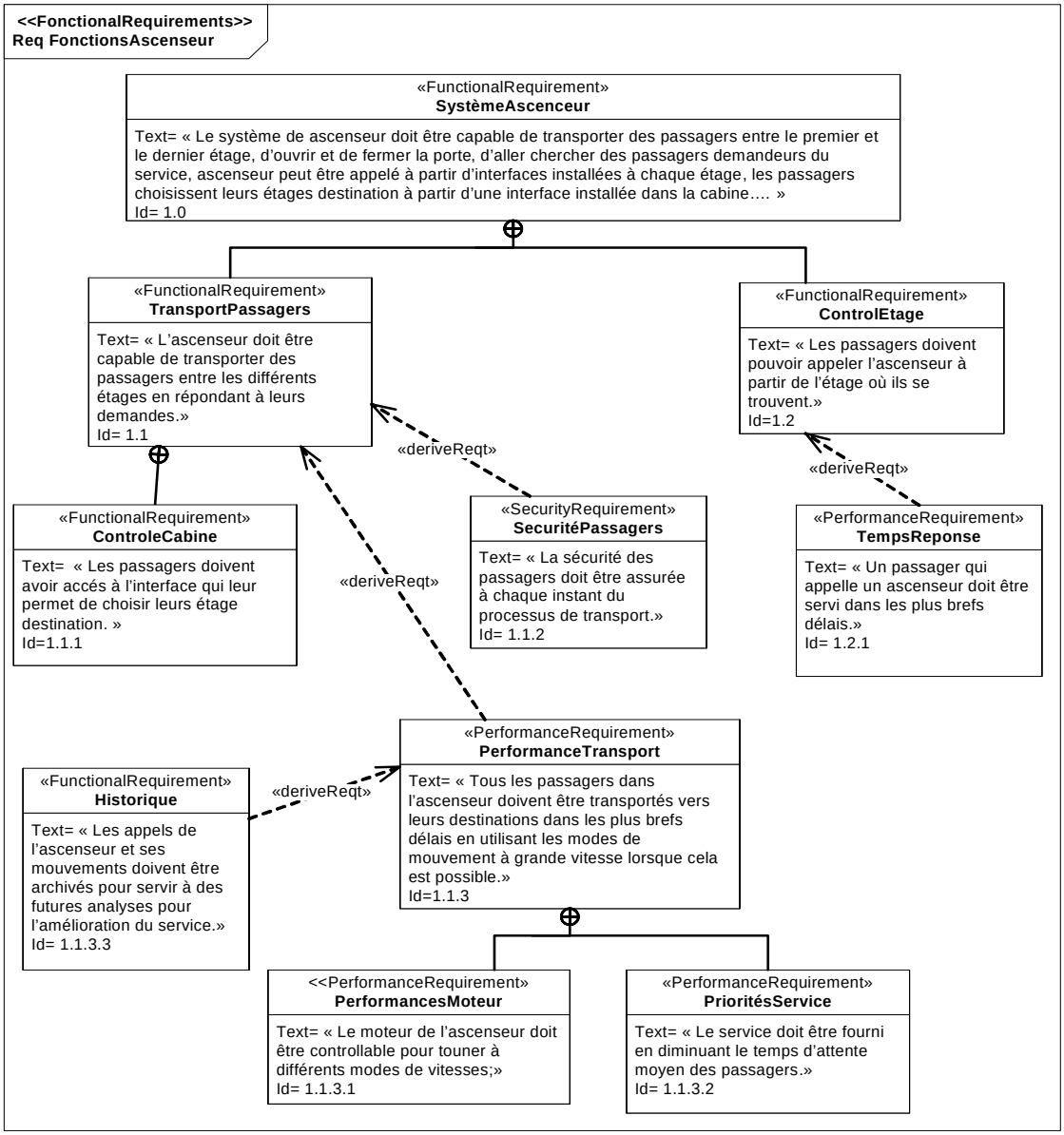


Figure 9.3. Diagramme des exigences utilisé en tant que support pour la définition des besoins des parties prenantes.

Les éléments de modélisation de ce diagramme permettent de décomposer des besoins en d'autres plus élémentaires, de décrire des inférences (le stéréotype <<deriveReqt>>) de nouvelles exigences, de documenter les raisonnements desquels le concepteur juge utile de garder une trace.

La deuxième étape est de décrire les services que le système doit fournir dans des diagrammes de cas d'utilisation (figure 9.4). Il s'agit des besoins fonctionnels.

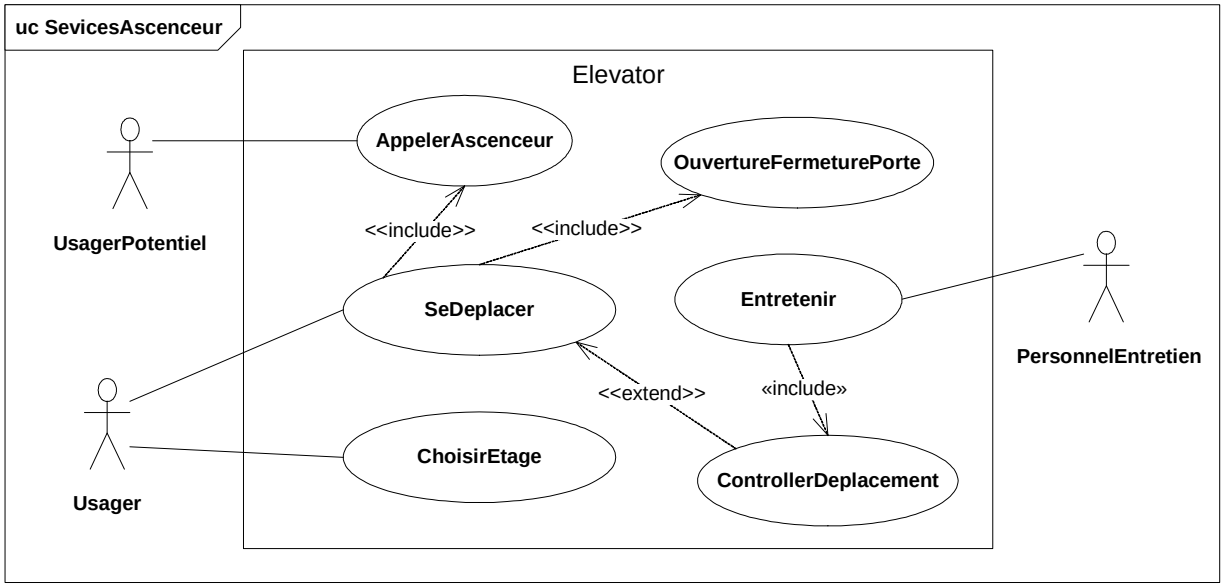


Figure 9.4. Diagramme des cas d'utilisation utilisé pour la définition des besoins des parties prenantes en terme de services/Fonctions.

9.1.2 Le processus d'analyse des exigences

Le deuxième processus (figure 9.5) est celui d'analyse des exigences qui permet de transformer des exigences sur des fonctions en exigences sur le système.

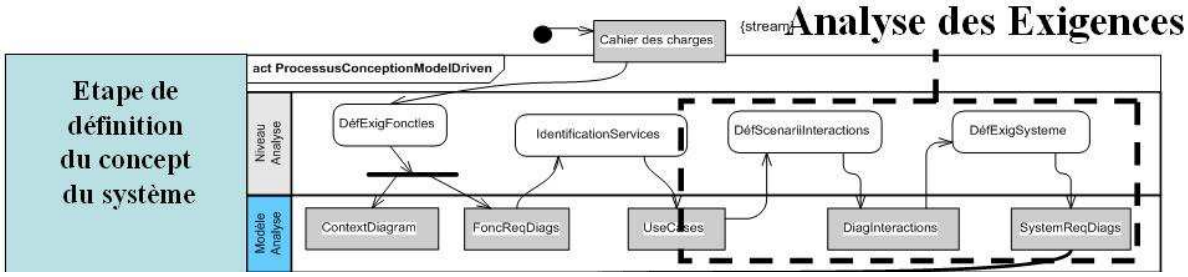


Figure 9.5. Le processus d'analyse des exigences appliqué à SysML étendu.

Au cours de ce processus, chaque cas d'utilisation est décrit par un diagramme des séquences (figure 9.6), ce qui permet d'identifier des sous-systèmes, les interactions qui les lient et les exigences inférées sur ces sous-systèmes.

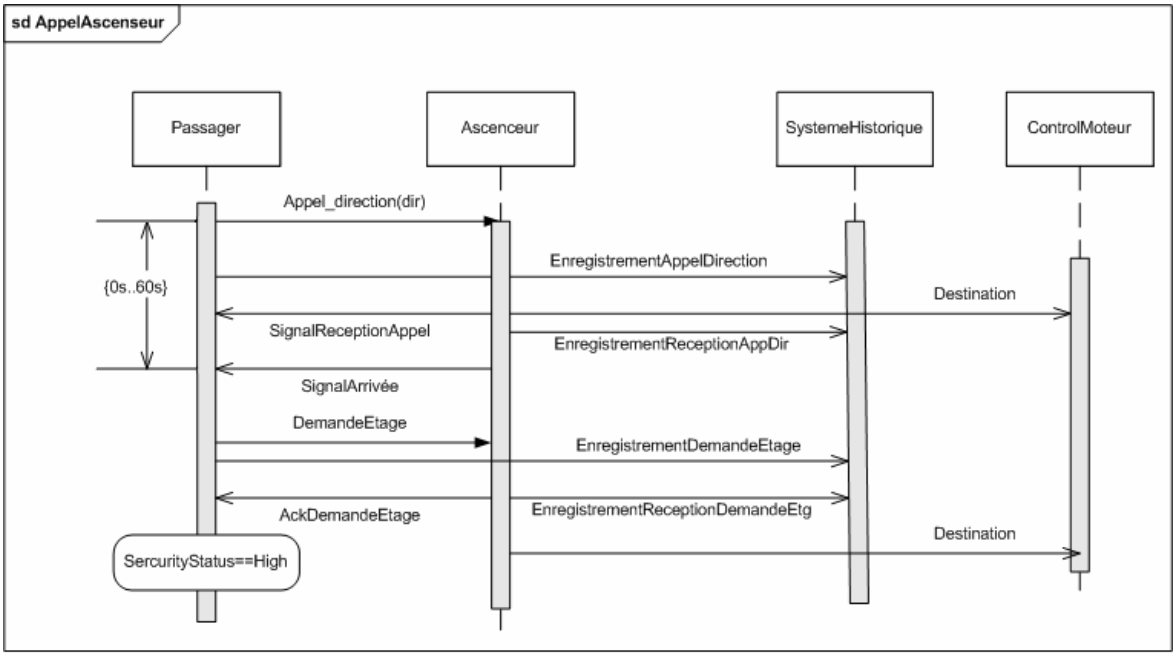


Figure 9.6. Cas d'utilisation AppelerAscenceur analysé
dans un diagramme des séquences.

Ce diagramme nous permet de passer d'un modèle basé sur les fonctions à un modèle basé sur les composants du système. Ce qui permet de dresser alors des diagrammes d'exigences sur ces composants/sous-systèmes. Dans ce diagramme des exigences (figure 9.7) nous montrons la mise en œuvre des extensions proposées dans le paragraphe 6.3.2.

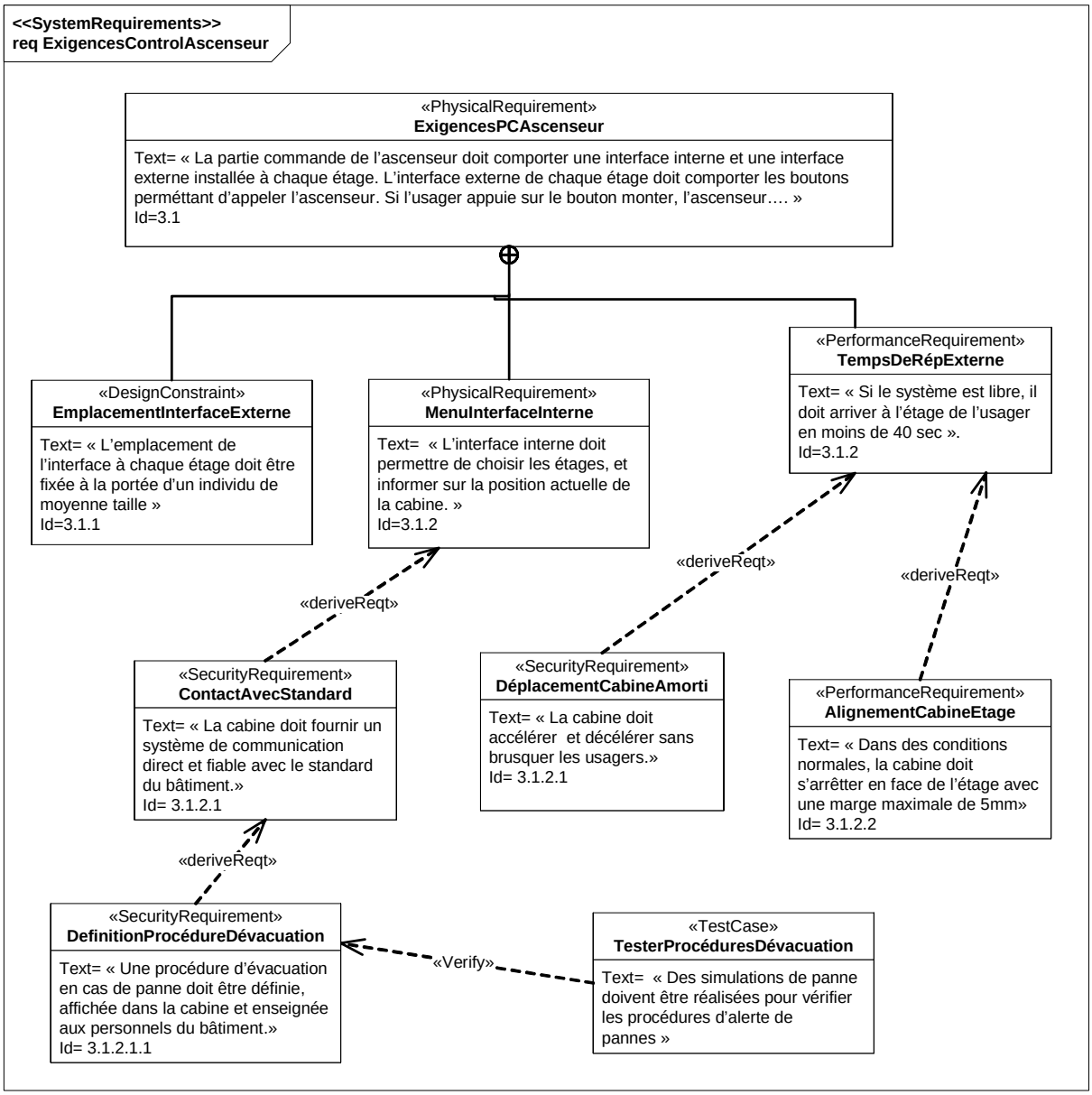


Figure 9.7. Diagramme des exigences utilisé en tant que support pour l'analyse des exigences et la description des exigences sur le système.

9.1.3 Le processus de conception de l'architecture

Nous pouvons maintenant commencer le processus de conception de l'architecture (figure 9.8) qui se traduit dans notre méthodologie par la décomposition logique de notre système (figure 9.9).

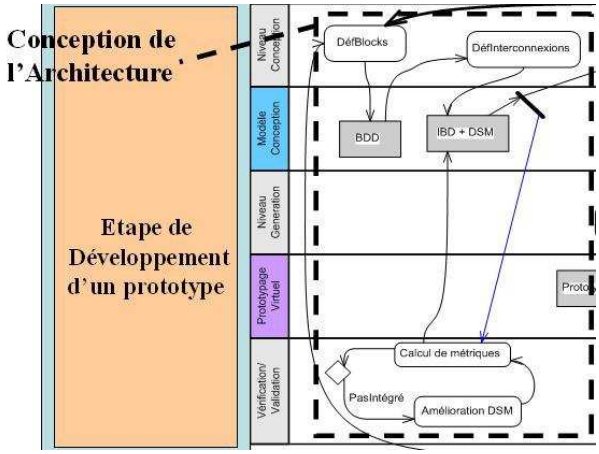


Figure 9.8. (Rappel) Le processus de conception de l'architecture.

Nous décrivons les diagrammes de définition de blocks en utilisant l'élément <<composant mécatronique>> et les extensions proposées au paragraphe 6.4.6.

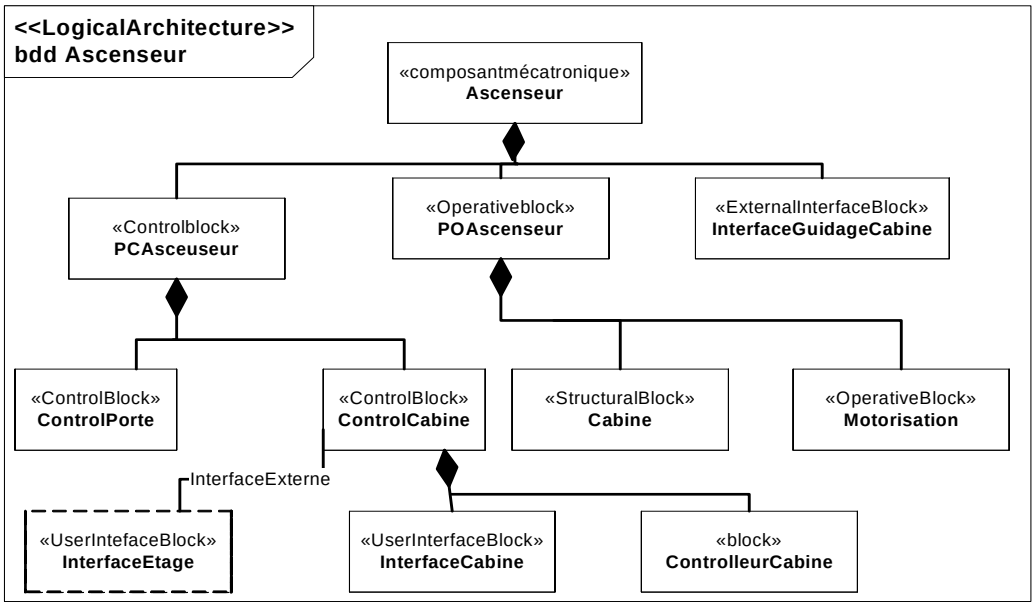


Figure 9.9. Diagramme de définition de blocks
utilisé pour la description de l'architecture.

Nous détaillons la description de l'ascenseur (figure 9.10) qui est un élément <<composant mécatronique>> en mettant en œuvre les extensions proposées pour les IBD (6.4.6), les connecteurs (6.4.1) et les ports (6.4.2).

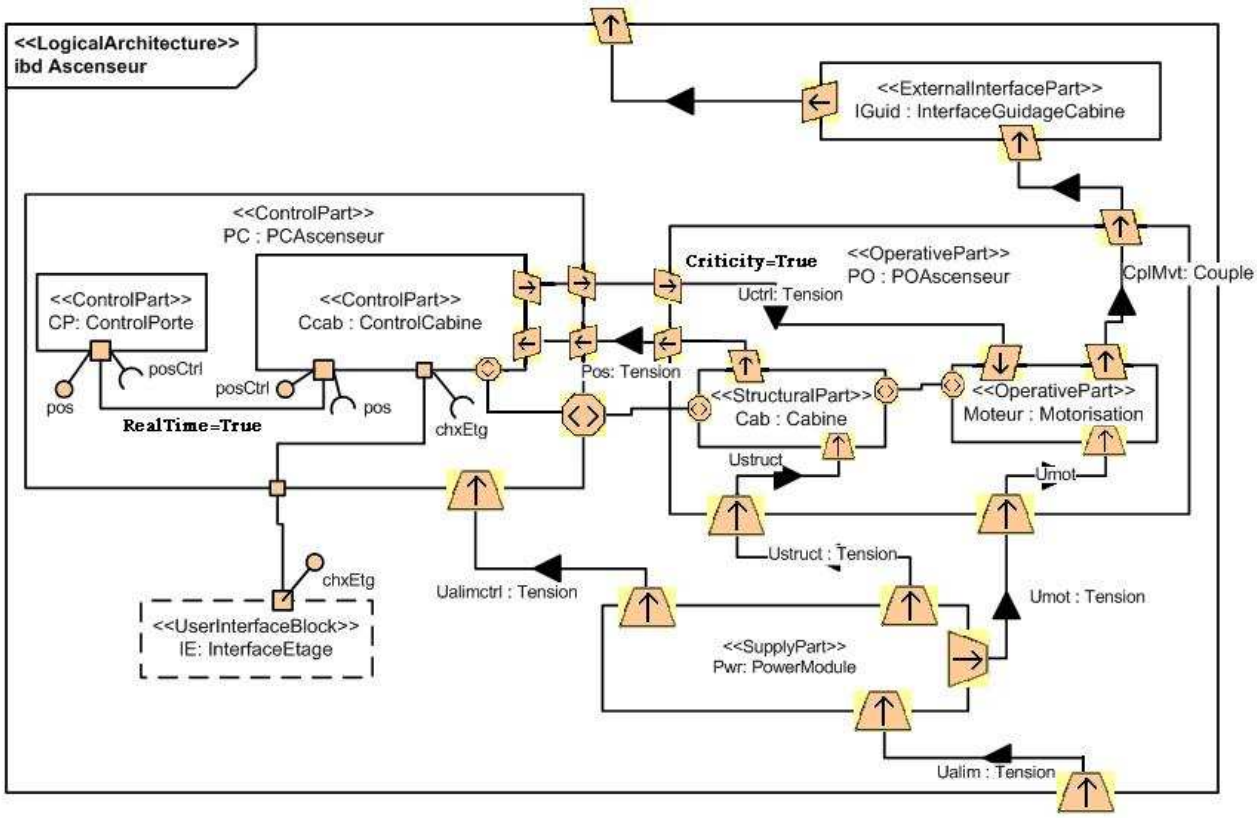


Figure 9.10. IBD utilisé pour la description de l'architecture d'un block.

Nous établissons par la suite la DSM (figure 9.11) du composant mécatronique Ascenseur en utilisant le codage proposé au paragraphe 6.4.3.2 :

DSM_Ascenseur	CP	CCab	Cab	mot	Pwr	IGuid
CP	X	8				
CCab	8	X	2+5		3	
Cab		2	X	2	3	
mot		10	2	X	3	
Pwr					X	
IGuid				5		X

Figure 9.11. DSM utilisée pour résumer matriciellement l'architecture du block.

L'application du partitionnement sur cette DSM en utilisant le logiciel PSM32 **[PSM32]** donne le partitionnement suivant (figure 9.12):

ASCENSEUR

03/06/2008 17:50:33 Created By: NoUser

	1	2	3	4	5	6
5! Pwr						
1! CP			8			
2! CCab	3	8		7		
3! Cab	3		2		2	
4! Mot	3		10	2		
6! IGuid					5	

Printed: 6/3/2008 17:51:27 Psm32: Problem Solving Matrix 3.9j -Join- (c) 1996-2003 Problematics/Blitzkrieg Software

Figure 9.12. Application du partitionnement de la DSM sur l'exemple.

Ce partitionnement donne un composant hautement intégré (le carré en couleurs {CP, CCab, Cab, Mot}). Les deux composants Pwr et IGuid peuvent être sortis du composant pour avoir un composant hautement intégré avec un minimum de couplage.

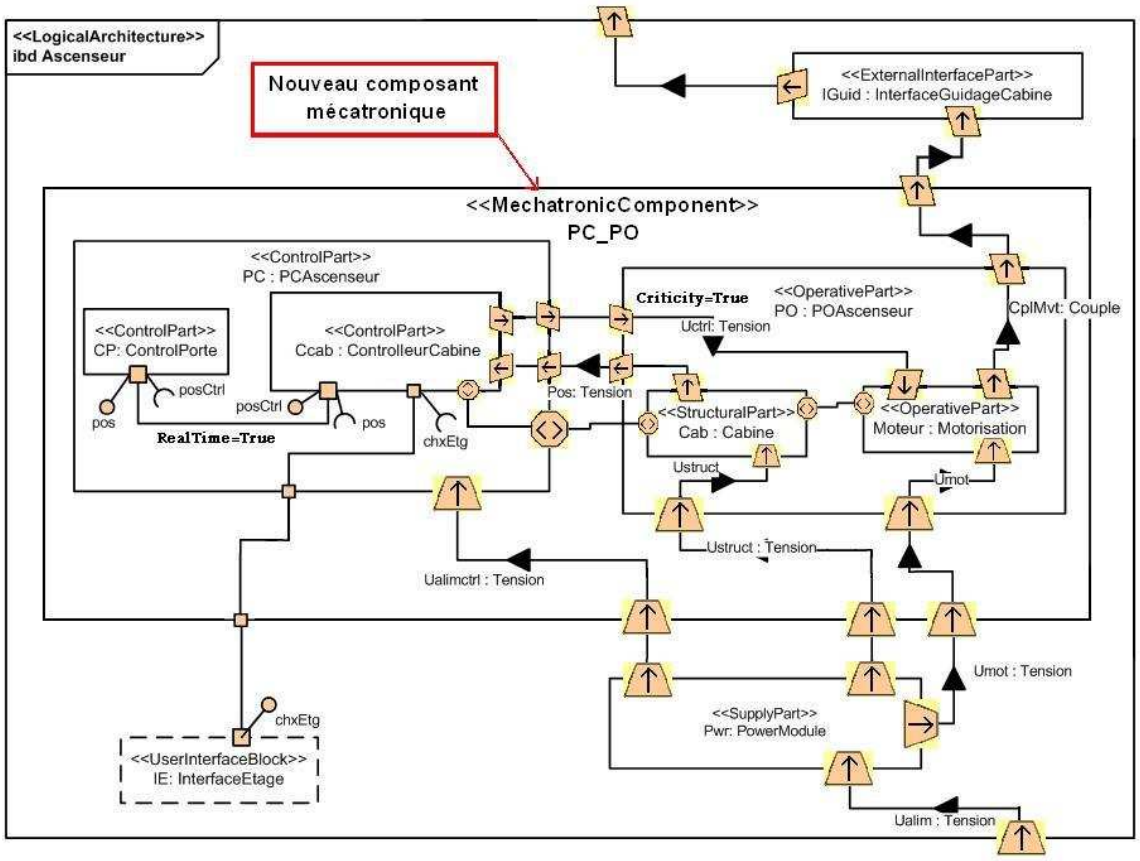


Figure 9.13. IBD de la figure 9.10 réorganisé
après application du nouveau partitionnement.

L’application de l’algorithme d’amélioration de la cohésion (paragraphe 8.4.2.4) donne le résultat de la figure 9.13. L’ensemble hautement intégré {CP, CCab, Cab, Mot} est inséré dans un nouveau composant mécatronique. Les autres composants ayant une interaction faible avec cet ensemble sont gardés en dehors du nouveau composant mécatronique. Des ports sont ajoutés aux frontières du CM pour respecter l’interdiction des nested connectors.

9.1.4 Le processus de réalisation et d’intégration

Ensuite, nous passons au processus de réalisation d’un prototype (figure 9.14). Au cours de ce processus nous décrivons le comportement des blocks.

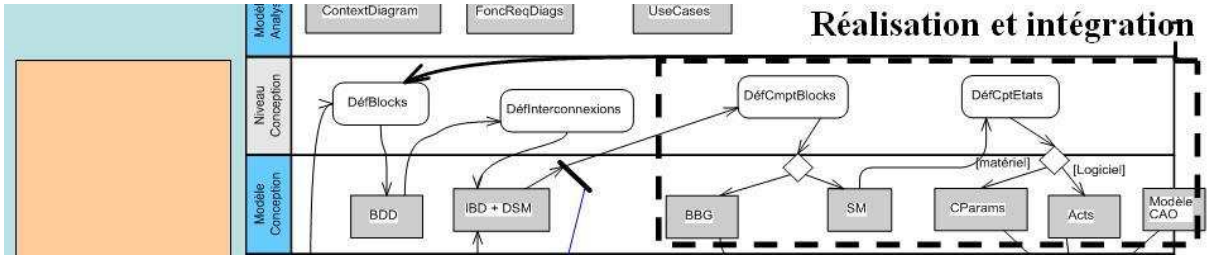


Figure 9.14. (Rappel) Le processus de réalisation
et d’intégration appliqué à SysML étendu.

Nous utilisons pour cela les allocations pour allouer du comportement aux différents composants/sous-systèmes. Nous pouvons ainsi allouer un diagramme d’états à un block (figure 9.15).

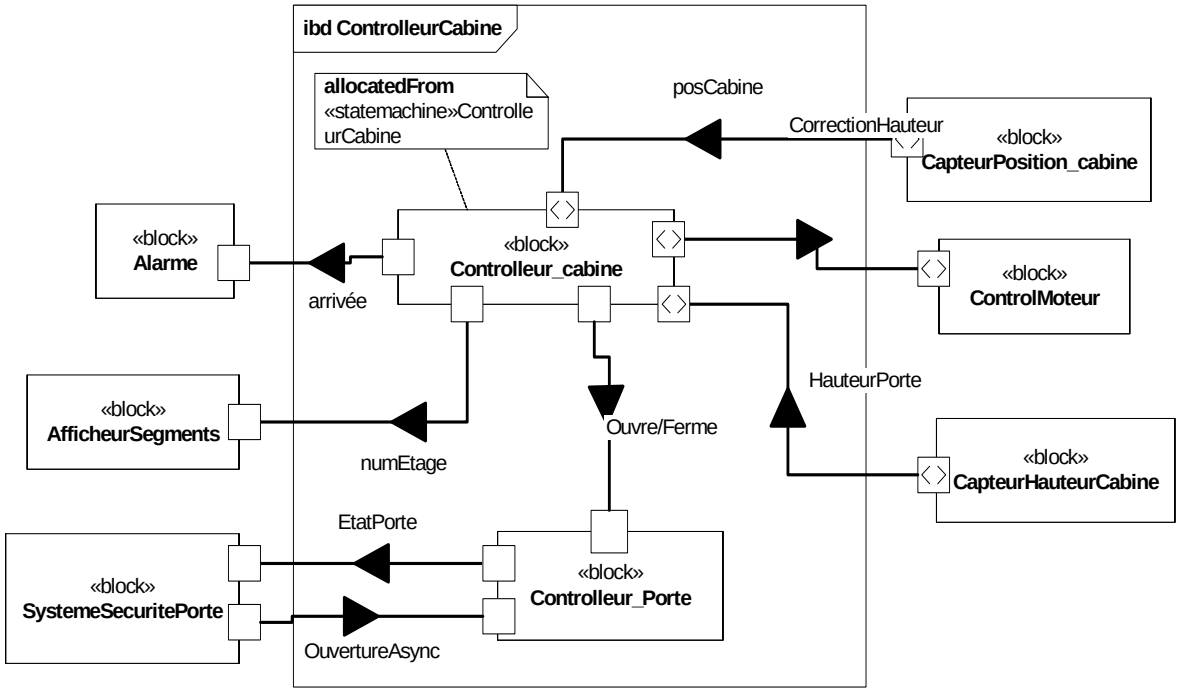


Figure 9.15. Allocation de machines d'états pour passer à la description du comportement.

Un diagramme d'états (figure 9.16) peut ainsi être associé aux composants pour décrire les comportements des composants dits réactifs ; qui réagissent à des événements. Ce comportement peut être hybride par l'association d'un comportement continu à un état particulier.

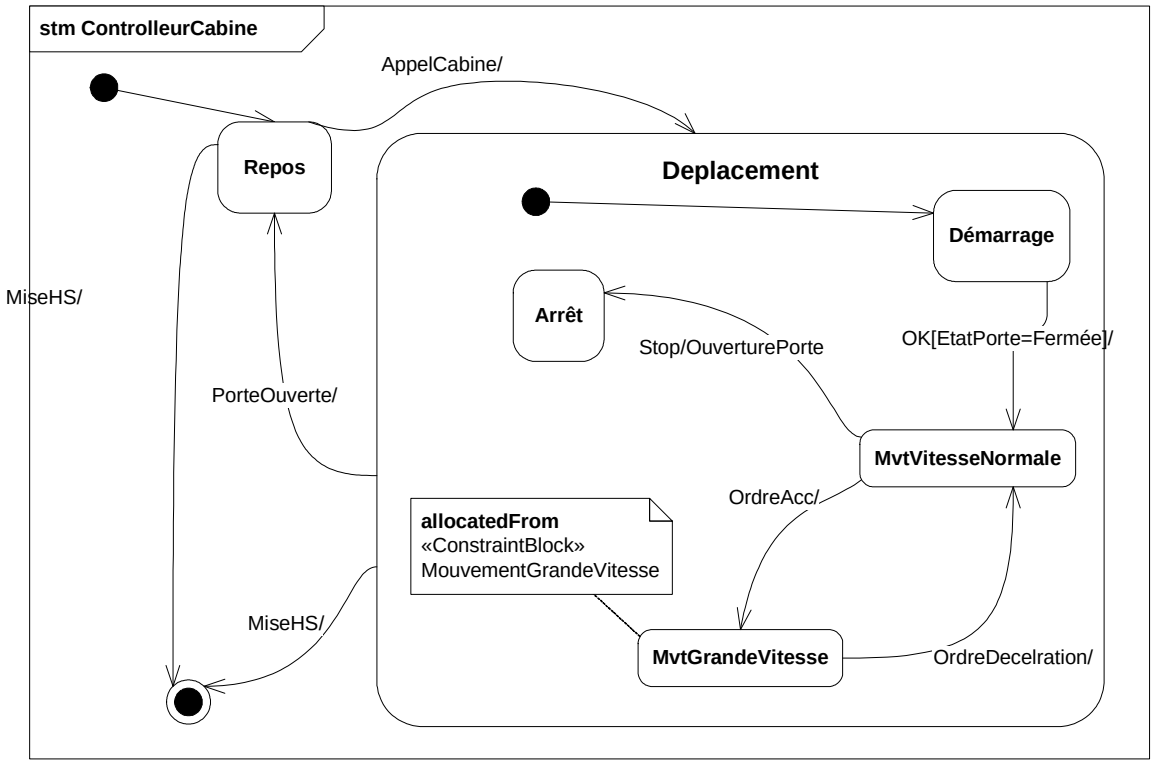


Figure 9.16. Diagramme d'états décrivant un composant à caractère réactif hybride.

Le comportement continu d'un état particulier peut être décrit grâce au diagramme des contraintes paramétriques (figure 9.17).

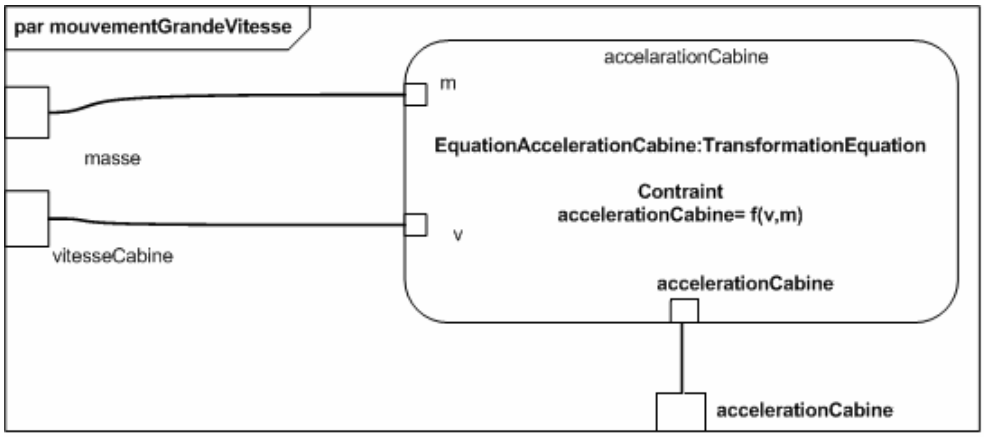


Figure 9.17. Comportement continu d'un état particulier.

9.2 Implémentation de la méthodologie MISSyM

9.2.1 Choix de l'outil

Il existe aujourd'hui plusieurs outils permettant d'implémenter une approche MDA. Bien que tous ces outils soient nouveaux et pas encore assez mûrs, les outils les plus avancés sont :

- Les DSL Tools de Microsoft **[DSL_Tools]** (Visual Studio Extensibility)
- La plateforme eclipse EMF/GEF/GMF **[Eclipse Project]**
- La suite MDA Modeler – UML Modeler d'Objecteering

9.2.1.1 Les DSL tools

Nous reprenons ici la définition donnée par Deursen, Klint et Visser dans **[Deursen2000]** : « Un DSL est un langage de programmation ou un langage de spécification exécutable, qui offre avec des notations et des abstractions appropriées l'expressivité nécessaire et souvent restreinte à, un domaine de problèmes particulier. »

Deurse, Klint et Visser, ajoutent que la puissance de ces langages est qu'ils sont totalement dédiés.

Nous avons implémenté un éditeur pour Bond Graphs sur la plateforme DSL Tools. La figure 9.18 montre la structuration de cette implémentation :

- Nous commençons par définir le modèle dans le couloir vertical gauche « Classes ans Relationships ». Nous décrivons alors les méta-classes avec leurs propriétés (exemple des méta-classes C, I, Jonction_0 munis, par héritage, d'un nom, d'une valeur et d'une unité) et les liens qui peuvent exister entre ces méta-classes (exemple du lien *NodeElementReferencesTargets* voir figure 9.18).
- puis nous définissons les éléments graphiques dans le couloir vertical droit « Diagram elements » en leurs associant les icônes lorsque nécessaire.

- Finalement, nous associons ces éléments graphiques aux méta-classes qu'elles représentent et nous générons automatiquement l'éditeur graphique Bond Graphs à partir de cette description graphique.

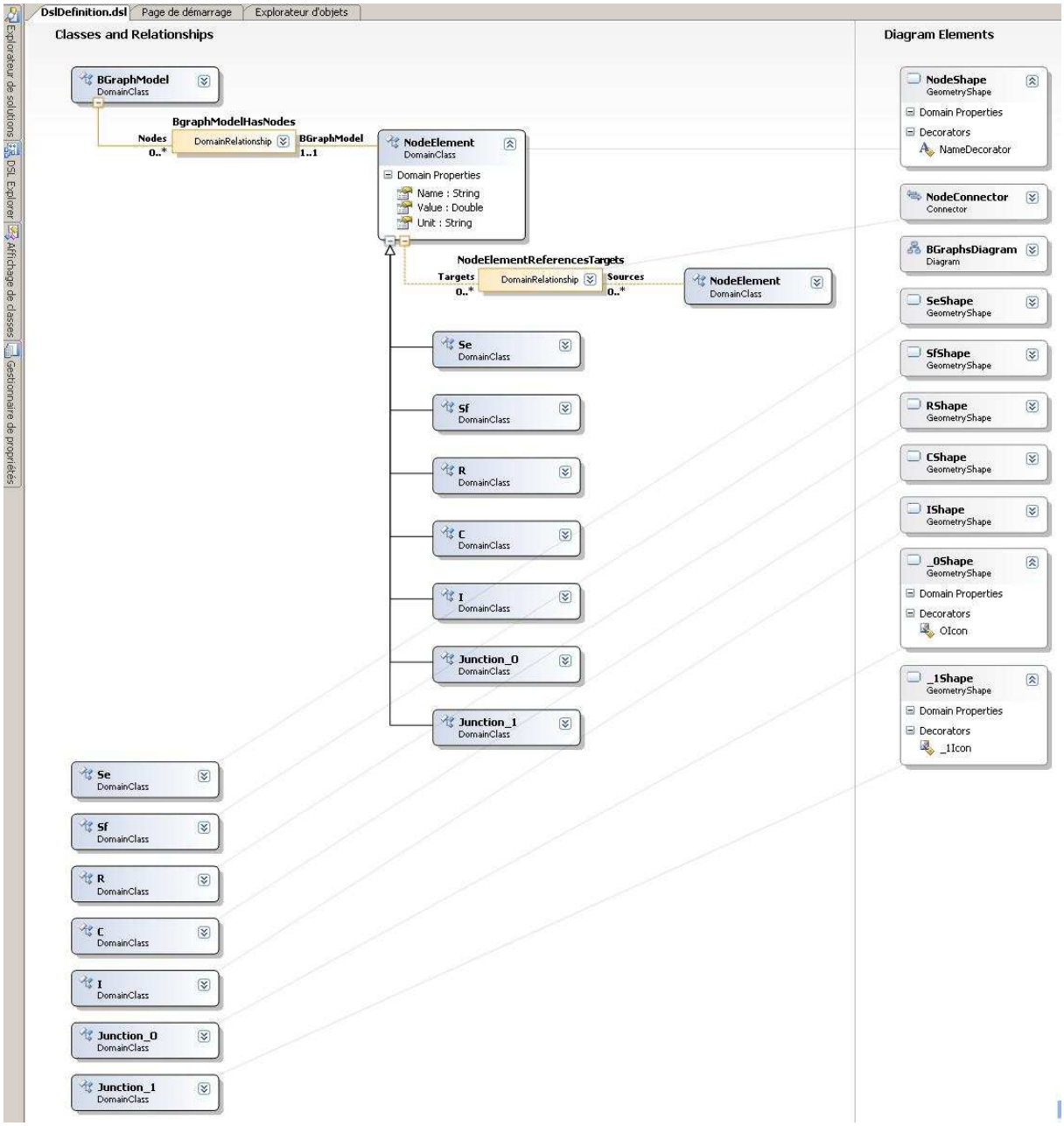


Figure 9.18. Description du langage Bond Graphs avec l'outil DSL Tools.

Cette plateforme s'est avérée très efficace et productive, La solution gère facilement la création d'un modèle associé à un diagramme. La description se fait aisément en utilisant un méta-méta-modèle (M3) assez simplifié. Par contre, ce méta-

méta-modèle est propre à l'outil donc toute possibilité d'échange du modèle doit passer par le fichier XMI généré puis par la transformation vers un autre format XMI utilisant le méta-méta-modèle de l'outil cible, ce qui rend l'échange trop compliqué. De plus, si notre méta-modèle devient très grand, l'utilisation de l'éditeur devient un peu délicate. La structuration du méta-modèle n'est pas bien supportée (pas de packages). L'outil est très bien adapté pour les petits langages (qui est la définition d'un DSL), mais pour des langages plus complexes avec plusieurs diagrammes, cet outil n'en a pas la vocation.

9.2.1.2 Eclipse EMF/GEF/GMF

Eclipse est une plate-forme modulable pour le développement d'applications dans un environnement JAVA. Plusieurs modules (bibliothèques) sont développés pour ECLIPSE et destinés au développement d'environnements ou d'ateliers de conception :

- EMF : bibliothèque permettant de créer un méta-modèle à partir du méta-méta-modèle Ecore qui est un sous-ensemble du MOF.
- GEF : bibliothèque permettant de modéliser l'outil graphique.
- OMELET est une bibliothèque pour l'intégration de modèles, la transformation de modèles et la représentation de modèles.
- GMT : (générative model transformer) est un ensemble d'outils pour le développement conduit pas les modèles d'éditeurs. Il se base sur EMF et GEF pour générer automatiquement l'éditeur graphique.

La plateforme Eclipse EMF/GEF/GMF est la plateforme de référence pour implémenter une approche basée sur MDA. Par contre nous cherchions une plateforme permettant le prototypage rapide d'éditeurs c'est-à-dire un outil permettant de créer un premier prototype de la méthodologie proposée en permettant de la tester et de, continuellement, intervenir sur le méta-modèle en intégrant de nouveaux stéréotypes et autres, et de prendre en compte directement ces modifications. L'outil idéal pour une telle approche est la suite Objecteering MDA Modeler que nous présentons dans ce qui suit.

9.2.2 Présentation d'Objecteering MDA Modeler

Nous avons choisi d'utiliser cet outil car il permet de bien structurer l'ensemble des éléments que nous avons décrit dans la définition de MISSyM (Sous-profiles, design-patterns, séparation des traitements tels que les métriques et la génération de code).

La suite d'outils MDA Modeler (figure 9.19) – UML Modeler d'Objecteering offre une plateforme de développement MDA complète. Le MDA Modeler permet de décrire des extensions appelées des composants MDA. Un composant MDA peut contenir :

- Un ou plusieurs profiles UML, chaque profile peut contenir des stéréotypes, des méta-propriétés (tagged-values), des notes ou des contraintes.
- Des design patterns.
- Des fonctions contenant du traitement, des opérations de lecture/écriture sur le modèle ou sur des fichiers. Ces fonctions peuvent être appelées à partir d'UML Modeler lorsque le composant MDA y est chargé.
- Des générateurs de code ou de documentation.

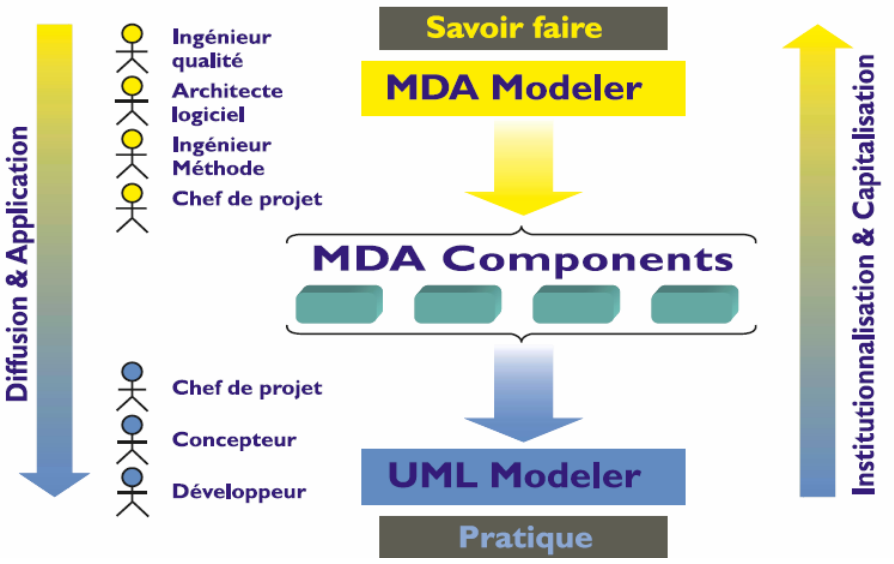


Figure 9.19. Architecture des outils MDA Modeler et UML Modeler d'Objecteering.

9.2.3 Implémentation des profiles proposés

Nous avons implémenté les profiles de la méthodologie MISSyM. Nous avons organisés le profile en sous-profiles pour permettre leur utilisation disjointe. Les sous-profiles sont (figure 9.20):

- SysML_IBD : Continent le sous-ensemble de SysML que nous avons étendu avec les profiles BlockBondGraphs et BlockDiagram, il contient aussi le composant mécatronique.
- BlockBondGraphs : Contient les extensions pour Bond Graphs basés sur le block SysML.
- BlockDiagram : Contient les extensions pour le diagramme en blocks basés sur le block SysML.
- SysMLAnalysis : Contient le sous-ensemble de SysML (extensions des use cases et des exigences) que nous avons étendu avec le package SysML15288.
- SysML15288 : Contient les extensions liées à la recommandation IEEE15288.

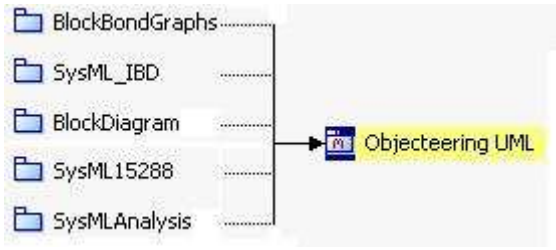


Figure 9.20. Les sous-profiles du modèle proposé.

L'implémentation du profile BlockBondGraphs et du profile BlockDiagram a permis de décrire le diagramme de la figure 9.21. L'exploitation des possibilités de personnalisation de la représentation graphique des stéréotypes a permis d'obtenir un vrai diagramme Bong Graph. La nécessité de répondre à cette exigence sur l'implémentation est importante car l'outil doit être utilisé par des concepteurs habitués à utiliser un éditeur Bond Graph.

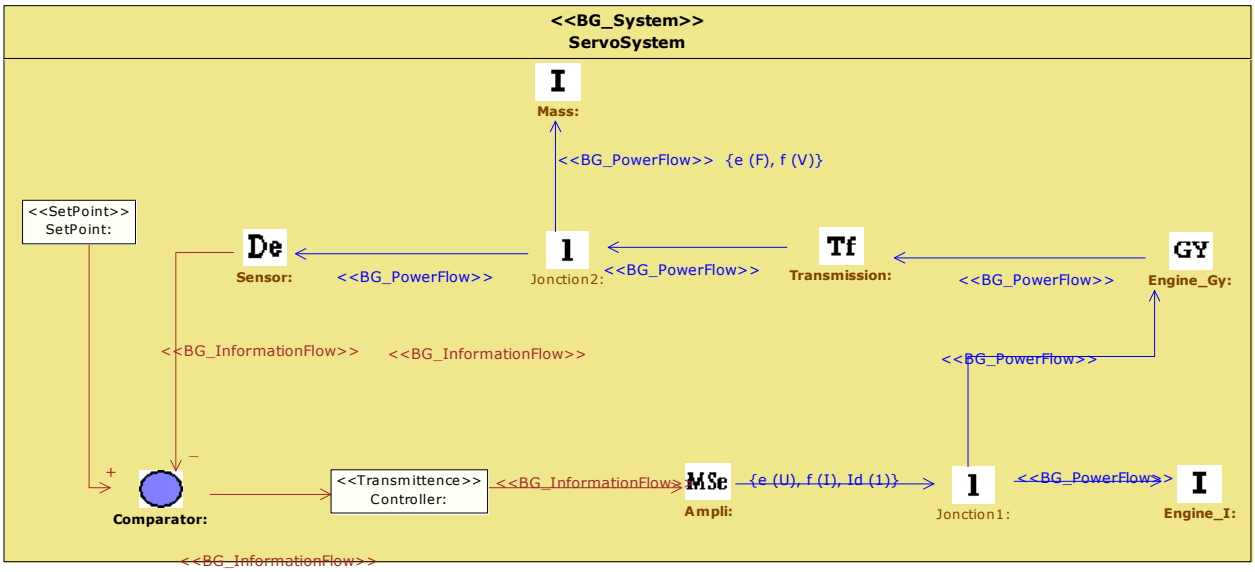


Figure 9.21. Bond Graph combiné au diagramme en blocks sous Objecteering.

Certaines mises en page ont été réalisées manuellement comme la sélection de l'affichage iconique pour les stéréotypes au lieu de l'affichage textuel (« stereotype »).

Ensuite la génération du code Modelica correspondant à ce diagramme donne le code suivant :

model BG_System	Début du modèle nommé BG_System
BondLib.Sources.mSe mSe; BondLib.Junctions.J1p3 j1p3_1; BondLib.Passive.I i(I=3); BondLib.Passive.GY gY(r=2); BondLib.Passive.TF tF(m=3); BondLib.Junctions.J1p3 j1p3_2; BondLib.Passive.I i1; BondLib.Sensors.De de; BondLib.Sensors.Pbond pbond; BondLib.Sensors.Pbond pbond1; BondLib.Sensors.Pbond pbond2; BondLib.Sensors.Pbond pbond3; BondLib.Sensors.Pbond pbond4; BondLib.Sensors.Pbond pbond5; BondLib.Sensors.Pbond pbond6; Modelica.Blocks.Sources.Constant const(k=5); Modelica.Blocks.Math.Feedback feedback; Modelica.Blocks.Continuous.Integrator integrator;	Déclaration des composants avec leurs paramètres en utilisant les librairies disponibles tel que la BondLib pour les composants Bond Graph et Modelica.Blocks pour les éléments du diagramme en blocks.
equation	Fin des déclarations et début de la description du comportement

connect(pbond.BondCon2, i.BondCon1);
connect(j1p3_1.BondCon2, pbond.BondCon1);
connect(pbond1.BondCon2, j1p3_1.BondCon1);
connect(mSe.BondCon1, pbond1.BondCon1);
	Le comportement est décrit par

connect(j1p3_1.BondCon3, pbond2.BondCon1);
connect(pbond2.BondCon2, gY.BondCon1);
connect(gY.BondCon2, pbond3.BondCon1);
connect(pbond3.BondCon2, tF.BondCon2);
connect(tF.BondCon1, pbond4.BondCon1);
connect(pbond4.BondCon2, j1p3_2.BondCon1);
connect(j1p3_2.BondCon3, pbond5.BondCon1);
connect(pbond5.BondCon2, i1.BondCon1);
connect(j1p3_2.BondCon2, pbond6.BondCon1);
connect(pbond6.BondCon2, de.BondCon1);
connect(de.OutPort1, feedback.u2);
connect(const.y, feedback.u1);
connect(integrator.y, mSe.s);
connect(feedback.y, integrator.u);
 	
end BG_System;
	connection des composants déclarés, le comportement de chaque composant est défini dans la librairie dans laquelle il est déclaré.
	
	Fin du modèle nommé BG_System

En simulant ce modèle Modelica nous obtenons le graphe suivant (figure 9.22):

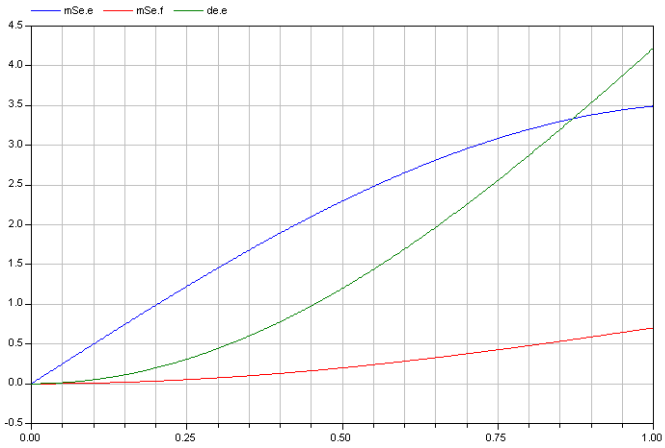


Figure 9.22. Graphe de la simulation du Bond Graph sur Dymola.

L'analyse des résultats de la simulation permettent de mieux comprendre le comportement du système. Ensuite les modifications apportées au modèle seront régénérées en code Modelica.

Conclusion générale

Dans cette thèse, nous avons proposé une méthodologie d'ingénierie système (IS) guidée par les modèles (ou Model-Driven Systems Engineering MDSE) avec une spécialisation pour les systèmes mécatroniques, nous basant sur les standards de l'IS (IEEE 15288) et de l'orienté-objet (standards OMG : MDA, UML/SysML). Nous avons intitulé cette méthodologie MISSyM. Nous nous sommes limités aux processus techniques de la recommandation IEEE 15288, car ce sont les processus qui se focalisent, entre autres, sur la conception du système. Utiliser les standards était pour nous une priorité car les ingénieurs ont besoin d'outils unifiés et pérennes. Mais ils ont également besoin comprendre comment mettre en œuvre tous ces standards chacun dans son domaine d'application et aussi comment les utiliser conjointement. Nous avons tenté de répondre à ce besoin en combinant ces standards et en expliquant comment ils s'articulaient au sein de MISSyM.

Pour le processus, nous avons choisi d'adapter les processus techniques du standard 15288 conjointement avec l'architecture de modèles MDA. Nous avons décrit un processus permettant la conception de systèmes hétérogènes de manière systémique. Ce processus est un processus itératif permettant la création de prototypes et leur simulation sur la plateforme Modelica. Très tôt au cours de ce processus, nous avons intégré un mécanisme de vérification de l'architecture basé sur la représentation en DSM et sur le calcul de métriques. Ce mécanisme de vérification est muni d'un algorithme d'amélioration de l'architecture mettant en valeur les composants hautement intégrés (à forte cohésion) pouvant faire l'objet de réutilisations futures.

Du point de vue des modèles, au niveau des PIM (ou Platform Independent Model), nous avons choisi pour base de travail le langage SysML. Nous avons ensuite évalué l'expressivité de SysML par rapport aux besoins décrits par les activités de la recommandation IEEE 15288, ce qui nous a amenés à apporter des extensions à SysML induites de cette évaluation. Nous avons regroupé ces extensions sous le profile SysML15288 pour permettre son utilisation de manière disjointe des autres

profiles proposés. Nous avons aussi proposé le profile SysML_Mecatronics contenant les extensions spécifiques aux systèmes mécatroniques. Ce profile regroupe le composant mécatronique ainsi que les extensions relatives aux connecteurs et aux ports. Nous avons aussi proposé l'intégration des DSM et des design patterns.

Au niveau PSM (ou Platform Specific Model), nous avons intégré les Bond Graphs pour fournir un outil d'analyse énergétique aux concepteurs. Nous avons aussi intégré le diagramme en blocks (DB) avec les Bond Graphs (BG) pour permettre de décrire des systèmes physiques commandés (processus en BG + commande en DB). Puis nous avons décrit les mécanismes de la génération du code Modelica correspondant et la simulation du modèle. Cela nous a aussi permis de démontrer les possibilités de cette approche en termes d'extensibilités et donc d'universalité.

Du point de vue de l'outil, nous avons choisi pour implémenter cette méthodologie l'outil Objecteering MDA Modeler (de SoftTeam) pour pouvoir travailler itérativement sur les modèles car cet outil est très bien adapté à un cycle itératif de conception de profiles. Il permet aussi de créer des générateurs de code de manière assez intuitive à travers des *templates* (ou patrons).

Perspectives

Pour la description des aspects d'ordonnancements, de contraintes temporelles, nous proposons d'utiliser le profile MARTE qui est en cours de standardisation. Dans cette thèse, nous n'avons pas abordé ces aspects mais un avantage des profiles UML est qu'ils peuvent être utilisés conjointement dans un même projet. MARTE est un profile connu des concepteurs qui viendra compléter les autres profiles standards d'UML. Ceci permettra un gain en temps en apprentissage de la conception des aspects temps réel.

Nous n'avons pas décrit la génération de code à partir du profile ModelicaML. Elle reste à être implémentée pour compléter la chaîne logicielle vers la simulation.

Le processus proposé est ouvert à la collaboration avec des outils CAD mécanique ou électrique (tel que CATIA) en utilisant des modèles 3D simplifiés pour le prototypage virtuel. Il serait utile d'augmenter cette collaboration en permettant le partage de modèles entre ces outils CAD et des outils PLM (Product Lifecycle Management), la norme STEP pourra apporter des réponses sur ces aspects.

D'autres langages peuvent être intégrés à SysML pour former une riche boite à outils, tels que les réseaux de Petri qui ont d'ailleurs déjà été intégrés dans UML nommés UML/PNO par **[Paludetto2004Petri]**. Ces travaux on montré que les réseaux de Petri sont utiles dans la conception et permettent la simulation. Ainsi SysML devrait intégrer les réseaux de Petri puisqu'ils sont très utilisés en conception. D'autres intégrations doivent être examinées tels que le langage VHDL-AMS (électronique) qui est aussi un outil très utile et permettant la simulation.

Un autre aspect important pouvant améliorer cette méthodologie et de créer une base de modèles réutilisables de composants mécatroniques et de design patterns. Ce genre d'outils peut être très utile dans une entreprise qui développe des solutions dans un même domaine, car les projets d'un même domaine ont souvent beaucoup de points communs, ce qui implique plus de réutilisation. Ceci peut aussi ouvrir de nouveaux types de services ; des entreprises peuvent se spécialiser dans la création de modèles de composants réutilisables et pourraient ainsi vendre des accès à des bases de modèles à d'autres entreprises qui assembleraient ces modèles pour concevoir leurs systèmes.

Pour augmenter les capacités graphiques de l'outil, il serait utile d'implémenter cette méthodologie sur des plateformes offrant plus de fléxibilité en termes de représentation graphique des extensions, en l'occurrence la solution *eclipse* EMF/GEF/GMF.

Enfin, l’application de la méthodologie dans un cadre de contrat industriel pourrait aider à affiner les modèles, le processus et la méthode de vérification de la qualité des modèles.

Références bibliographiques

[AADL2006] P.H. Feiler, D.P. Gluch, J.J. Hudak, *The Architecture Analysis & Design Language (AADL): An Introduction, Performance-Critical Systems*, February 2006, Technical Note CMU/SEI-2006-TN-011, disponible sur le site officiel http://www.aadl.info/

[Aldebaran ropbotics] http://www.aldebaran-robotics.com/pageProjetsNao.php

[Andersson2007] P. Andersson, M. Host, *UML and SystemC - a Comparison and Mapping Rules for Automatic Code Generation*, Proceedings of the Forum on specification and Design Languages conference (FDL), Barcelona, Spain, September 18 - 20, 2007

[ANSI] American National Standards Institute, http://www.ansi.org/

[ANSI/EIA 632] Document Number ANSI/GEIA EIA-632, *Processes for Engineering a System*, Government Electronics and Information Technology Association 01 Septembre 2003

[AP233] *AP233 public and private information portal*, http://www.ap233.org/

[AP233ConceptModel] D.W. Oliver, *Draft #12 of Concept Model for Systems Engineering MDSD review*, Version March 27, 2003, Wakefield, R.I. http://step.nasa.gov/

[Apvrille2004] L. Apvrille, J-P. Courtiat, C. Lohr, P. de Saqui-Sannes, *TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit*, IEEE Transactions on Software Engineering, vol 30, num 7, p 473, juillet 2004.

[Apvrille2006] L. Apvrille, P. de Saqui-Sannes, F. Khendek, *TURTLE-P: a UML profile for the formal validation of critical and distributed systems*, Springer-Verlag Software and Systems Modeling, vol 5 p 449–466, 2006.

[ATL2006] F. Allilaire, J. Bézivin, F. Jouault, I. Kurtev : *ATL - Eclipse Support for Model Transformation*. Proceedings of the Eclipse Technology eXchange workshop (eTX), the ECOOP 2006 Conference, Nantes, France. 2006

[Bahill1998] T. Bahill, B.gissing, *Re-evaluating systems engineering concepts using systems thinking*, IEEE Trans Syst Man Cybernet Oart C Appl Rev 28(4) pg 516-527, 1998

[Bahill2003] T. Bahill, J. Daniels, *Using Object-oriented and UML tools for hardware design: A case study*, Systems engineering vol6 n°1 p28-48. 2003

[Bansiya2002] J. Bansiya, C. Davis, *A hierarchical model for object-oriented design quality assessment*. IEEE Transactions on software engineering, vol. 28, No. 1, page. 4-17, 2002

[Baum2001] W.J. Baumol, A.S. Blinder, *Macroeconomics: Principles and Policy*, Orlando, FL: Harcourt 8ième edition 2001

[Berger2002] A. S. Berger. *Embedded Systems Design : An Introduction to Processes*, Tools and Techniques. CMP Books, ISBN 1 578 20073 3, 2002

[Bernandez 2005] B. Bernardez, A. Duran, M. Genero, *Metrics for use cases : A survey of current proposals. In Metrics for software conceptual models*. Imperial college press, 2005

[Bézivin2004] J. Bézivin, *Sur les principes de base de l'ingénierie des modèles*, RSTI-L'Objet 10(4):145—157. 2004

[Boehm1988] B. Boehm, A *Spiral Model of Software Development and Enhancement*, Computer, Mai 1988, pp. 61-72

[Boehm1994] B. Boehm, *A Collaborative Spiral Software Process Model Based on Theory W*, 11 Août, 1994

[BondLib] *Free library to model physical systems with bond graphs*, http://www.modelica.org/libraries/BondLib

[Borutzky1999] W. Borutzky, *Relations between graph based and object-oriented physical systems modelling*, ICBGM'99 International Conference on Bond Graph Modelling and Simulation, San Fransisco, CA, Janvier. 17-20, 1999, pp.11-17

[Broenink1999] J.F. Broenink, *Object-oriented modelling with bond Graphs and modelica*. International conference on bond Graph modelling ICBGM'99, Simulation series Vol 31 no 1,SCS, page 163-168, 1999

[Browning1998] T.R. Browning. *Use of Dependency Structure Matrices for*

Product Development Cycle Time Reduction. In Proceedings of the fifth ISPE international conference on concurrent engineering, research and applications, Tokyo, Japan 15-17 Juillet 1998

[Browning2001] T.R. Browning, *Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions*. IEEE Transactions on engineering management, Vol. 48, No. 3, August 2001

[Calvez1990] J.P Calvez, *Spécification et conception des systèmes : une méthodologie*. ISBN: 2 225 82107 0, Masson, 1990.

[Cano2007] J. Cano, N. Martinez, R. Seepold, F. López Aguilar**,** *Model-driven development of embedded system on heterogeneous platforms***,** Forum on Specification and Design Languages (FDL'07), Barcelona, Spain, September 18-20, 2007

[Caraa2004] C.T. Caraa, T.M. McGinnity, L.J. McDaid, *Integration of UML and VHDL-AMS for analogue system modelling*, Formal Aspects of computing 2004, 16: 80-94

[CATIA] *DSS CATIA*, http://www.3ds.com/fr/corporate/about-us/brands/catia/

[Clarkson2005] J. Clarkson, C. Eckert, *Design process improvement, a review of current practice*, ISBN 1-85233-701-X, Springer-Verlag London Limited 2005

[CORBA_OMG] *Corba OMG Homepage*, http://www.corba.org/

[Curtis1988] B. Curtis, H. Krasner, N. Iscoe, *A Field Study of the Software Design Process for Large Systems*, Communications of the ACM, Vol. 31, No. 11, 1988, pp. 1268-1287

[CWM2001] *Common Warehouse Metamodel (CWM) specification*. Object Management Group, OMG Document ad/2001-02-01, 2001

[Damm2005] W. Damm, B. Josko, A. Pnueli, A. Votintseva, *A discrete-time UML semantics for concurrency and communication in safety-critical applications*, Science of Computer Programming, Vol 55 page 81–115, 2005.

[DeRemer1976] F. DeRemer and H. Kron, *Programming-in-the-Large versus Programming-in-the-Small*, IEEE Trans. Software

Eng., vol 2, no 2, page 321-327, Juin 1976.

[Deursen2000] A.V. Deursen, P. Klint, J. Visser, *Domain-Specific Languages: An Annotated Bibliography*, SIGPLAN Notices, vol 35 no 6, 2000

[Dijkstra1968] E.W.G Dijkstra, The *Structure of THE Multiprogramming System*, Communications of the ACM 11, 341-346, Mai 1968

[Dijkstra1968b] E.W.G Dijkstra, *GO TO statement considered harmful*, Communications of the ACM 11, 147-148, Mars 1968

[Douglass2002] B.P. Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*, The Addison-Wesley Object Technology Series, ISBN-13: 978-0201699562, 3 octobre 2002

[DSL_Tools] *Domain-Specific Language Tools*, http://msdn2.microsoft.com/fr-fr/vstudio/aa718368(en-us).aspx or Visual Studio Extensibility Developer Center : http://msdn2.microsoft.com/fr-fr/vsx/default(en-us).aspx

[DSM] *DSM homepage*, Massachusetts Institute of Technology (MIT) and the University of Illinois at Urbana-Champaign (UIUC), http://www.dsmweb.org/

[Dudra2003] S.P. Dudra, Object-*Oriented Design Approach to Systems Engineering of a Mechanical Steering System*, International Truck & Bus Meeting & Exhibition, Ft. Worth, TX, USA, Session: Systems Engineering in Heavy Duty Vehicles, November 2003

[Dupuis2006] Y. Dupuis, *Conception d'une interface de marche pour la réalité virtuelle*, LISMMA SUPMECA Toulon (EA 2336), Soutenue le 8 novembre 2006

[Dymola] *The Dynamism Website*, http://www.dynasim.se/index.htm

[Eclipse Project] *Eclipse Modeling Project*, http://www.eclipse.org/modeling/

[EIA] *Electronic Industries Alliance*, http://www.eia.org/

[Endres2003] A. Endres, D. Rombach, *A Handbook of Software and Systems Engineering, Empirical Observations, Laws and Theories*, The Fraunhofer IESE Series on Software Engineering, Pearson Addison wesley ISBN 0 321 15420 7,

2003

[Estefan2007] J. A. Estefan, *Survey of Model Based Systems Engineering (MBSE) Methodologies*, Jet propulsion laboratory, California institute of technology, Pasadena, California, USA, INCOSE MBSE Focus Group, 25 May 2007

[Favre1997] W. FAVRE, *Contribution à la représentation bond graph des systèmes mécaniques multicorps*. Thèse de l'Institut National des Sciences Appliquées de Lyon, 1997

[Fernandez2004] L. Fuentes-Fernandez, A. Vallecillo-Moreno, *An introduction to UML profiles*, UPGRADE the European journal for the informatics Professional, http://www.upgrade-cepis.org, Vol. V, No 2, April 2004 UML and Model Engineering, pages 6-13, 2004

[Fontan2008] B. Fontan, *Méthodologie de conception des systèmes temps réel distribués en contexte UML/SysML*, Thèse en Informatique réalisée à l'université Toulouse III –Paul Sabatier, U.F.R,soutenue le 17 janvier 2008.

[Forsberg1995] K. Forsberg, M. Harold, *Application of the "Vee" to Incremental and Evolutionary Development*, Proceedings of the Fifth Annual International Symposium of the National Council on Systems Engineering, St. Louis, MO, Juillet 1995

[Friedenthal2003] S. Friedenthal, R. Burkhart, *Extending UML From Software To Systems*, Proceedings Of The INCOSE 2003, International Symposium, 2003

[Pop2007] A. Pop, D. Akhvlediani, P. Fritzson, *Towards Unified Systems Modeling with the ModelicaML UML Profile*, International Workshop on Equation-Based Object-Oriented Languages and Tools, Linköping University Electronic Press, Berlin, Germany, 2007

[Gandanegara2003] G. Gandanegara, *Méthodologie de conception systémique en Génie Electrique à l'aide de l'outil Bond Graph Application à une chaîne de traction ferroviaire*, thèse INP Toulouse 2003.

[Gérard2007] S. Gérard, P. Feiler, J.-F. Rolland, M. Filali, M.-O. Reiser, D. Delanote, Y., L. Pautet, I. Perseil, *UML and AADL 2007 Grand Challenges*, ACM Special Interest Group on

Embedded Systems, 4(4):1-17, October 2007

[Gamma1994] E. Gamma, R. Helm, R. Johnson, J.M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software***,** Addison-Wesley Professional Computing Series, ISBN-13: 978-0201633610, 1994

[HadjAmor2008] H. Hadj Amor, T. Soriano, *Integrating OpenModelica simulator with HLA*, Mecatronics2008, 7th edition of France-Japan (5th Europe-Asia) Congress on Mechatronics, Le Grand-Bornand, France, 21-23 Mai, 2008.

[Hales2000] M.K. Hales, R C. Rosenberg, *Structured modelling of mechatronic components using multiport templates*, Proceedings ASME IMECE 2000 DSC- Vol.69-2, page 787-794, 2000

[Harashima1996] F. Harashima, M. Tomizuka, T. Fukuda, *Mechatronics – What is it, why and how? An editorial*, IEEE/ASME Transactions on Mechatronics 1, 1–4. 1996

[Harmel2007] G. Harmel, *Vers une conception conjointe des architectures du produit et de l'organisation du projet dans le cadre de l'ingénierie système*, Thèse en automatique présentée à l'UFR des sciences et techniques de l'université de Franche-Comté, 05 juillet 2007.

[Henderson2002] B. Henderson-Sellers, D. Zoghbi, T. Klemota, S. Parasuram, *Sizing use cases: How to create a standard metrical approach*. Proceedings of the 8th Object-oriented Information Systems, Montpelier, France page 409-421. Springer-verlag, 2002.

[Hien2001] N.V. Hien, *Une méthode industrielle de conception de commande par automate hybride développée en objets*, thèse de l'université d'Aix-Marseille III, 2001.

[Hyperlynx] *CADInformatique Hyperlynx*, http://www.cadinformatique.com/Hyperlynx.htm

[IEC] International Electrotechnical Commission, http://www.iec.ch/

[INCOSE] *International Council On systems Engineering*, http://www.incose.org/

[INCOSE2004] Technical Board International Council on Systems

Engineering (INCOSE), *Systems engineering Handbook*, INCOSE-TP-2003-016-02, Version 2a, 1 Juin 2004

[IEEE1220_1998] IEEE Standard 1220-1998, *IEEE Standard for Application and Management of the Systems Engineering Process*, ISBN 0-7381-1544-4 SS94720

[IEEE15288] *15288 Adoption of ISO/IEC 15288:2002 Systems Engineering – System Life Cycle Processes*, IEEE Computer Society. 8 Juin 2005

[Isermann2000] R. Isermann, *Mechatronic systems: concepts and Applications*, Transactions of the Institute of Measurement and Control 22,1 (2000) pp. 29–55

[Jacobson1999] I.Jacobson, G.Booch, J.Rumbaugh, *The unified software development process*, Addison-wesley, Reading, MA, 1999

[Kadima2005] H. Kadima, *MDA Conception orientée-objet guidée par les modèles*, ISBN 2 10 007356 7, Dunod, Paris, 2005.

[Karnopp2000] D.C. Karnopp, D.L. Margolis, R.C. Rosenberg, *System Dynamics, Modeling and simulation of mechatronic systems*. John Wiley and Sons Inc. ISBN 0-471-33301-8, 2000

[KICAD] http://www.lis.inpg.fr/realise_au_lis/kicad/

[Kleppe2003] A. Kleppe, J. Warmer, W. Bast, *MDA Explained: The Model Driven Architecture(TM): Practice and Promise*, The Addison-Wesley Object Technology Series, ISBN-13: 978-0321194428, 1 Mai 2003

[Kreku2007] J. Kreku, M. Hoppari, K. Tiensyrjä, P. Andersson, *SystemC workload model generation from UML for performance simulation*, the Forum on specification and Design Languages (FDL'07), Barcelona, Spain, September 18-20, 2007

[LattaHomepage] M. Latta Homepage, http://michaellatta.spaces.live.com/

[Marchesi1998] M. Marchesi. *OOA metrics for the Unified Modeling Language. Software Maintenance and Reengineering,* Proceedings of the Second Euromicro Conference on Volume 8-11 Mar 1998 Page(s):67 – 73. 1998

[MARTE2007] *A UML Profile for MARTE, Beta 1*. OMG Adopted Specification OMG Document Number: ptc/07-08-04, Août 2007

[Massiani2005] A. Massiani, F. Nouvel. *Méthodologie de conception appliquée aux systèmes de radiocommunications de quatrième génération*. IETR rennes. MajecSTIC 2005

[MDA2003] J. Miller, J. Mukerji, *MDA guide version 1.0.1*, Object Management Group, Document n° omg/2003-06-01, 12 June 2003

[Modelica] *The Modelica Homepage*, http://www.modelica.org/

[MOF2006] *Meta Object Facility (MOF) Core Specification OMG Available Specification Version 2.0*, formal/06-01-01. Janvier 2006

[NASA1995] *NASA Systems Engineering Handbook: Fundamentals of Systems Engineering*. SP-610S juin 1995

[Nguyen2004] K. Dang Nguyen, Z. Sun, P.P. Thiagarajan, Weng-Fai Wong, *Model-driven design via executable UML to SystemC*. School of computing, national university of Singapore. Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International Volume , Issue , 5-8 Dec. 2004 Page(s): 459 – 468

[Oliver2005] I. Oliver, *Applying UML and MDA to Real Systems Design***,** Design, Automation, and Test in Europe, Proceedings of the conference on Design, Automation and Test in Europe - Volume 1, Pages: 70 – 71, 2005, ACM Special Interest Group on Design Automation, IEEE Computer Society Washington, DC, USA. ISBN ~ ISSN:1530-1591 , 0-7695-2288-2

[OMG] The Object management group, http://www.omg.org/

[OOSEM2001] H. Lykins, S. friedenthal, A. Meilisch, *Adapting UML for an Object Oriented Systems Engineering Method (OOSEM)*, Lockheed Martin, 2001

[OpenModelica] OpenModelica Homepage, http:// www.ida.liu.se/labs/pelab/modelica/OpenModelica.html

[Paludetto2004] M.Paludetto, J.Delatour, A.Benzina, *Vers une formalisation*

des besoins des systèmes embarqués, RSTI – TSI. Vol 23 – n° 4, pages 543 to 567, 2004

[Paludetto2004Petri] M. Paludetto, J. Delatour, A. Benzina, *UML et réseaux de Petri*, RSTI – TSI vol 23 – n°4, pg 543-567, 2004

[PAPYRUS] Papyrus homepage, an Open Source Tool for Graphical UML2 Modeling, CEA List, http://www.papyrusuml.org.

[PSM32] PSM32 Tools, http://www.problematics.com/

[Royce1970] W.W. Royce, *Managing the Development of Large Software Systems*, Proceedings of IEEE WESCON 26, pp. 1-9, Août 1970

[RUP] Rational Unified Process, Best Practices for Software Development Teams, Rational Software White Paper, TP026B, Rev 11-2001

[SDL] *SDL Forum Society*, http://www.sdl-forum.org/SDL/index.htm

[SDL2000] *Specification and Description Language*, Union Internationale des Télécommunications – Télécommunication Section Normalisation, Recommendation Z.100, Electronic Bookshop, Geneva, 2000

[Sghaier2005] A. Sghaier, *Une méthode pour le prototypage virtuel de machines industrielles*, thèse de l'université de sud Toulon-Var, 2005.

[Sharman2004] Sharman, D.M., Yassine, A.A., 2004. *Characterizing Complex Product Architectures.* In *Systems Engineering, vol.7 No1, 2004.* Wiley periodicals

[Simon1992] H.A. Simon, J. Schaeffer. *The Game of Chess*, Handbook of Game Theory with Economic Applications 1: 1-17, 1992

[sony] Sony aibo, http://support.sony-europe.com/aibo/index.asp

[STEP] STEP ISO 10303, http://step.nasa.gov/

[Stevens1974] W.P Stevens, G.J Myers, L.L. Constantine, *Structured Design*, IBM Systems J. 13, 2, 115–139, 1974

[Steward1981] D. V. Steward, *Systems Analysis and Management: Structure, Strategy and Design*. Petrocelli Books, New York, 1981

[Swarz2006] R.S. Swarz, J.K. DeRosa, *A Framework for Enterprise Systems Engineering Processes*, Journées Internationales Génie Logiciel & Ingénierie de Systèmes et leurs Applications, ICSSEA 2006

[SysML2007] *OMG Systems Modeling Language (OMG SysML™), V1.0*, OMG Document Number: formal/2007-09-01 Standard document URL: http://www.omg.org/spec/SysML/1.0/PDF

[Tien2003] J. M. Tien, D. Berg, A case for service systems engineering, ISSN 1004-3756/03/1201/13 Journal of Systems Science and Systems Engineering, CN11-2983/N ©JSSSE 2003 Vol. 12, No.1, pp13-38 Mars, 2003

[TOPCASED] *Toolkit in Open Source for Critical Applications & Systems Development*, http://www.topcased.org/

[TURKI2004] S.Turki, *Description d'un méta-modèle pour une plate-forme de simulation temps-réel*, mémoire DEA MCAO, LISMMA Cesti-Toulon, Universités de Provence, de la Méditerranée, d'Aix-Marseille III ENSAM, Juin 2004

[TURKI2005] S. Turki, T. Soriano, A. Sghaier, *a SysML profile for mechatronics integrating Bond Graphs*, 9th WSEAS International Conference on CIRCUITS, COMMUNICATIONS, COMPUTERS (CSCC'05), Vouliagmeni, Athens, Greece, July 11-16 2005

[UML SPT 2005] OMG document. *UML Profile for Schedulability, Performance, and Time Specification*, January 2005 Version 1.1. formal/05-01-02

[UML_OMG] UML Resource Page, http://www.uml.org/

[UML1.4] *OMG Unified Modeling Language Specification, version 1.4 september 2001*, http://www.omg.org/docs/formal/01-09-67.pdf

[UML2007] *Unified Modeling Language : Superstructure version 2.1.1*, formal 2007-02-03 Date February 2007. OMG Group

[UMLInfra 2004] UML 2.0 infrastructure specification, OMG Adopted Specification ptc/04-10-14. Novembre 2004

[UMLOMEGA] *The Omega UML Profile*, http://www-omega.imag.fr/index.php

[UMLSE RFP 2003] *UML for systems engineering RFP*, OMG Document: ad/03-03-41, 2003

[WARD1985] P. T. WARD, S. J. MELLOR, *Structured Development for Real-Time Systems*, Yourdon Computing series Yourdon press , Prentice Hall, 1985

[Weilkiens2008] T. Weilkiens, Systems *Engineering with SysML/UML, Modeling, Analysis, Design*. Morgan Kaufmann Publishers Inc 1ère édition, ISBN: 978-0-12-374274-2, 2008

[Wirth1971] N. Wirth, *The programming language Pascal*. Acta lnformatica 1, 1 (1971), 35-63

[Yurstseven1999] Yurtseven, M.K., *Systems Engineering and Soft Systems Methodology: A Review*, Dogus University Journal , 1; 225-230, 1999

[Zaytoon2001] J. Zaytoon, *Systèmes dynamiques Hybrides*, Hermes Science Europe Ltd, page 94, 2001

ANNEXE A : Le standard IEEE 15288

Les processus décrits dans le standard IEEE 15288 sont :

A.1 Le processus de définition des besoins des parties prenantes :

Ce processus permet la définition des exigences des parties prenantes sur le système. Les parties prenantes étant les utilisateurs, mais aussi les agents de maintenance, de formation et autres intervenants sur le système au cours de son cycle de vie. Ce processus doit permettre l'identification de ces intervenants et leurs besoins puis la transformation de ces besoins en exigences sur le système. Ces exigences seront la référence pour valider le système. Ce processus doit aboutir :

- Aux caractéristiques nécessaires et au contexte des services.
- Aux contraintes sur le système.
- A la traçabilité de ces exigences aux parties prenantes et à leurs besoins.
- Et aux premières exigences de haut niveau sur le système.

Les activités de ce processus :

- ✓ Identifier les parties prenantes
- ✓ Découvrir leurs besoins
- ✓ Définir les contraintes apparentes et incontournables du système
- ✓ Définir les scenarii pour retrouver les services fournis
- ✓ Identifier les interactions entre utilisateurs et le système
- ✓ Spécifier les exigences relatives aux caractéristiques critiques (sécurité, sûreté, santé et environnement etc.)
- ✓ Analyser les exigences obtenues en donnant des priorités en cas de conflits, compléter les sous-spécifications, lever les ambiguïtés.
- ✓ Résoudre les exigences erronées.
- ✓ Présenter ces exigences aux parties prenantes pour s'assurer de leur conformité à leurs besoins.

- ✓ Vérifier avec les parties prenantes si les exigences répondent bien à leurs besoins.
- ✓ Mettre les exigences dans un format de stockage accessible pendant le cycle de conception.
- ✓ Maintenir la traçabilité des exigences aux parties prenantes et leurs besoins originaux.

A.2 Le processus d'analyse des exigences

Ce processus doit permettre la transformation des services désirés en produit désiré. Les résultats attendus de ce processus sont :

- Les caractéristiques nécessaires, les fonctions et les performances nécessaires pour une solution satisfaisante.
- Les contraintes qui affecteront la conception et la manière de les respecter.
- La traçabilité de ces spécifications aux exigences.
- Un jeu de test de base pour vérifier que la spécification est satisfaisante.

Les activités effectuées au cours de ce processus sont :

- ✓ Définir les fonctions fournies par le système à l'extérieur.
- ✓ Définir pour chaque fonction que le système fournit, sa qualité de service et les conditions sous lesquelles le service est fonctionnel.
- ✓ Définir les contraintes de réalisation inévitables.
- ✓ Définir les mesures techniques et de qualité permettant de réaliser certaines fonctionnalités.
- ✓ Spécifier les exigences du système et ses fonctions en identifiant les risques ou la criticité relative à certaines caractéristiques (la sûreté, la sécurité, la disponibilité, etc).
- ✓ Analyser ces spécifications et s'assurer de leur intégrité.
- ✓ Définir la traçabilité de ces spécifications aux exigences.

- ✓ Continuer à enrichir ces spécifications par les choix, raisonnements, suppositions, etc.

A.3 Le processus de conception de l'architecture

Ce processus vise à synthétiser une solution qui satisfait aux exigences. Ce processus doit produire :

- La conception de l'architecture de haut niveau.
- Les descriptions des éléments à implémenter du système satisfaisant aux exigences.
- L'incorporation dans la solution de l'architecture des exigences des interfaces.
- La traçabilité de l'architecture aux exigences.
- Un jeu de test de base pour vérifier les éléments du système.
- La préparation pour l'intégration des éléments du système.

Les activités réalisées au cours de ce processus sont :

- ✓ Définir les architectures logiques appropriées en définissant les exigences dérivées de ces choix architecturaux.
- ✓ Partitionner les fonctions et les allouer à des éléments de l'architecture en générant les exigences relatives à ces allocations.
- ✓ Analyser ces architectures pour établir des critères de conception sur chaque élément (performance, comportement, durée de vie, etc.).
- ✓ Déterminer les allocations des exigences aux opérateurs.
- ✓ Déterminer quels éléments sont disponibles (à acheter ou réutiliser).
- ✓ Evaluer des solutions alternatives, les modéliser à un niveau de détail qui permet de les comparer aux exigences.
- ✓ Définir les interfaces entre éléments du système et ceux aux frontières du système.
- ✓ Spécifier la conception physique choisie.
- ✓ Enregistrer la conception architecturale.

- ✓ Maintenir la traçabilité entre l'architecture et les exigences.

Les trois processus décrits jusque là sont réitérés récursivement jusqu'à obtenir des éléments qu'on peut acheter, réutiliser ou construire.

A.4 Le processus de réalisation

Il a pour but de produire un élément du système. Il vise à définir les actions de fabrication. Les résultats attendus de ce processus sont :

- Une stratégie de réalisation /fabrication.
- Définition des contraintes liées à la technologie de réalisation.
- L'élément du système réalisé.
- Empaquetage et stockage de l'élément.

Les activités réalisées par ce processus sont :

- ✓ Générer une stratégie de réalisation (procédure de réalisation, processus de fabrication, etc.).
- ✓ Identifier les contraintes imposées par la stratégie et la technologie.
- ✓ Réaliser ou adapter les éléments du système.
  - o Fabrication du hardware, définition des techniques de fabrication et tests.
  - o Création du software et test.
  - o Formation des opérateurs.
- ✓ Montrer que les exigences ont été réalisées.
- ✓ Empaquetage de l'élément du système et stockage.

A.5 Le processus d'intégration

Ce processus vise à assembler le système. Les résultats attendus de ce processus sont :

- Une stratégie d'intégration.
- Les contraintes liées à l'intégration.

- Le système assemblé et prêt à être testé.
- Une liste des non-conformités aux exigences.

A.6 Les autres processus techniques

D'autres processus techniques sont décrits par le standard 15288 :

- ✓ Le processus de vérification : Il permet de confirmer que le système est conforme à la conception et qu'il satisfait aux exigences du système.
- ✓ Le processus de transition : Il vise à installer le système de manière à pouvoir fonctionner et fournir ses services.
- ✓ Le processus de validation : Ce processus veut prouver que le système fournit bien les services exigés.
- ✓ Le processus de mise en service : Il permet d'utiliser le système dans son environnement réel.
- ✓ Le processus de maintenance : Il permet au système de continuer à fournir correctement ses services.
- ✓ Le processus de fin de vie : Il permet l'arrêt de service du système.

A.7 Le Processus d'adaptation (tailoring)

Le processus d'adaptation est un guide permettant d'adapter et d'organiser les processus du standard 15288. Ainsi, l'application efficace de ce processus doit aboutir à un modèle de cycle de vie en termes d'étapes et leurs contributions au système et éventuellement à de nouveaux processus. Les activités couvertes par ce processus d'adaptation sont :

- ✓ Identifier les circonstances influençant l'adaptation comme par exemple :
  - o Stabilité et variabilité des environnements opérationnels.
  - o Les risques commerciaux ou de performance par rapport aux parties concernées.
  - o Nouveautés, taille, complexité.
  - o Date de départ et durée d'utilisation.

- Questions d'intégrité comme la sécurité, l'ergonomie, la disponibilité.
- Les technologies émergeantes.
- Profile du budget et ressources organisationnelles disponibles.
- Disponibilité des services des systèmes « support ».

✓ Recueillir des informations des parties concernées par le cycle de vie comme les parties prenantes du système.

✓ Utiliser le processus de prise de décision (processus de projet) pour décider de cette adaptation.

✓ Définir un modèle de cycle de vie permettant la création et l'utilisation du système de manière satisfaisante.

✓ Identifier le modèle de cycle de vie en termes d'étapes, de leurs objectifs et des contributions apportées.

- Les étapes données en exemple dans le standard peuvent être sélectionnées et réutilisés.
- Ces étapes données en exemple peuvent aussi être modifiées ou encore non adoptées.
- D'autres étapes peuvent être ajoutées et définies en termes d'objectifs et de contributions.

✓ Sélectionner les processus élémentaires du cycle de vie pouvant réaliser les contributions attendues de chaque étape définie dans le cycle de vie.

- Les processus du cycle de vie décrits dans le standard peuvent être sélectionnés comme tel ou modifiés.
- De nouveaux processus peuvent être décrits en terme de leurs objectifs et de leurs contributions.

Le standard propose donc d'organiser le cycle de vie en étapes contenant chacune un ensemble articulé de processus. Il présente en annexe un exemple d'adaptation que nous présentons dans le paragraphe suivant.

A.8 Exemple d'adaptation : Les étapes du cycle de vie

Le standard donne en exemple six étapes du cycle de vie :

- ✓ L'étape de définition du concept
- ✓ L'étape de développement
- ✓ L'étape de production
- ✓ L'étape d'utilisation
- ✓ L'étape de support
- ✓ L'étape de mise hors service

Nous en présenterons les deux premiers car ils seront particulièrement développés et adaptés dans le chapitre « solution adoptée ».

A.8.1 L'étape de définition du concept

Cette étape regroupe les processus permettant de mettre en évidence de nouvelles opportunités de marché et de développer des exigences système préliminaires et une solution faisable de la conception. Les productions de cette étape sont :

- ✓ L'identification de nouveaux concepts améliorant les fonctionnalités, les performances ou réduisant les coûts.
- ✓ Evaluation de la faisabilité de solutions et concepts selon les objectifs des parties prenantes.
- ✓ L'identification des besoins des parties prenantes et des exigences préliminaires sur le système.
- ✓ Raffinement des contributions des étapes du modèle de cycle de vie du système.
- ✓ Identification des risques, plans d'évaluation des étapes du modèle de cycle de vie du système.
- ✓ Identification des services nécessaires des systèmes supports à travers la vie du système.
- ✓ Plans d'exécution des étapes suivantes.
- ✓ Plans et critère de sortie de l'étape de développement.

- ✓ Identification de risques et plans d'évaluation de l'étape courante et des suivantes.
- ✓ Satisfaction du critère de sortie de l'étape.
- ✓ Accord pour passer à l'étape de développement.

A.8.2 L'étape de développement

Cette étape vise à développer un système qui répond aux besoins des acquéreurs et pouvant être produit, testé, évalué, mis en route, maintenu et aussi mis hors service. Les contributions de cette étape doivent être :

- ✓ Exigences du système, budget du projet et plan évalués et raffinés.
- ✓ L'architecture du système composée d'éléments matériels, logiciels et humains avec les interfaces internes et externes.
- ✓ Documentation de la vérification et de la validation.
- ✓ Confirmation que le système répond aux besoins, qu'il est productible, peut être mis en service, maintenable, peut être mis hors service et à un coût avantageux.
- ✓ Exigences raffinées des systèmes support.
- ✓ Informations techniques, tels que
    - o Diagrammes, modèles et dessins du matériel.
    - o Documentation de la conception du logiciel.
    - o Spécifications de l'interface
    - o Plans de production
    - o Instructions de mise en service
    - o Manuels de formation des opérateurs
    - o Procédures de maintenance
    - o Identification d'informations utiles pour la mise hors service.
- ✓ Construction d'un prototype ou d'un système final.
- ✓ Raffinement des contributions attendues et des estimations de coûts des étapes de production, utilisation, support et mise hors service.

- ✓ Définition des services de systèmes supports nécessaires dans les étapes suivantes.
- ✓ Plans et critères de sortie de la phase de production.
- ✓ Identification de risques courants.
- ✓ Critères de sortie satisfaits.
- ✓ Accord pour continuer vers l’étape de production.