

Valorisation de l'Ingénierie Système à Base de Modèles, pour l'analyse de sûreté de fonctionnement des systèmes complexes critiques intégrant des COTS

Robin Cressent

▶ **To cite this version:**

Robin Cressent. Valorisation de l'Ingénierie Système à Base de Modèles, pour l'analyse de sûreté de fonctionnement des systèmes complexes critiques intégrant des COTS. Autre. Université d'Orléans, 2012. Français. NNT : 2012ORLE2047 . tel-00812898

HAL Id: tel-00812898

https://theses.hal.science/tel-00812898

Submitted on 13 Apr 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ D'ORLÉANS

ÉCOLE DOCTORALE SCIENCES ET TECHNOLOGIES

Laboratoire PRISME

THÈSE présentée par :

Robin CRESSENT

Soutenue le : **12 décembre 2012**

pour obtenir le grade de : **Docteur de l'université d'Orléans**

Discipline/ Spécialité : Sciences et Technologies Industrielles

Valorisation de l'Ingénierie Système à Base de Modèles, pour l'analyse de sûreté de fonctionnement des systèmes complexes critiques intégrant des COTS

THÈSE dirigée par :

Frédéric KRATZ	Professeur des Universités, ENSIB

RAPPORTEURS :

Hamid DEMMOU	Maître de Conférences, HDR, Université Paul Sabatier
Jean-Marc FAURE	Professeur des Universités, SUPMECA

JURY:

Thomas COURRIERE	Responsable du projet LEA, MBDA
Hamid DEMMOU	Maître de Conférences, HDR, Université Paul Sabatier
Jean-Marc FAURE	Professeur des Universités, SUPMECA
Christine ROUSSELLE	Professeur des Universités, Université d'Orléans
Bruno VALLESPIR	Professeur des Universités, Université de Bordeaux - Président
Frédéric KRATZ	Professeur des Universités, ENSIB
Vincent IDASIAK	Maître de Conférences, ENSIB

Remerciements

Les travaux présentés dans ce mémoire ont été réalisés au sein du pôle IRAuS (Image, Robotique, Automatique, Signal) du laboratoire PRISME de l'Université d'Orléans et ont été accueillis au sein de l'Ecole Nationale Supérieure d'Ingénieurs de Bourges (ENSIB). Ils ont été financés par l'entreprise MBDA dans le cadre du projet LEA (Lietaoutnii Experimentalnii Aparrat).

Je tiens en premier lieu à remercier Madame Christine ROUSSELLE, Professeur des Universités, Directrice du laboratoire PRISME pendant la majeure partie de la durée de ma thèse, de m'avoir accueilli au sein de son laboratoire et d'avoir accepté d'être présente dans le jury. Je remercie également Monsieur François FALEMPIN, initiateur de la collaboration entre l'ENSIB et MBDA, d'avoir permis la réalisation de ces travaux. Je remercie également Monsieur Joël Allain, pour la mise à disposition des moyens de l'ENSIB et m'avoir ainsi permis de concrétiser ces travaux dans les meilleures conditions qui soient.

Ces travaux ont été dirigés par Monsieur Frédéric Kratz, Professeur des Universités à l'ENSIB et Monsieur Vincent Idasiak, Maître de Conférences à l'ENSIB, à qui je souhaite exprimer toute ma reconnaissance pour ces années de travail extrêmement enrichissantes, rigoureuses et conviviales. Je tiens à les remercier pour leur disponibilité, leurs grandes compétences scientifiques et leur soutien tout au long de cette thèse, y compris au cours de nos débats enflammés sur la définition d'un « système ». De même, je remercie Pierre David pour la qualité de son travail de thèse qui a initié le mien et pour les nombreux échanges scientifiques que nous avons pu avoir au cours de nos différentes collaborations.

Je remercie Messieurs Hamid DEMMOU et Jean-Marc FAURE, respectivement Maître de Conférences HDR à l'Université Paul Sabatier de Toulouse et Professeur des Universités à l'Ecole d'ingénieurs SUPMECA de Paris, qui ont accepté la charge de rapporteur. Ils ont également accepté de participer à mon jury, qu'ils trouvent ici l'expression de ma plus profonde reconnaissance.

Je remercie également l'ensemble des membres du jury qui ont accepté d'examiner avec attention ces travaux : Monsieur Bruno VALLESPIR, Professeur des Universités à l'Université de Bordeaux et Monsieur Thomas COURRIERE, Ingénieur Système en charge du projet LEA chez MBDA.

Je souhaite remercier tous les personnels du laboratoire PRISME et de l'ENSIB pour leur disponibilité qui ont permis un déroulement de cette thèse efficace et agréable. De même, j'adresse aussi mes remerciements à tous les membres du projet LEA pour leur participation et leur intérêt pour ces travaux.

Sur une note plus personnelle, je tiens à remercier mes compagnons d'infortune du quotidien : Julien PAVIER pour avoir partager avec moi ses résultats dans le cadre du PORK, malgré mon inexpérience dans son domaine d'expertise et Pierre THOUVENIN pour sa vivacité naturelle inspirante et ses pouvoirs jedi.

Je souhaite aussi remercier ma famille. Particulièrement mes grand parents pour leur attentions constantes et rassurantes, mes parents pour leur soutient et leurs conseils toujours à propos et mon frère, malgré son adhésion à la secte des iSheep. Je remercie également tous mes amis. Notamment, mes camarades de JdR qui m'ont permis de m'échapper de cette réalité pendant quelques heures chaque mois et Pauline à qui je souhaite de vivre l'aventure dont elle rêve.

De façon plus solennelle, je souhaite adresser mes plus sincères remerciements aux groupes de personnes qui m'ont permis de supporter ses trois années de labeur : *Amon Amarth*, groupe de métal viking, qui a reveillé ma routine en 1ère année de thèse, *Infectious Grooves*, groupe de métal funk, qui a égayé mes errances de 2ème année, ainsi que *Exodus* et *Walls of Jericho*, tous deux groupes de thrash métal, qui ont transcrit avec brio mes frustations de 3ème année en musique.

Enfin, je souhaite une bonne lecture de ces travaux à toute personne qui ne liera pas uniquement les remerciements.

Table des matières

Remerciements 3
Table des matières 5
Table des figures 9
Table des tableaux 11
Introduction Générale 13
Chapitre I. Problématique de la conception de systèmes embarqués sûrs 17
I. Introduction 19
II. Ingénierie Système 19
1. Terminologie 19
2. Processus d'IS 22
3. Ingénierie système à base de modèles (ISBM) 25
4. Quelques langages de modélisation 27
4.1. Simulink, modélisation, simulation et génération de code 27
4.2. AADL, modélisation des systèmes embarqués et temps réels 28
4.3. SysML, le langage de l'ingénierie système 30
III. Sûreté de fonctionnement 32
1. Concepts de la sûreté de fonctionnement 32
2. Exploitation de l'ISBM pour la SdF 33
2.1. Travaux antérieurs à SysML 33
2.2. MéDISIS, utilisation de SysML comme modèle central 34
2.2.1. Génération d'AMDEC composant 35
2.2.2. Génération de modèle Altarica DF 36
2.2.3. La Base de données des Comportements Dysfonctionnels (BCD) 36
IV. Les défis de l'utilisation de COTS 37
1. L'apport des COTS pour un projet industriel 38
2. L'impact des COTS sur les études de SdF 39
3. Intégration des COTS au système 39
3.1. Validation des COTS logiciels 40
3.2. Validation des COTS matériels 40
V. Les études SdF adaptés aux COTS 41
1. L'AMDEC fonctionnelle 41
1.1. Principes 41
1.2. Le processus de rédaction 42
2. Le guide FIDES 48
VI. Conclusion 49
Chapitre II. Processus d'ingénierie système supporté par SysML 51

I. Introduction 53
II. Elicitation des besoins 53
1. Description détaillée de l'activité 53
2. Réification de cette activité en SysML 55
III. Définition des exigences 58
1. Description détaillée de l'activité 58
2. Réification de cette activité en SysML 62
IV. Analyse Fonctionnelle 64
1. Description détaillée de l'activité 64
2. Réification de cette activité en SysML 65
V. Description Organique 67
1. Description détaillée de l'activité 67
2. Réification de cette activité en SysML 70
2.1. Définition et organisation des composants 70
2.2. Description détaillée des composants 75
VI. Conclusion 77
Chapitre III. Génération de pré-AMDEC fonctionnelles 79
I. Introduction 81
II. Exploitation de l'ISBM supporté par SysML pour l'AMDEC fonctionnelle 82
1. Les entités SysML utiles à l'AMDEC 82
2. Le processus de génération de pré-AMDEC fonctionnelle 85
2.1. Lister les fonctions du système 85
2.2. Établissement des modes de défaillances 87
2.3. Identifier l'environnement périphérique de chaque fonction 88
2.4. Rechercher le besoin fonctionnel dont dépend la fonction étudiée 91
2.5. Rechercher les composants support et les exigences impactées 92
2.6. Modes de défaillances spécifiques, causes internes, critères de criticité, moyens de réduction du risque 94
3. Utilisation de la pré-AMDEC par l'expert de SdF 94
4. Base de données des Dysfonctionnements des Fonctions : BDF 99
4.1. Les informations utiles 99
4.2. Utilisation de la BDF 101
4.3. Apport au travail de l'expert 102
III. Implémentation de l'algorithme de génération de pré-AMDEC fonctionnelle 103
1. La technique d'implémentation 103
2. Extraction des données 104
3. Ouverture de l'outil à d'autres sémantiques 108
IV. Conclusion 109
Chapitre IV. Intégration des études FIDES à la méthodologie MéDISIS 111

I. Introduction 113
II. La BCD de MéDISIS 113
1. Principes 113
2. Le méta-modèle de la BCD 114
III. FIDES 119
1. Présentation 119
2. Les principes de la méthodologie 120
IV. Intégration de l'évaluation de fiabilité FIDES à MéDISIS 124
1. Connexion de la méthodologie FIDES à la BCD 124
2. Valorisation du modèle système pour l'étude FIDES 127
3. Utilisation de la BCD et FIDES pour évaluer la fiabilité d'un composant 128
V. Conclusion 131
Chapitre V. Application au projet LEA 133
I. Introduction 135
II. Projet LEA 135
1. Présentation 135
2. Planification 137
III. Génération de la pré-AMDEC fonctionnelle de LEA 138
1. Elicitation des besoins 138
2. Analyse fonctionnelle 140
3. Pré-AMDEC 142
IV. Evaluation de fiabilité d'un COTS avec FIDES 145
V. Valorisation de l'ISBM pour la conception et la validation de LEA 149
1. Analyse d'ordonnancement 149
2. Simulation et injection de fautes 154
3. Développement du logiciel embarqué de LEA 157
VI. Conclusion 162
Conclusion Générale 165
Glossaire 169
Liste des publications de l'auteur 171
Références 173
ANNEXES 181
ANNEXE 1 : Différentes méthodes d'analyse de sûreté de fonctionnement 183

Table des figures

Figure I.1 : Domaines couverts par les 3 normes générales d'IS 22
Figure I.2 : Processus d'IS de définition du système 24
Figure I.3 Le modèle du système comme pivot du projet 27
Figure I.4 : Principe du Model Based Design selon Mathworks 27
Figure I.5 : Représentation graphique des différentes entités du langage AADL 29
Figure I.6 : Organisation des diagrammes SysML 31
Figure I.7 : La méthodologie MéDISIS 35
Figure I.8 Processus classique de rédaction d'AMDEC fonctionnelle 43
Figure II.1 : Exemple de diagramme de contexte pour l'élicitation des besoins 56
Figure II.2 : Exemple de UCD pour l'élicitation des besoins 57
Figure II.3 : Processus de définition des exigences 59
Figure II.4 : Processus de validation des exigences 61
Figure II.5 : Différentes représentations de la relation deriveReqt 63
Figure II.6 : Organisation des types d'exigences par décomposition 63
Figure II.7 : Processus d'analyse fonctionnelle 64
Figure II.8 : Organisation de la hiérarchie fonctionnelle 67
Figure II.9 : Processus de description de l'architecture organique 68
Figure II.10 : Relations de traçabilités entre les éléments du modèle système 70
Figure II.11 : Block Definition Diagram: Bicyclette 71
Figure II.12 : Value Type et Value Property 72
Figure II.13 : Relation d'héritage 72
Figure II.14 :Internal Block Diagram : Bicyclette 74
Figure II.15 : Calcul de vitesse de la bicyclette 76
Figure III.1 : Méta-Modèle de l'AMDEC fonctionnelle 83
Figure III.2 : Méta-modèle de l'OpaqueAction 85
Figure III.3 : Méta-modèle de la relation Refine 86
Figure III.4 : Attribution des id de fonction 87
Figure III.5 : Détermination de l'environnement périphérique d'une opaqueAction 88
Figure III.6 : Entrée et Sortie d'un niveau hiérarchique supérieur 89
Figure III.7 : Méta-modèle de Requirement SysML 92
Figure III.8 : Méta-modèle d'Allocation SysML 93
Figure III.9 : Diagramme d'activité appartenant à l'analyse fonctionnelle de la bicyclette 94
Figure III.10 : Méta-Modèle de la BDF 100
Figure III.11 : Capture d'écran de l'outil logiciel de rédaction de pré-AMDEC 108
Figure IV.1 : Méta-modèle de la BCD 115
Figure IV.2 : Allocations entres les concepts de la BCD 116

Figure IV.3 : Exemple d'un mode de défaillance représenté dans la BCD 117
Figure IV.4 : Exemple de description détaillée du déclenchement 117
Figure IV.5 : Exemple de « DéfOpération » à deux niveaux de granularité 118
Figure IV.6 : Exemple de STM « DéfComportement » 119
Figure IV.7 : Le modèle mathématique FIDES d'évaluation de fiabilité modélisé en SysML 124
Figure IV.8 : Exemple de diagramme de contexte 128
Figure IV.9 : Schéma électrique du produit d'exemple 129
Figure IV.10 BDD et IBD du produit d'exemple 129
Figure IV.11 : Extrait de la BCD: Résistance 130
Figure V.1 : Chronologie de l'essai du véhicule LEA 136
Figure V.2 : Planification des activités des DSL pour le projet LEA 137
Figure V.3 : Diagramme de contexte du véhicule LEA 139
Figure V.4 : UCD des besoins du véhicule LEA, phase d'essai en vol 139
Figure V.5 : AD de la fonction : « Contrôler le vol autonome » 140
Figure V.6 : AD de la fonction : « Mettre en route le moteur » 141
Figure V.7 : Exemple d'utilisation de l'élément SysML : *rationale* 142
Figure V.8 : BDD présentant l'architecture globale du véhicule LEA 146
Figure V.9 : Représentation des interfaces d'un composant aux niveaux BDD et IBD 147
Figure V.10 : Décomposition du Promux en fonctionnalités techniques 148
Figure V.11 : Représentation de la décomposition organique au niveau BDD 149
Figure V.12 : IBD du véhicule LEA 152
Figure V.13 : SD de la transmission Ethernet des données capteurs 153
Figure V.14 : Modèle AADL du véhicule LEA 153
Figure V.15 : Modélisation de la contrainte sur l'âge des données capteurs avec un ParD 154
Figure V.16 : Modèle de simulation opérationnelle et dysfonctionnelle 156
Figure V.17 : Différents types de « switch » en Simulink 156
Figure V.18 : Assertion de la contrainte sur l'âge des données capteurs en Simulink 157
Figure V.19 : Exemple de simulateur temps réel dSPACE 159
Figure V.20 : Interface de pilotage et d'observation du boitier dSPACE 160
Figure V.21 : Configuration HIL de validation fonctionnelle du logiciel de vol 160
Figure V.22 : Configuration HIL de validation de la maquette fonctionnelle 161
Figure V.23 : Photographie de la maquette fonctionnelle dans notre laboratoire 161
Figure CG.1 : Méthodologie MéDISIS à l'issue des travaux présentés dans la thèse 167

Table des tableaux

Tableau I.1 : Présentation des types de diagrammes SysML 31
Tableau I.2 Forme tabulaire générique pour AMDEC fonctionnelle 42
Tableau I.3 Matrice de criticité des modes de défaillance 46
Tableau II.1 : Représentation SysML des concepts de l'élicitation des besoins 57
Tableau II.2 : Représentation SysML des concepts de définition des exigences 62
Tableau II.3 : Représentation SysML des concepts de l'analyse fonctionnelle 66
Tableau II.4 : Représentation SysML des concepts de la description organique (définitions et interfaces) 73
Tableau II.5 : Représentation SysML des concepts de la description organique (hiérarchie et interactions) 75
Tableau III.1 : Entités SysML utiles à l'AMDEC fonctionnelle 84
Tableau III.2 Forme tabulaire générique pour la pré-AMDEC fonctionnelle 85
Tableau III.3 : Extrait de pré-AMDEC fonctionnelle 96
Tableau III.4 : Extrait de l'AMDEC fonctionnelle complétée 98
Tableau IV.1 : Connexion des concepts de la méthodologie FIDES et de la BCD 125
Tableau IV.2 : Paramètres d'évaluation de la fiabilité extraits de la BCD 130
Tableau IV.3 : Description du cycle de vie, déduit du modèle système 130
Tableau V.1 : Extrait de la pré-AMDEC fonctionnelle du véhicule LEA 143
Tableau V.2 : Table de correspondance des concepts entre AADL et SysML 151
Tableau V.3 : Table de correspondance des concepts entre Simulink et SysML 155

Introduction Générale

À l'heure actuelle et depuis plusieurs années, les nouveaux systèmes développés par les industriels ne cessent de se complexifier, de faire intervenir toujours plus de technologies différentes et cela, pour maximiser la rentabilité, améliorer les fonctionnalités, voire proposer de nouveaux services. Quelque soit le domaine concerné : Énergie, Transport terrestre, Aéronautique, Productique, … nous pouvons remarquer une augmentation du nombre de composants et de fonctions réalisées par du logiciel et un besoin important de faire collaborer plusieurs domaines d'expertise à différents niveaux du projet (conception, fabrication,…). De plus, dans un contexte économique et financier de crise tel que nous le vivons aujourd'hui, les aspects rentabilité, diminution de temps de conception et réduction de coûts sont plus que jamais sur le devant de la scène.

Afin de réduire les coûts de développement des systèmes complexes critiques, il est nécessaire de planifier l'ensemble des activités des différents domaines d'expertise dès les premières phases de spécification et anticiper les éventuels points durs. L'effort de recherche a été porté sur l'intégration des analyses de Sûreté de Fonctionnement (SdF) à un processus d'Ingénierie Système (IS), afin de réduire le temps nécessaire à l'analyse du système et de maximiser l'utilité des activités d'IS. Dans le cadre de la thèse, nous nous concentrons sur les phases de spécification et de conception du système. Un point particulier sera étudié : l'utilisation de COTS (Component Off The Shelf). En effet, dans le cadre de projet industriels actuels, de multiples partenaires et sous-traitants interviennent, apportant un nombre conséquent de COTS à intégrer et valider. Ces composants sur étagères représentent un défi particulier quant à la validation des exigences de SdF lorsqu'ils sont intégrés à un système présentant de fortes contraintes de sûreté de fonctionnement.

La conception de ces systèmes nécessite le respect de différentes normes et standards afin d'être validée. Ainsi, l'ensemble des systèmes complexes impose une phase importante de validation de leur comportement aussi bien fonctionnel que dysfonctionnel. Répondre à ces besoins de validation est également l'objet des analyses de sûreté de fonctionnement. Or la sûreté de fonctionnement ne peut obtenir des résultats efficients sans avoir un accès complet aux données de conception du système. Afin de maintenir la cohérence des données du système à travers l'ensemble des domaines d'expertise, notamment la SdF, des techniques rigoureuses d'IS sont primordiales. Le défi sous-jacent est donc de favoriser l'interconnexion des activités d'IS et de domaines spécifiques et d'optimiser le temps nécessaire à l'échange d'informations entre outils. Le but recherché est celui évoqué par les auteurs de [Rauzy et al. 2008] : « [dissoudre] les frontières entre les environnements de développement virtuel au service des Concepteurs, et les référentiels de simulation accompagnant le travail des Fiabilistes, Logisticiens et Risk Managers, […] pour laisser place à une discipline commune que l'on pourrait qualifier d'Assurance Performentielle… ».

L'étape préalable indispensable est la compréhension des formalismes, méthodes et outils employés par les différents domaines d'expertise considérés. Lorsque les interfaces de chaque processus et activité sont bien définies, il est alors possible de concevoir les liens. Nous considérons dans cette thèse l'Ingénierie Système à Base de Modèles (ISBM) qui est une bonne alternative d'ingénierie pour le développement de systèmes complexes critiques. Depuis quelques années, l'ISBM dispose d'un langage de modélisation qui lui est dédié : SysML. Ce langage permet de créer des modèles exprimant des concepts inhérents à toutes les technologies susceptibles d'être employées pour la

description d'un système. C'est pourquoi dans l'ensemble de cette thèse nous avons utilisé SysML en tant que support à toutes les activités d'ingénierie système.

Dans la continuité des travaux de Pierre David [David 2009], nous avons étudié les corrélations existantes entre les concepts manipulés par SysML est ceux nécessaires à la réalisation d'études de SdF. La méthode MéDISIS (Méthode D'Intégration des analyses SdF au processus d'Ingénierie Système) décrite dans [David 2009], avait pour but principal la détermination du comportement dysfonctionnel d'un système à partir de la description fonctionnelle de celui-ci en SysML. Ainsi MéDISIS dispose de deux principaux processus de traduction, de SysML vers l'AMDEC composant et de SysML vers Altarica DataFlow. Lors de la mise au point de MéDISIS, il a aussi été mis en avant la nécessité de stocker l'information dysfonctionnelle apportée par les experts de SdF, dans une base de données utilisable par les processus de traduction. En effet, il est préférable de ne pas « polluer » le modèle fonctionnel du système en SysML avec l'ensemble des informations dysfonctionnelles, mais ces informations permettent une amélioration continue de l'efficacité de la traduction. C'est ainsi que la Base de données des Comportements Dysfonctionnels (BCD) a été mise au point. La BCD respecte un méta-modèle basé sur SysML qui permet une interaction forte et rapide avec le processus de traduction. En particulier, la thèse s'inscrit dans la continuité de ces travaux et s'efforce d'enrichir la méthodologie MéDISIS. Les objectifs sont quelque peu différents puisque nous travaillons maintenant sur l'ensemble des étapes de l'ingénierie système et non plus uniquement sur la partie description organique comme cela était le cas dans les travaux précédents de notre équipe. Notre but est d'étendre les fonctions MéDISIS pour l'analyse d'impact de l'intégration de COTS.

Ces considérations sont au cœur du partenariat entre notre équipe de recherche et MBDA dans le cadre du projet LEA (Lietaoutnii Experimentalnii Apparat). Le projet LEA, initié par MBDA et l'ONERA, a pour objectif de définir un véhicule expérimental à statomixte pour réaliser et tester des prototypes en conditions réelles de vol. Le moteur à statomixte est un concept de statoréacteur fonctionnant d'abord en combustion subsonique, puis en combustion supersonique, pour pouvoir évoluer dans un domaine de Mach très large (2 à 8). Le véhicule LEA évoluera, lors des essais en vol, à des vitesses comprises entre Mach 4 et Mach 8, dans la stratosphère (12 à 50 km d'altitude). Les essais auront lieu sur le territoire russe en collaboration avec l'entreprise RADUGA qui assurera la mise en conditions de tests du véhicule LEA. Au terme de l'essai en vol, le véhicule est détruit, rendant critique, lors de la phase de vol, la récupération et la transmission au sol des données des quelque 300 capteurs installés sur le fuselage et le moteur du véhicule. Ainsi, le système embarqué (calculateur de vol, centrale d'acquisition, …) qui gère ces fonctions représente un bon exemple de système complexe critique. Ce projet industriel nous permet donc d'illustrer et de valider qualitativement les travaux de recherches menés au cours de la thèse.

La présentation de ces travaux est organisée selon le plan suivant :

- Dans le chapitre I, nous présentons la problématique de la spécification et de la conception de systèmes complexes à fortes contraintes de sûreté de fonctionnement. Nous exposons les principes de l'IS et de l'ISBM. Nous présentons ensuite des langages de modélisation et rappelons ce qui motive le choix de SysML. Ensuite, nous verrons des méthodes spécifiques de sûreté de fonctionnement qui sont utiles aux analyses d'un système complexe et comment elles ont été rapprochées du processus d'IS. À travers la présentation des défis imposés par l'utilisation de COTS, nous sommes amenés à détailler les approches de SdF adaptées à ces

composants : notamment, l'AMDEC fonctionnelle qui s'applique sans connaissance approfondie de l'architecture organique, et le guide FIDES qui propose une méthode d'évaluation de la fiabilité de COTS.

- Dans le chapitre II, nous montrons que le choix du langage SysML ne suffit pas lorsque l'on traite l'ensemble des activités d'IS. Il est alors nécessaire de spécifier la sémantique correspondant aux entités graphiques du langage. C'est pourquoi dans cette partie, l'ensemble du processus d'ingénierie système est décrit et les règles de modélisation SysML associées à ces activités sont présentées. Les quatre grandes étapes de l'ingénierie système sont traitées : élicitation du besoin, définition des exigences, analyse fonctionnelle et description organique. Le but de chaque activité d'ingénierie système est expliqué en décrivant la sémantique qui doit être appliquée aux éléments du langage SysML. À l'issue de ce chapitre, nous disposons de règles de modélisation SysML adaptées à notre propos et à nos besoins.

- Dans le chapitre III, nous décrivons le méta-modèle sous-jacent à la pratique de l'AMDEC fonctionnelle et présentons ensuite les connaissances du modèle système en SysML utiles à la rédaction de l'AMDEC. Les algorithmes de traitement de données SysML permettant de fournir à l'expert en sûreté de fonctionnement une pré-AMDEC seront détaillés. Cette pré-AMDEC a pour objectif de structurer les informations SysML pour maximiser l'efficacité de l'étude de l'expert. Nous verrons aussi, succinctement, les moyens à notre disposition pour l'implémentation de ce processus de traduction/génération d'AMDEC fonctionnel dans un outil logiciel manipulant les modèles SysML au format XMI (format d'échange standardisé).

- Dans le chapitre IV, nous détaillons le fonctionnement et la structure de la BCD de MéDISIS. Puis nous utilisons le diagramme paramétrique de SysML pour représenter la méthode de calcul de taux de défaillance propre au guide FIDES. Nous mettons ensuite en avant les relations évidentes qui existent entre la base de données définie conjointement à la méthodologie MéDISIS et le retour d'expérience formalisé dans le guide FIDES. De sorte qu'il est alors possible d'intégrer la méthode de calcul FIDES à la BCD décrite par [David 2009], [Cressent et al. 2012a] et [Cressent et al. 2012b] et de décrire un processus permettant de réaliser une étude FIDES à partir du modèle SysML.

- Dans le chapitre V, nous présentons le projet industriel LEA et les défis qu'il représente. Nos règles de modélisations sont appliquées pour l'ensemble des activités d'IS. Les processus utiles de la méthodologie MéDISIS sont alors déployés : aussi bien ceux décrit dans cette thèse que ceux mis au point spécifiquement pour le projet LEA que nous décrirons succinctement. Enfin, l'utilité de ces processus pour la planification des activités de conception et validation du système embarqué de LEA est illustrée.

Chapitre I. Problématique de la conception de systèmes embarqués sûrs

I. Introduction

Dans le cadre de cette thèse, nous nous intéressons à la conception de systèmes embarqués à fortes contraintes de sûreté de fonctionnement. Dans le cadre du projet LEA, le système conçu est innovant, utilisant de multiples technologies pour de meilleures performances. De plus, un processus de développement réduisant les délais et les coûts est testé pour la conception du calculateur de vol. De nombreux challenges se révèlent alors : comment interfacer des composants de technologies différentes, voire de natures différentes ? Comment assurer des critères de performance ou de sûreté pour de tels systèmes ? Comment organiser les activités d'experts de spécialités différentes dans un but commun ? Comment tenir compte des priorités parfois antagonistes de ces experts ? Comment prévenir au mieux les surcoûts que peut engendrer la réalisation d'un tel projet ? L'ensemble des recherches menées en Ingénierie Système (IS) vise à apporter des éléments de réponse à toutes ces questions. De plus, à l'heure actuelle, une solution aux problèmes d'acquisition de nouvelle technologie et de temps de développement trop long existe : l'utilisation de composants sur étagère appelés COTS. Cependant, les COTS sont difficiles à valider lorsqu'ils sont utilisés pour réaliser des fonctions où la sûreté de fonctionnement est primordiale. Ainsi, il est nécessaire de trouver comment évaluer l'impact de l'introduction de COTS dans un système soumis à de fortes contraintes de SdF. L'objet de cette thèse consiste à montrer l'intérêt de l'ingénierie système pour évaluer cet impact dans le cas de la conception d'un système complexe embarqué tel que le véhicule LEA.

Dans ce premier chapitre, nous commencerons par définir l'ensemble des termes utiles au domaine de l'ingénierie système. Un état des lieux des pratiques d'IS est ensuite réalisé. Nous détaillerons l'approche ISBM : Ingénierie Système à Base de Modèles (de l'anglais, MBSE : Model Based System Engineering) pour l'étude de systèmes complexes. Nous verrons les particularités de différents langages d'ISBM pour l'étude de systèmes complexes embarqués et quels sont les avantages de chacun afin de choisir le langage adapté aux besoins de spécifications d'un système complexe. Dans un deuxième temps, nous étudierons comment différentes pratiques de SdF peuvent bénéficier d'une approche ISBM. Nous détaillerons notamment, le cadre méthodologique de MéDISIS qui propose différentes solutions pour valoriser une approche ISBM en SysML pour la réalisation d'étude de SdF. Enfin, nous verrons en quoi consiste exactement la problématique liée à l'utilisation de COTS. En classifiant les différents types de COTS, nous serons amenés à identifier les méthodes de SdF applicables ou non dans un contexte où des COTS sont utilisés. Enfin, l'intérêt d'une méthodologie comme MéDISIS sera présenté pour valoriser l'approche ISBM pour la conception d'un système complexe intégrant des COTS.

II. Ingénierie Système

1. Terminologie

Afin de pouvoir définir l'ingénierie système, il d'abord nécessaire de définir le concept même de « système » :

Système : La définition donnée par l'AFIS (Association Française d'Ingénierie Système) décrit un système comme « un ensemble d'éléments en interaction entre eux et avec l'environnement, intégré

pour rendre à ce dernier les services correspondants à sa finalité » [AFIS 2009]. Les normes IEC 61508, 61511 [IEC 61508][IEC 61511] et l'INCOSE (INternational COuncil on Systems Engineering) [INCOSE 2010] insistent sur la dimension de composition en indiquant qu'un élément constitutif peut être un autre système appelé alors sous-système. Toutes ces définitions considèrent également que les éléments peuvent être de toute nature : matérielle, logicielle, humaine, documentaire…

On pourra orienter la nature de notre système en qualifiant le système de plusieurs adjectifs :

Système complexe : Le qualificatif complexe a été débattu par [Magee et al. 2004] et [Guillerm 2011]. Ce que nous pouvons retenir est que le terme « complexe » apporte autant une notion de taille que de granularité pour le nombre et la complexité des éléments constitutifs de notre système. Un système complexe se caractérise donc par :

- un grand nombre d'éléments avec de nombreuses interactions entre éléments créant de nombreuses dépendances à analyser.
- ou des éléments eux-mêmes complexes que ce soit au niveau physique, fonctionnel ou opérationnel.

Système critique : Un système sera critique si de son bon fonctionnement dépendent des enjeux importants, tant en termes financiers, humains que logistiques. La conception d'un système critique va donc de pair avec la définition de fortes contraintes de sûreté de fonctionnement qui valideront ou non l'aspect dangerosité du système.

Système embarqué : Le sens du qualificatif embarqué retenu dans la thèse est celui d'un système destiné à une exploitation dans des conditions environnementales particulières et introduisant une dualité entre l'aspect logiciel et matériel. Ce qualificatif est commun dans les domaines de l'aéronautique, de l'automobile et du transport en général où les ressources sont limitées et les contraintes physiques sont fortes. De plus, dans le domaine de l'ingénierie des systèmes embarqués, un dernier point est omniprésent, les contraintes temporelles. Ainsi, un système embarqué est un système dont les fonctions doivent s'exécuter de façon à réagir en un temps minimum aux stimuli de son environnement.

Si l'on considère que l'ingénierie consiste en l'étude et en la réalisation d'un projet, alors on retrouve la définition de l'IS donnée par la norme IEEE 1220 [IEEE 1220] :

Ingénierie Système : il s'agit d'une approche interdisciplinaire qui concerne la mise en œuvre des capacités nécessaires à la réussite de la réalisation des systèmes (complexes, critiques, embarqués…). Afin d'assurer cet objectif, l'IS comprend diverses techniques d'organisation et de réalisation des tâches.

Le but de l'IS est donc d'organiser et formaliser le processus de création afin de permettre la conception de nouveaux systèmes caractérisés par un fort besoin d'innovation [Friedenthal et al. 2008]. Le fait de rendre les systèmes développés innovants, et par là même compétitifs, dépend de nombreux critères tels que :

- Intégrer de multiples technologies.
- Atteindre un haut niveau de complexité.
- Réduire les coûts de développement.
- Améliorer les fonctionnalités.

- Améliorer l'interopérabilité.
- Améliorer les performances.
- Améliorer la FMDS (Fiabilité, Maintenabilité, Disponibilité, Sécurité)
- Réduire la taille.
- Intégrer et créer des Systèmes de Systèmes.

L'IS concerne le processus de développement des systèmes dans sa globalité, en intégrant toutes ses parties prenantes, dans toute sa durée. Les activités d'IS regroupent donc l'ensemble des activités exécutées de la naissance du besoin, à la livraison du produit final, voire jusqu'à sa fin de vie. D'un point de vue technique, la méthode utilisée pour l'IS doit permettre de traiter :

- Des niveaux d'abstraction différents.
- Des exigences évoluant au cours du projet.
- Des interconnexions multiples entre technologies multiples.
- Des contraintes liées à l'environnement du système et sa mission.

D'un point de vue général, l'IS inclut les activités techniques et de management de projet dont les objectifs sont :

- Satisfaction des parties prenantes.
- Recherche de qualité technique.
- Tenue des délais et des budgets.

L'IS se compose donc de moyens d'organiser les activités des différents corps de métiers et de spécifier l'ensemble des informations du système étudié. Pour cela, une politique d'IS devra définir ces processus, ses méthodes et méthodologie, ses outils et son langage. L'ensemble de ces termes ne fait pas l'objet d'un consensus dans tous les domaines qui l'emploient ni même au sein même de la communauté d'IS. C'est pourquoi nous nous proposons de les définitions suivantes. La définition de « Processus » par [Estefan 2008] s'accorde avec celle de l'AFIS [AFIS 2009] :

Processus : Un processus est un ensemble d'activités corrélées ou interactives qui transforme des éléments d'entrée en éléments de sortie. Un processus définit la logique d'organisation des activités à accomplir sans préciser comment chaque activité est réalisée.

[Estefan 2008] évoque ensuite la définition de « méthode » comme étant un ensemble de techniques utilisées pour réaliser une activité. Cependant [Estefan 2008] explique aussi que chaque méthode peut être perçue comme un processus, avec une séquence de tâche à accomplir. En d'autres termes, le « Comment » d'un niveau d'abstraction donné devient un processus du niveau directement inférieur. Par contre, même si chaque processus peut lui-même être composé de sous-processus, il existe un « Comment » unitaire, non sécable sous forme de processus au plus bas niveau. C'est pourquoi dans la thèse nous définirons la méthode ainsi :

Méthode : Il s'agit du processus technique qui permet la création de l'information qui est alors exprimée dans le langage de description système. Il s'agit du processus technique de plus bas niveau dans une branche d'imbrication de processus.

Cette dernière définition introduit un nouveau terme : le langage de description système. À partir de la définition du dictionnaire [Académie Française 1935] et celle issue des travaux de Chomsky sur la linguistique théorique [Chomsky 1956], nous pouvons proposer la définition suivante :

Langage : Il s'agit d'un ensemble de phrases de longueurs finies, elles-mêmes composées à partir d'un nombre fini de symboles formant l'alphabet, permettant la communication et l'expression de la pensée. Un langage se compose d'une syntaxe (ou grammaire) qui régit l'organisation des symboles et des phrases, et d'une sémantique qui décrit comment interpréter les phrases et symboles pour accéder au sens de ce qui est représenté à l'aide du langage.

Syntaxe : La syntaxe d'un langage est l'ensemble des règles qui régissent la façon dont les symboles du langage peuvent ou doivent être agencés. La syntaxe régule entre autres la mise en forme des symboles, les relations entre ces symboles ainsi que leur représentation. La syntaxe correspond à la forme que les phrases du langage peuvent prendre.

Sémantique : La sémantique décrit comment la structure d'une phrase (i.e. la syntaxe de la phrase) peut être interprétée pour déduire le sens des symboles qui composent la phrase et de la phrase elle-même. La sémantique permet d'accéder aux concepts qui sont réifiés sous forme de phrase (utilisant l'alphabet du langage et respectant la syntaxe du langage).

Nous avons donc défini l'ensemble des termes qui sont régulièrement manipulés lorsque l'on traite de l'ingénierie système. Nous allons maintenant nous intéresser aux différents processus d'IS existants.

2. Processus d'IS

Il existe plusieurs référentiels et standards traitant de l'ingénierie système, notamment l'EIA 632 [EIA 632], l'IEEE 1220 [IEEE 1220] et l'ISO 15288 [ISO 15288]. Chacun de ces standards s'attache à décrire à des niveaux de détails variés comment mener à bien une stratégie d'ingénierie système. La Figure I.1 issue de [AFIS 2009] traduit l'étendue et le niveau de détails pris en compte par chacune de ces normes. Sur cette figure, la hauteur de l'ellipse traduit le niveau de détails transcrit dans le standard, alors que la largeur de l'ellipse correspond au périmètre des domaines couverts.

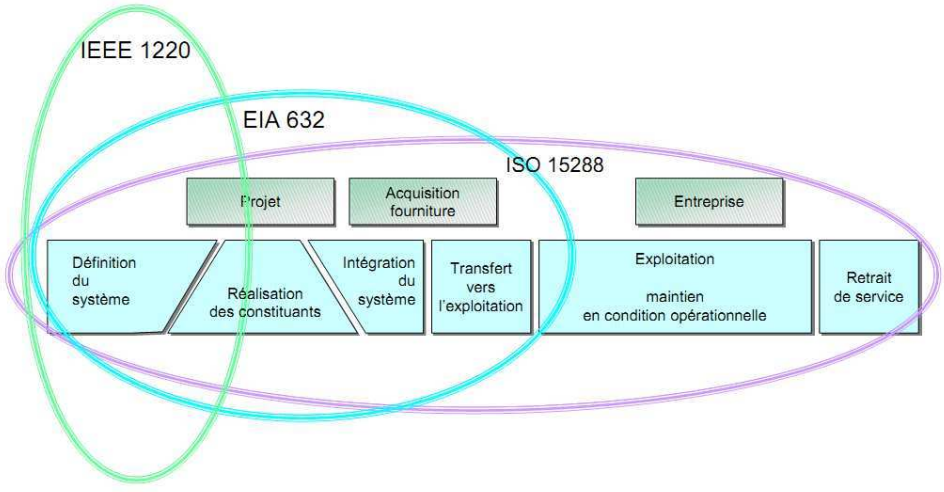


Figure I.1 : Domaines couverts par les 3 normes générales d'IS

Ces normes sont maintenues par des organisations d'ingénieurs : American National Standards Association (ISO 15288), Electronic Industries Alliance (EIA 632) et International Electronic and Electrical Engineers (IEEE 1220). On trouve aussi des manuels d'applications de l'IS plus spécialisés édités par des associations du domaine : le « Systems Engineering Handbook » de l'INCOSE et « Découvrir et Comprendre l'Ingénierie Système » de l'AFIS. La Nasa a aussi édité son propre manuel : le « NASA Systems Engineering Handbook ».

Ces différents documents de références sont cohérents entre eux à haut niveau d'abstraction. Dans le cadre du projet LEA, l'ensemble des phases décrites dans la Figure I.1 ne nous intéresse pas. En effet, LEA est un véhicule expérimental qui ne volera qu'une seule fois, ainsi seules les trois premières phases : Définition du système, Réalisation des constituants et Intégration du Système sont applicables dans le cas du système LEA (l'ensemble du projet LEA est détaillé dans le chapitre V). De plus, l'objectif des travaux de recherche menés dans le cadre du projet LEA est d'intégrer les études de sûreté de fonctionnement au plus tôt afin de réduire l'impact de celles-ci sur le temps de développement et les coûts du projet. De ce fait, nous restreignons donc le cadre de notre recherche à la première phase : Définition du système, bien que notre participation au projet se poursuive dans les phases suivantes par un travail d'ingénierie pure. Parmi les activités qui sont proposées, dans les documents précédents, pour définir le processus d'IS de définition du système, nous trouvons :

- « Définition des exigences », parfois isolée de « Définition des exigences des parties prenantes du projet ».
- « Description de l'architecture du système », parfois scindée en architecture fonctionnelle et architecture organique.
- « Validation » des différentes entités décrites auparavant : exigences, fonctions, composants…

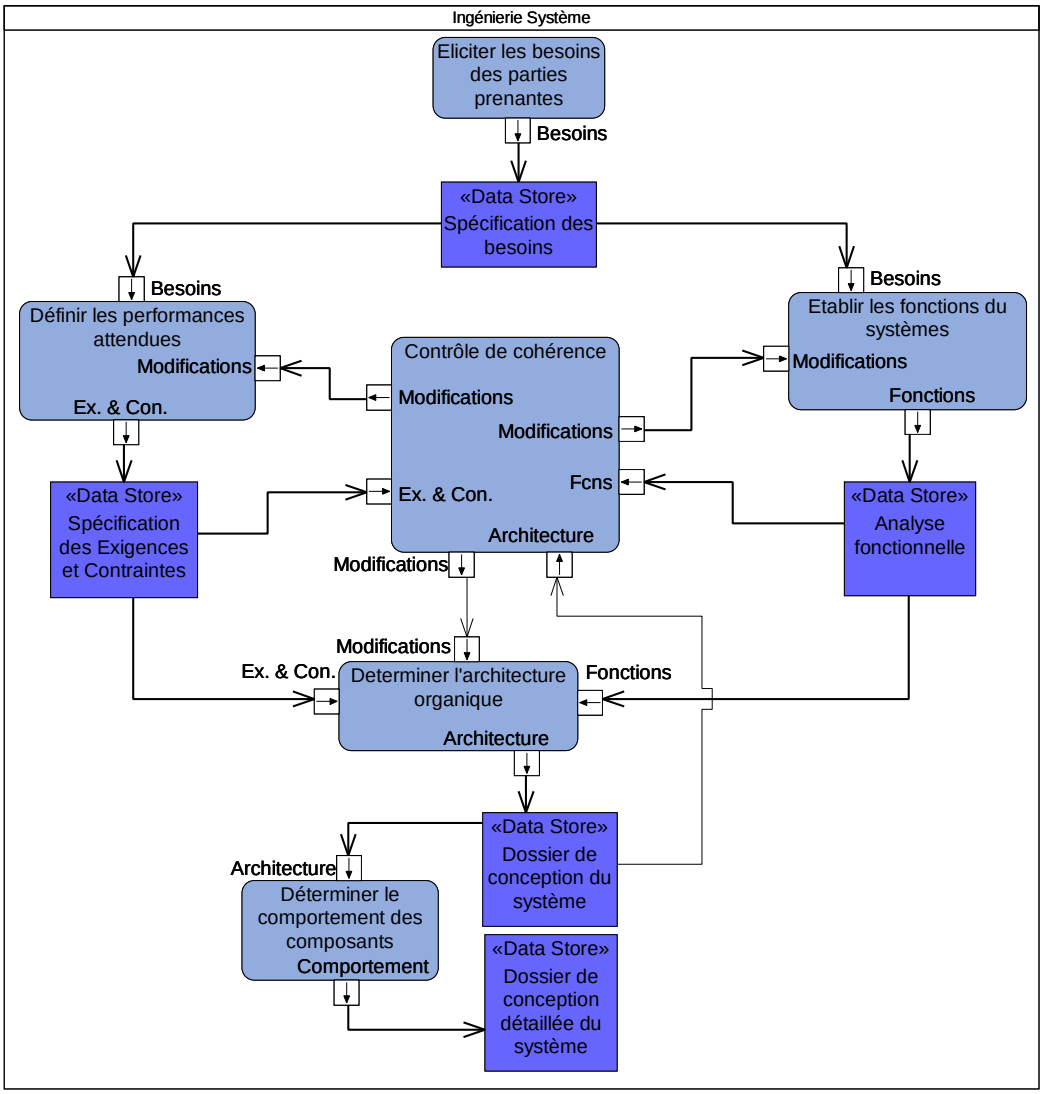


Figure I.2 : Processus d'IS de définition du système

Afin de donner un cadre à notre étude d'IS, le processus d'IS retenu est celui présenté en Figure I.2. Il est proche du « System Engineering Process » proposé par la norme IEEE 1220. La différence vient de la simplification de celui-ci, notamment concernant les activités de validation. Pour chaque activité, on considère que la validation de l'activité elle-même est réalisée dans un sous-processus. L'activité de contrôle de cohérence de l'ensemble du système est une activité à part entière, telle que c'est le cas dans [IEEE 1220]. L'autre point de différentiation avec le processus IEEE 1220 est l'apparition d'une activité : « Elicitation du besoin » qui est présente dans les normes et manuels étudiés, mais rarement représentée en tant qu'activité à part entière. La Figure I.2 présente le processus d'IS que nous utiliserons dans la suite de la thèse. Les activités sont représentées sous forme de rectangle à coins arrondis. Les rectangles à coins droits représentent les documents résultant des différentes activités. Sur la Figure I.2, on se place dans le cas d'un processus d'ingénierie système à base de documents. Nous verrons par la suite en quoi l'ingénierie système à base de modèles diffère de l'ingénierie système documents.

Dans les niveaux inférieurs de détails, chaque standard propose des processus qui lui sont spécifiques. Bien que l'ensemble des standards cités précédemment propose des solutions intéressantes, la norme IEEE 1220, de par sa spécialisation sur la phase qui nous préoccupe particulièrement sera notre référence principale.

3. Ingénierie système à base de modèles (ISBM)

L'ingénierie système à base de documents est l'approche traditionnelle pour l'ingénierie des systèmes physiques. Certains domaines l'ont abandonnée au profit d'une approche d'ISBM. Les outils de modélisation assistés par ordinateur sont utilisés pour l'ingénierie mécanique et l'ingénierie électrique depuis les années 80. L'ingénierie logicielle a quant à elle largement adopté et contribué à l'ISBM au cours des années 90. C'est donc naturellement que l'ISBM se répand maintenant au sein des communautés d'ingénieurs système apportant de nombreux avantages qui devraient faire de l'ISBM l'approche de référence en matière de développement de système dans les années à venir [INCOSE 2007], [Friedenthal et al. 2007].

L'ISBM est l'utilisation de langages de modélisation comme supports aux activités de spécification, conception, analyse, vérification et validation des systèmes, débutant de la phase de conception préliminaire et se poursuivant à travers le développement jusqu'aux dernières phases de vie du système [INCOSE 2007]. L'objectif de l'ISBM est de faire progresser l'IS dans de nombreux domaines comme la communication, la précision de l'analyse, l'intégration des résultats ou la réutilisation des connaissances produites. Ces critères sont identiques aux buts de l'IS (cf Chapitre I, II.1). L'ISBM vise donc à faciliter la réalisation des objectifs de l'IS. Les bénéfices de l'utilisation de l'ISBM sont présentés par [Friedenthal et al. 2008] :

- Amélioration de la communication entre domaines d'expertise
- Adoption de vues sur le système.
- Capture des spécifications et données de conception dans un format normalisé et neutre.
- Amélioration de la qualité de l'étude.
- Exigences plus complètes, expressives et vérifiables.
- Meilleure traçabilité entre exigences, conception, analyses et tests.
- Cohérence des spécifications.
- Amélioration de la productivité.
- Évaluation de l'impact des modifications facilitée.
- Réutilisation de modèles existants dans plusieurs projets.
- Réduction des erreurs et tests d'intégration rapide (par respect de la syntaxe de modélisation).
- Génération automatique de documents.

Afin d'aborder en quoi consiste l'ISBM, il nous faut d'abord définir deux termes qui sont au cœur de l'approche à base de modèles : « modèle » et « vue ». [David 2009] reprenant [Friedenthal et al. 2008] propose les définitions suivantes :

Modèle : Représentation d'un ou plusieurs concepts pouvant être réalisés dans le monde physique. Ils sont une abstraction qui ne contient pas tous les détails de l'entité modélisée, dans certaines vues. Les modèles sont représentés sous de nombreuses formes : graphiques, mathématiques, représentations logiques...

Vue : Présentation partielle des connaissances sur l'élément modélisé. Les vues permettent de partitionner l'étude d'un système afin de proposer à un analyste le sous-ensemble d'information nécessaire et suffisant à sa prise de décision. L'usage des vues permet de découper le problème global d'analyse ou de conception du système, en sous-problèmes plus facilement analysables.

Pour être mise en place, l'ISBM doit être supportée par un langage de modélisation. Un langage de modélisation est avant tout un langage tel que défini précédemment, cependant certaines de ses composantes prennent des formes particulières. L'alphabet d'un langage de modélisation est un recueil d'entités graphiques, textuelles ou autres. Les phrases créées à l'aide de cet alphabet doivent permettre de représenter les informations dans les différentes vues que couvre le modèle. Enfin, l'ensemble des informations représentées dans chaque vue, en utilisant l'alphabet précédent, constitue le modèle. Dans le cadre d'un langage de modélisation graphique comme SysML, les entités graphiques constituent l'alphabet et les diagrammes, l'ensemble des phrases et vues. La syntaxe d'un tel langage consiste à décrire comment l'alphabet peut et doit être utilisé pour constituer des phrases. Le respect d'une syntaxe permet d'assurer une cohérence dans la représentation que l'on fait d'un concept. La syntaxe guide le modélisateur. À la syntaxe, il faut associer une sémantique qui doit faire le lien entre le signifiant et le signifié, ici l'objet du modèle (diagramme, entité,…) et le concept original. Au final, le modèle est le résultat de la réification des concepts manipulés et résultants des activités d'IS.

En ISBM, le modèle est d'abord employé pour spécifier et concevoir le système. Le modèle met en avant l'interconnexion des éléments (exigences, fonctions, composants,…) issus de chaque activité. L'utilisation de ce modèle unique permet une cohérence et une traçabilité des relations accrues. De même, si le langage de modélisation dispose d'outils logiciels adaptés, l'utilisation de l'ISBM offre de nombreux avantages. Le modèle peut alors servir de support de transition pour les développements particuliers utilisant des langages spécifique à un domaine (DSL : Domain Specific Language). Il peut servir également à la remontée des performances évaluée à l'aide de ces DSL. Enfin, ce modèle sert à la génération de documents contractuels ou normés. Le modèle occupe alors une place centrale au sein du développement des systèmes, et permet à tous les domaines d'expertise de disposer d'une référence commune. La Figure I.3 inspirée de [David 2009] montre le modèle comme le point de rencontre entre les connaissances, les études utilisant des DSL et la production documentaire. Il sert aussi bien à préparer les études particulières qu'à répercuter leurs résultats et conclusions.

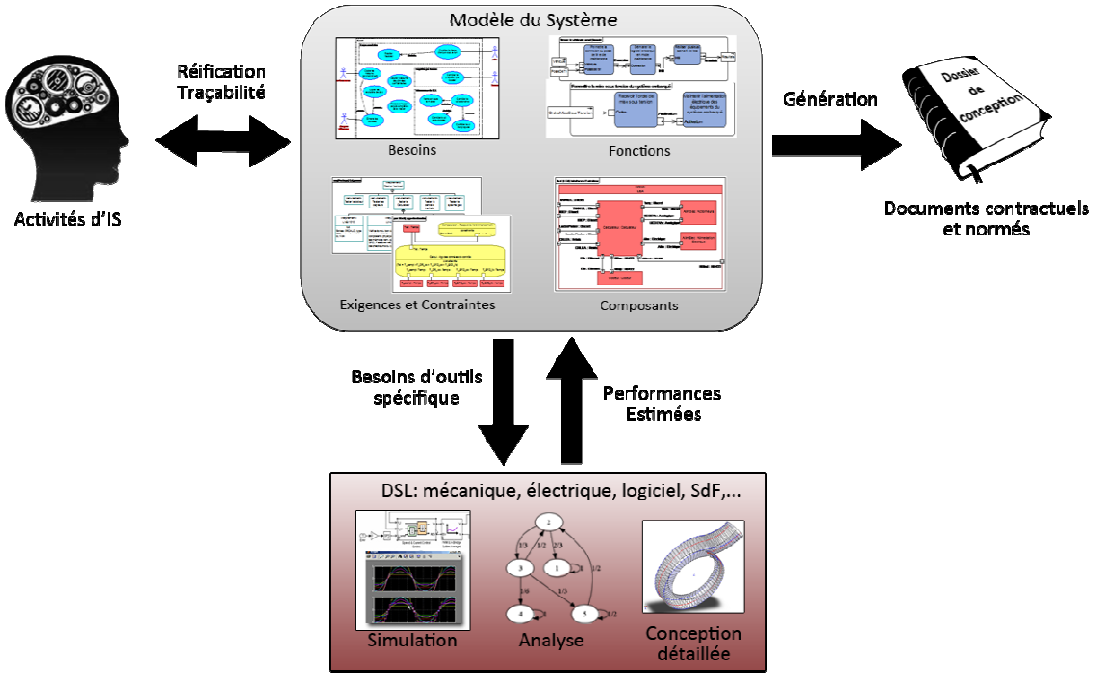


Figure I.3 Le modèle du système comme pivot du projet

4. Quelques langages de modélisation

4.1. Simulink, modélisation, simulation et génération de code

Simulink est un environnement de conception et de simulation par modélisation (Model-Based Design), destiné aux systèmes dynamiques. Il fournit un environnement graphique interactif et un ensemble personnalisable de bibliothèques de blocs qui permettent de concevoir, simuler, mettre en œuvre et tester différents systèmes (en particulier dans le domaine des communications, du contrôle-commande, du traitement des signaux et des traitements vidéo et image). Simulink est intégré avec MATLAB, ce qui apporte une gamme d'outils pour le développement d'algorithmes. L'approche Model Based Design mise en avant par la société MathWorks [MathWorks], qui développe et maintient Simulink, est présentée dans la Figure I.4.

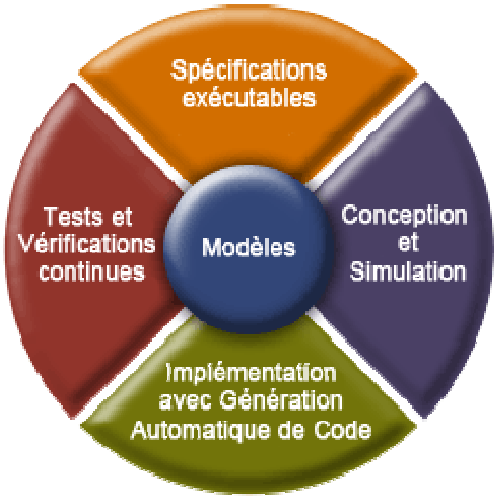


Figure I.4 : Principe du Model Based Design selon Mathworks

Nous pouvons observer 4 activités principales rendues possibles par le modèle central. Cependant, seule l'activité de Spécification et dans une moindre mesure l'activité de Conception, correspondent à ce que nous attendons d'un langage système dans le cadre de la thèse. Nous allons donc succinctement présenter la syntaxe offerte par Simulink pour ces activités.

Simulink est un logiciel qui procure un environnement de modélisation basée sur des schémas-blocs. D'un point de vue modélisation, Simulink met à disposition :

- Des blocs génériques. Ces blocs peuvent être imbriqués les uns dans les autres pour définir une décomposition des spécifications et de la conception selon plusieurs niveaux de détails.
- Des ports pour modéliser les entrées et sorties de chaque bloc.
- Des lignes pour connecter les ports entre eux.
- Des bibliothèques de blocs spécifiques de bas niveau (non décomposables). Ces blocs représentent des composants ou éléments fonctionnels unitaires : addition, « ou » logique, multiplexeur, …

L'alphabet du langage Simulink se résume donc principalement aux trois symboles : bloc générique, port et ligne. Il existe certains éléments syntaxiques dont l'utilisation est étroitement liée à des besoins sémantiques particuliers : représentation des exigences et de la satisfaction des exigences par un mécanisme d'association des exigences aux blocs, représentation de la logique d'état interne des composants par un module spécifique : Stateflow, …. Cependant, la syntaxe de Simulink reste minimaliste et cela implique une sémantique délicate à définir puisque tous les concepts de l'ingénierie système doivent reposer sur un total de 3 éléments syntaxiques. Ainsi Simulink n'est pas qualifié pour être le modèle central du système tel que ce concept a été présenté au §3.

En contrepartie, l'abondance d'outils associés aux modèles Simulink : simulation, génération de code, tests structurels et fonctionnels, validation de la modélisation, bibliothèque de composants spécialisés, aide à la rédaction de dossier de certification,… font de Simulink un DSL important avec lequel le modèle système doit pouvoir interagir. Un exemple d'utilisation standard de Simulink pourra être trouvé dans [Ishaque 2011] où un modèle Simulink est créé pour simuler un système photovoltaïque.

4.2. AADL, modélisation des systèmes embarqués et temps réels

Historiquement, AADL a été développé pour répondre aux besoins spécifiques des systèmes embarqués temps réel tels que les systèmes avioniques. En particulier, le langage peut décrire les mécanismes standards de flux de contrôles et de données utilisés dans les systèmes avioniques, et des aspects non-fonctionnels importants tels que les exigences temporelles, le partitionnement temporel et spatial. Le nom d'AADL signifie *Architecture Analysis & Design Language (langage d'analyse et de conception d'architectures).* La description d'une architecture en AADL consiste donc en la description de ses composants et leur composition sous forme arborescente. AADL est d'abord un langage textuel formel dont la première version est parue en 2004, [SAE 2004]. Plusieurs annexes au standard sont publiées en 2006 ([SAE 2006]) définissant une notation graphique associée, un méta-modèle d'AADL pour l'échange de fichier et une notation des profils non-fonctionnels. Enfin, en 2009, le standard du langage textuel a été révisé et mis à jour [SAE 2009], pour harmonisation avec les nouvelles notations et fonctionnalités apportées par les annexes.

Chaque composant est décrit en AADL en deux phases. La première, le *type*, correspond à l'interface fonctionnelle du composant, ce qui est visible des autres composants. La seconde, l'*implémentation*, décrit le contenu du composant (sous-composants, propriétés, connexions, etc.). L'intérêt de scinder une description de composant en type et implémentation est de séparer deux points de vue. Décrire le type revient à spécifier l'interface du composant, exprimer à quoi il doit ressembler vu de l'extérieur : c'est le principe de la vue « boite noire ». En revanche, l'implémentation représente l'intérieur du composant : c'est le principe de la vue « boite blanche ». Dans la pratique, la description du type et de l'implémentation pourront être faites par des personnes différentes, chacune ayant en charge une étape dans le raffinement de la description de l'architecture.

Chaque composant appartient à un type. Ces types sont prédéfinis et se décomposent en types matériels, types logiciels ou types composites (i.e. commun au logiciel et au matériel) :

- mémoire (*memory*) ;
- périphérique (*device*) ;
- processeur (*processor*) ;
- bus ;
- donnée (*data*) ;
- sous-programme (*subprogram*) ;
- tâches *(thread)* ;
- groupe de tâches (*thread group*) ;
- processus (*process*) ;
- système (*system*).

L'ensemble de ces types représente l'alphabet du langage AADL, et chacun de ces symboles possède sa propre notation graphique (Figure I.5).

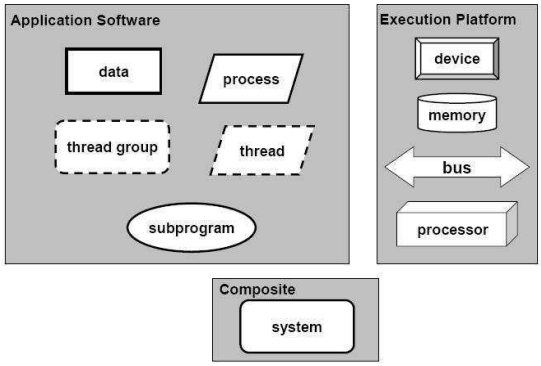


Figure I.5 : Représentation graphique des différentes entités du langage AADL

À chaque entité peuvent être associées des propriétés valuées permettant de caractériser le composant. Certaines propriétés sont prédéfinies, c'est-à-dire qu'elles sont identifiées par un nom, un type et la liste des catégories de composants sur lesquelles elles s'appliquent. Par exemple, les tâches *(thread)* disposent de propriétés temps-réel telles que la période, l'échéance ou la durée d'exécution. De nouvelles propriétés, et de nouveaux types de propriétés peuvent aussi être définis par l'utilisateur et associés à tout ou partie des catégories de composants. Ce mécanisme de propriétés est un point fort d'AADL en matière d'extensibilité. Grâce à lui, toute notion spécifique au besoin de l'utilisateur peut être prise en compte dans sa description.

L'ensemble de la syntaxe proposé par AADL permet une représentation précise de l'architecture organique et logicielle de systèmes embarqués. De nombreux travaux ont recours au langage AADL principalement dans le cadre d'application pour l'aéronautique. L'autre grande force d'AADL est de posséder d'ores et déjà des outils logiciels qui lui sont dédiés s'adaptant aux forces du langage : la modélisation des propriétés temporelles et l'interconnexion entre composants physiques et

logiciels. Cheddar [Singhoff 2007] en est un parfait exemple. Cheddar est un outil de simulation permettant de calculer différents critères de performance (contraintes temporelles, dimensionnement de ressources). L'outil permet, entre autres, de tester le respect des contraintes temporelles d'un jeu de tâches modélisant un système temps réel. En plus de Cheddar, il existe plusieurs éditeurs de modèle AADL [Topcased][Stood] et un compilateur AADL développé par l'équipe S3/Télécom-Paris-Tech : Ocarina [Lasnier 2009].

Dans le cadre du projet LEA, AADL apporte ses capacités à décrire un système embarqué et particulièrement la dualité logiciel/matériel, cependant la problématique que nous étudions dans la thèse est plus large que cela. En effet, bien que AADL propose une modélisation organique des composants logiciels et matériels, et même une modélisation des comportements défaillants [Feiler & Rugina 2007][SAE 2006], il est cependant impossible de réaliser l'ensemble des activités définies dans notre processus d'IS (Figure I.2). Le langage AADL ne peut donc pas être utilisé comme langage central de modélisation système tel que nous le décrivions (Figure I.3). Finalement, le langage AADL sera plus facilement traité comme un DSL pour l'analyse des contraintes temps réels du système.

4.3. SysML, le langage de l'ingénierie système

SysML est un langage de modélisation spécifique au domaine de l'ingénierie système [OMG 2012]. Il permet la spécification, l'analyse, la conception, la vérification et la validation de nombreux systèmes et systèmes-de-systèmes [Desroches 2010]. SysML se définit comme une extension d'un sous-ensemble d'UML 2.0 (Unified Modeling Language) via l'utilisation du mécanisme de profils définis par [OMG 2011a]. Historiquement, UML a tenté en 2003 (lors du passage de UML à UML 2.0) de s'ouvrir à l'IS en proposant de s'appliquer non plus aux « systèmes logiciels », mais aux systèmes en général. Néanmoins, UML n'a pas rencontré le succès escompté auprès de la communauté des ingénieurs système, principalement à cause de ses notations trop orientées logiciel et d'un manque d'expressivité pour des problématiques spécifiques aux systèmes physiques [Hause 2006], [Willard 2007]. C'est pourquoi l'INCOSE et l'OMG ont développé le langage SysML adopté en 2007 [OMG 2007] et dont la dernière mise à jour (version 1.3) date d'avril 2012 [OMG 2012]. SysML est donc construit comme un sous-ensemble des notations UML, augmenté de nouveaux artefacts destinés à traiter des spécificités de l'IS, entre autres :

- La syntaxe de SysML est plus riche et flexible : SysML impose moins de restrictions liées à la vision d'UML centrée sur le logiciel, et ajoute deux nouveaux types de diagrammes. Le premier peut être utilisé pour la gestion des exigences, le deuxième peut être utilisé pour l'analyse des performances et l'analyse quantitative. Grâce à ces améliorations, SysML est capable de modéliser une large gamme de systèmes, incluant tant du matériel, que du logiciel, de l'information, des processus, du personnel, ou des équipements.
- SysML fournit des tableaux d'allocations flexibles qui supportent les différentes relations possibles entre les exigences, les éléments fonctionnels et les éléments structurels. Cette fonctionnalité permet de réaliser la vérification et la validation du système.

SysML est donc constitué d'une partie commune avec UML, appelée UML4SysML, et d'une partie spécifique correspondant au profil SysML, comme illustrée par la Figure I.6.

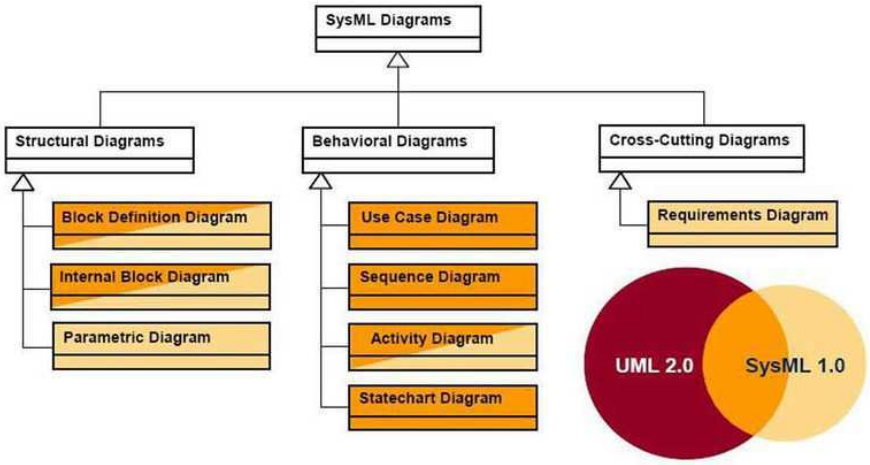


Figure I.6 : Organisation des diagrammes SysML

Les Requirements Diagrams et les Parametric Diagrams sont des nouveautés de SysML alors que les Block Definition Diagrams, les Internal Block Diagrams et les Activity Diagrams sont des variations de diagrammes UML. Le Tableau I.1 ci-dessous résume l'ensemble des diagrammes proposés par le standard SysML.

Tableau I.1 : Présentation des types de diagrammes SysML

Diagramme	Description
Use Case Diagram (UCD)	Il modélise les fonctionnalités que le système doit fournir. Le cas d'utilisation est souvent utilisé comme besoin fonctionnel unitaire pour la description et la recette du système.
Sequence diagram (SD)	Le diagramme de séquence modélise la chronologie des interactions entre les entités.
Activity Diagram (AD)	Le diagramme d'activité modélise les activités et les flux entre ces activités.
State Machine (STM) Diagram	Il représente les différents états que peut prendre un élément ou une opération ainsi que ses réactions aux évènements extérieurs.
Block Definition Diagram (BDD	Le diagramme de Bloc en SysML donne une représentation statique des entités du système, de leurs propriétés et de leurs.
Internal Block Diagram (IBD)	Le diagramme interne de bloc SysML donne une représentation « Boîte blanche » qui matérialise les imbrications des parties et leurs interconnexions par les ports.
Package diagram	Le diagramme de Package montre l'organisation générale du modèle. Il sert en plus à organiser les différentes vues du système.
Parametric Diagram (ParD)	Nouveau dans SysML ce diagramme modélise les paramètres physiques et performantiels du système et les relations mathématiques et logiques entre ces paramètres et les propriétés des éléments structurels.
Requirement Diagram (ReqD)	Le diagramme de spécification est nouveau dans SysML et il permet de collecter et d'organiser toutes les exigences textuelles du système.
Allocation tables	Les tables d'allocation sont de simples tableaux et non des diagrammes qui récapitulent les relations entre entités afin de faciliter le suivi de projet.

La syntaxe de SysML décrite dans le standard s'accorde avec les besoins de l'IS décrit au §I. Des éléments SysML sont disponibles pour représenter l'ensemble des concepts manipulés en IS. De plus, l'ensemble des outils d'allocation et de contrôle de cohérence en fait un candidat idéal pour une approche d'ISBM. C'est pourquoi nous entérinons dans cette thèse les choix faits dans [David 2009] d'utiliser SysML comme langage central d'une méthodologie d'ISBM.

Les travaux de [David 2009] sont principalement définis pour l'analyse de l'architecture organique d'un système, or la syntaxe SysML est suffisamment détaillée pour que les différentes sémantiques proposées par la littérature ne différent que peu pour la modélisation des composants. Ceci permettait aux travaux de [David 2009] de ne pas imposer une sémantique particulière, s'appuyant sur le tronc commun entre elles. Cependant, les travaux de la thèse traitent de l'ensemble des activités d'IS définies en §I. Dans ce cas, plusieurs sémantiques basées sur la syntaxe SysML existent : Harmony d'IBM Telelogic ([Douglass 2005], [Hoffmann 2006]), OOSEM (Object-Oriented, Systems Engineering Method, [Lykins et al. 2000], [Friedenthal et al. 2008]) ou RUP SE (Rational Unified Process for Systems Engineering [Kruchten 2003], [Cantor 2003]). Il est alors impossible de ne pas imposer un cadre sémantique à nos travaux étant donné que deux symboles SysML peuvent avoir des sens différents selon la sémantique retenue. Nous nous efforcerons donc de détailler notre solution sémantique, nécessaire au bon fonctionnement de la méthodologie MéDISIS étendue par nos travaux en définissant nos propres règles de modélisation dans le chapitre II. Malgré tout, l'ensemble des travaux de la thèse reste vrai quelque soit la sémantique, à condition d'adapter le raisonnement. Un exemple est donné dans la dernière partie du chapitre III pour illustrer ce propos.

III. Sûreté de fonctionnement

Cette section introduit les concepts de base de la sûreté de fonctionnement tel qu'ils sont définis dans [Laprie et al 1996], [Laprie 2004] et [Villemeur 1988], travaux dans lesquels les lecteurs trouveront une vue plus détaillée de l'ensemble du domaine. L'accent sera ensuite mis sur l'intérêt de connecter les concepts de la sûreté de fonctionnement avec les modèles d'IS.

1. Concepts de la sûreté de fonctionnement

La sûreté de fonctionnement (SdF) peut être définie comme la « science des défaillances » [Villlemeur 88]. Elle inclut ainsi leur connaissance, leur évaluation, leur prévision, leur mesure et leur maîtrise. Au sens large, elle est l'aptitude d'une entité à satisfaire une ou plusieurs fonctions requises dans des conditions données.

Selon la ou les applications auxquelles le système est destiné, l'accent peut être mis sur différentes attributs de la sûreté de fonctionnement, ce qui revient à dire que la sûreté de fonctionnement peut être vue selon des aspects différents parfois complémentaires ou antagonistes, qui permettent de définir ses attributs :

- Le fait d'être prêt à l'utilisation conduit à la **disponibilité** ;
- La continuité du service conduit à la **fiabilité** ;
- L'absence de conséquence catastrophique pour l'environnement conduit à la **sécurité innocuité** ;

- L’absence de divulgations non-autorisées de l’information conduit à la **confidentialité** ;
- L’absence d’altérations inappropriées de l’information conduit à l’**intégrité** ;
- L’aptitude aux réparations et aux évolutions conduit à la **maintenabilité**.

Un service correct est délivré par un système lorsqu’il accomplit sa fonction. Une défaillance est la cessation de l’aptitude d’une entité à accomplir une fonction requise. Le service est considéré défaillant lorsqu’il ne respecte plus la spécification fonctionnelle. Les défaillances observées d’un même service peuvent prendre plusieurs formes qui sont nommées les Modes de Défaillance (MdD).

2. Exploitation de l’ISBM pour la SdF

Le défi du développement de systèmes complexes à fortes contraintes de SdF, comme les systèmes de sécurité ou les systèmes en interaction critique avec l’Homme ou son environnement, est de réussir à introduire efficacement les méthodes d’analyse de SdF dans le processus de conception [Guillerm 2011]. Pour la réalisation de ce type de système, nous avons conclu qu’il était nécessaire d’employer un processus d’IS utilisant des modèles. Nous avons préconisé l’emploi du langage SysML. Nous étudions dans les paragraphes suivants, la mise en place de ces analyses au cœur de l’ISBM. Nous définissons notamment le cadre méthodologique MéDISIS : Méthode D’Intégration des analyses Sûreté de fonctionnement au processus d’Ingénierie Système, qui constitue la base des travaux de la thèse par la suite.

2.1. Travaux antérieurs à SysML

Le projet décrit dans [Bondavalli et al. 1999a], [Bondavalli et al. 2001] propose une méthodologie pour faciliter les études SdF des systèmes à logiciel. Pour cela, le langage UML est utilisé comme modèle central afin de bénéficier de sa souplesse et son expressivité pour tous les acteurs d’un projet. Ensuite, les participants du projet ont défini un ensemble de transformations automatiques vers des formalismes et outils d’analyses SdF dédiés. Les modèles d’entrée UML expriment le comportement global du système, incluant les comportements défaillants. Afin de rendre possible les transformations de modèle, les auteurs définissent des restrictions d’utilisation d’UML et introduisent leurs propres stéréotypes relatifs à la SdF. [Bondavalli et al. 1999b] présentent une transformation d’UML en Réseaux de Petri (RdP) temporisés. [Bernardi et al. 2002] et [Merseguer et al. 2002] proposent l’exploitation combinée des statemachines (STM) et SD pour obtenir des RdP composés. [Lopez-Grao et al. 2004] poursuivent ces travaux par la génération de Réseaux de Petri Stochastiques Généralisés (RdPSG). Des travaux comme [Pai & Dugan 2002] ont développé d’autres types de transformations comme la création d’Arbre de défaillance dynamique à partir de diagrammes de classe et de STM.

Ces modèles ne reflètent pas uniquement les aspects fonctionnels du système, mais s’intéressent aussi aux mécanismes de la SdF. C’est pourquoi leur création nécessite l’emploi de stéréotypes d’entités UML capables de modéliser les défaillances et leur propagation. Les principaux désavantages de ces approches sont qu’elles restent spécifiques au traitement de systèmes logiciels et que l’utilisation de stéréotypes pour la description des propriétés dysfonctionnelles rend leurs réemplois standards sur différents outils UML impossible. Toutefois, ces travaux ont tracé de nombreuses correspondances entre les modèles UML et des formalismes permettant la réalisation

d’analyses de performances et de SdF variées, comme le calcul de disponibilité, la prévision de fiabilité, la vérification de caractéristiques temporelles, la possibilité de reconfiguration dans des états sûrs.

2.2. MéDISIS, utilisation de SysML comme modèle central

La méthodologie MéDISIS (Méthodologie D’Intégration des analyses de Sdf à l’Ingénierie Système) proposée dans [David 2009] a pour but de définir un enchaînement de traitements de l’information et des connaissances, pour l’étude de systèmes complexes et multi-technologiques (Figure I.7). MéDISIS a aussi pour vocation de réutiliser les outils des industriels tout en proposant une méthode ne ciblant aucun d’entre eux en particulier. Les objectifs d’une telle méthodologie sont :

- Faciliter la transmission de connaissances entre équipes de métiers différents.
- Accélérer la réalisation des études de SdF.
- Organiser l’exploitation commune des connaissances sous forme de modèles.
- Permettre la réutilisation des connaissances entre projets.
- Identifier les besoins d’analyse et réaliser le suivi de leurs résultats.
- Améliorer la cohérence et la qualité des analyses SdF.

MéDISIS repose sur quelques hypothèses de travail. Un modèle SysML du système issu des activités d’ISBM est le support de la méthodologie. MéDISIS propose ensuite des procédures permettant de faciliter les analyses de SdF à partir des informations du modèle. La méthodologie prévue pour être utilisable à divers niveaux de détails du système, être itérative (permettre la descente dans ces niveaux de détails) et répétable. Les travaux décrits dans [David 2009] introduisent deux processus de génération d’analyse de SdF : génération d’AMDEC composants et de modèle Altarica DF à partir de l’analyse d’un modèle SysML. Les connaissances collectées au cours de l’étude concernant les composants, sous-systèmes et systèmes devant être réutilisables pour d’autres projets afin de remplir le rôle des approches modernes d’IS, la Base de données des Comportements Dysfonctionnels (BCD) a été définie comme un élément central de MéDISIS. Cette BCD regroupe les connaissances collectées sur les mécanismes de défaillances des entités composant le système analysé.

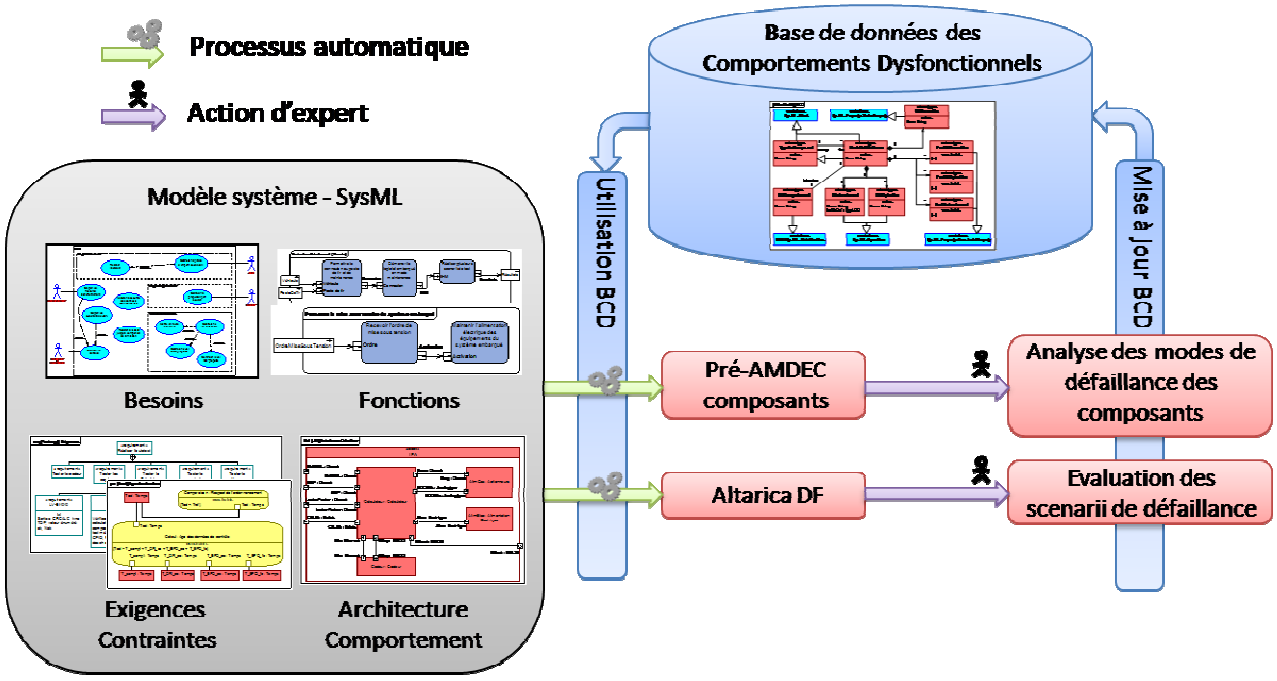


Figure I.7 : La méthodologie MéDISIS

2.2.1. Génération d'AMDEC composant

L'utilisation de MéDISIS pour la génération d'AMDEC à partir des premières phases d'IS permet de réaliser une détection systématique des risques pesant sur le système très tôt dans la conception. Ces risques sont analysés et classifiés afin de diriger la suite de l'étude. Ainsi, l'automatisation et la réalisation d'AMDEC à partir des modèles système sont des leviers importants pour la réussite de l'intégration des AMDEC à l'ISBM. En effet, par l'analyse de modèles purement fonctionnels exprimés en SysML, il devient possible d'automatiser des étapes comme la recherche de composants et de guider l'analyse de l'expert lors de la rédaction de l'AMDEC finale.

Lors d'une AMDEC, le processus de prise de décision est mené pour établir des relations de cause à effet à partir de vues structurelles et fonctionnelles du système pour générer un tableau AMDEC. L'étude AMDEC passe par les étapes suivantes :

1. Recensement des composants à étudier.
2. Pour chaque composant, rechercher les MdD.
3. Pour chaque MdD, rechercher les causes possibles.
4. Pour chaque couple MdD/causes possible, rechercher les effets possibles sur les composants périphériques.
5. Pour chaque effet, réaliser la cotation du risque (cf Annexe 1), proposer des moyens de détection et spécifier les actions à mener pour diminuer le risque.

Le processus MéDISIS effectue ces actions en analysant le modèle SysML, en conjonction avec l'utilisation de la BCD. Cette BCD permet de faire appel au Retour d'EXpérience (REX) sur des systèmes ou composants similaires déjà utilisés. Ce REX est primordial pour les AMDEC où seuls les jugements d'experts s'expriment. Ainsi la recherche des MdD et la cotation du risque se font avec le support de la BCD puisqu'il s'agit de données dysfonctionnelles qui ne sont pas modélisées au cours des activités d'IS. L'action d'un expert pour finaliser les modèles générés sera toujours

nécessaire. Cette action permet de compenser l'éventuelle absence de données dysfonctionnelles utiles dans la BCD. Elle permet aussi de manière générale de filtrer l'information générée pour cadrer avec les pratiques de SdF souhaitée. Dans le cadre de MéDISIS, la connexion à la BCD est double. En effet, les résultats de l'AMDEC validée par les experts sont également réintroduits dans la BCD pour préciser et enrichir ses connaissances. Cette phase est découpée en deux étapes : la création d'un rapport AMDEC préliminaire destiné à soutenir les experts et éliminer la lourdeur de l'étude, et une finalisation manuelle de l'AMDEC par les experts dont le jugement et la prise de décision finale ne peuvent être automatisés. Ensuite, les apports de l'expert sont mémorisés dans la BCD participant ainsi à l'amélioration constante de la qualité de la pré-AMDEC rédigée.

2.2.2. Génération de modèle Altarica DF

Le second processus MéDISIS consiste à construire un modèle Altarica DF permettant une analyse formelle de la SdF du système. Ce modèle reprend traditionnellement la définition de la structure et des comportements fonctionnels et dysfonctionnels des composants. Il redéfinit donc une partie déjà couverte par le modèle SysML. Il est alors intéressant d'assister la création de ce modèle en traduisant le modèle système en SysML. La génération permet d'accélérer l'analyse de SdF et d'assurer une plus grande cohérence entre les modèles manipulés au cours du projet. La construction du modèle AltaRica est effectuée en deux étapes. Dans un premier temps, le modèle fonctionnel est constitué par traduction des concepts SysML. Ensuite, la partie dysfonctionnelle est ajoutée sur le modèle précédemment constitué dans lequel apparaissent les composants physiques de l'architecture et leur comportement nominal. Cette seconde étape est réalisée par complétion par l'expert et grâce à la BCD. La mise au point de l'algorithme de traduction de SysML vers Altarica DF est initiée par analyse des concordances des concepts manipulés dans chaque langage. Le fait que chacun d'entre eux soit un langage orienté objet facilite ce rapprochement. Ensuite, pour chaque élément SysML à prendre en compte, l'expression correspondante en AltaRica est associée moyennant des restrictions ou adaptations nécessaires afin de respecter la syntaxe du langage.

L'étape d'adjonction du comportement dysfonctionnel peut ensuite être réalisée par utilisation de la BCD. Elle permet de faire appel à des éléments de modèles dysfonctionnels déjà connus, qui viennent se greffer à la partie fonctionnelle décrite dans le modèle système. La BCD peut alors être vue comme une bibliothèque de modèles dysfonctionnels. De même que pour la génération d'AMDEC, l'action d'expert permet de compenser une absence éventuelle de données utiles dans la BCD et l'expert dispose alors d'un modèle fonctionnel cohérent avec celui défini par les activités d'IS pour le guider dans son étude. Les comportements dysfonctionnels définis par l'expert sont alors pérennisés par insertion dans la BCD.

2.2.3. La Base de données des Comportements Dysfonctionnels (BCD)

L'usage de la BCD centralise la gestion des connaissances créées et réutilisées tout au long de MéDISIS. Afin de faciliter l'interconnexion des algorithmes de génération de modèle et la BCD, celle-ci est définie à l'aide d'un méta-modèle basé sur SysML. Ainsi le modèle du système en SysML et la BCD forment le socle central de la méthodologie MéDISIS. Le méta-modèle décrit par [David 2009] dispose de l'ensemble des concepts nécessaire à une utilisation pour la génération d'AMDEC et de modèle Altarica DF ainsi que la pérennisation des données dysfonctionnelles

apportées par les experts dans le cadre de la réalisation de ces types d'études SdF. La dernière version du méta-modèle de la BCD a été présentée dans [Cressent et al. 2012b] où sont exposés les travaux que nous détaillons dans le chapitre IV. Les informations dysfonctionnelles qui y sont stockées sont réifiées en utilisant le langage SysML afin de maintenir une interaction aisée avec le modèle système.

La BCD repose sur un couple d'entités associées : composant et MdD. Le composant est une copie de celui utilisé par les activités d'ingénierie système. Les modes de défaillance d'un composant seront quant à eux décrits plus en détail, à travers notamment de 3 axes : les mécanismes de déclenchement de la défaillance, la description des fonctionnements défaillants du composant et la logique régissant l'exécution des fonctionnements normaux ou défaillants. Ces moyens de modélisation, originellement conçus dans la perspective des processus de rédaction d'AMDEC et de génération de modèle Altarica DF, s'avèrent permettre des utilisations plus larges. En effet, la structure de la BCD permet à travers une variété importante de moyens de modélisation (empruntés à la syntaxe SysML sur laquelle elle se base) la représentation des données utiles à de nombreuses études de SdF. Au cours de la thèse, des travaux annexes ont notamment été menés pour étendre MéDISIS à la génération de modèles AADL [Cressent et al. 2010] [Cressent et al. 2011c] et à la génération de modèles Simulink [Cressent et al. 2011a] [Cressent et al. 2011c]. L'ensemble de ces travaux tire encore parti de la BCD.

IV. Les défis de l'utilisation de COTS

Le terme COTS pour « Component off the shelf » (en français : composant sur étagère) regroupe plusieurs type de composants et plusieurs niveaux de détails de description. Les deux principaux concepts qui sont signifiés par ce terme peuvent être définis ainsi :

- ***COTS logiciel :*** Historiquement ce qui fut en premier désigné sous le terme COTS, ce sont les composants logiciels commerciaux. Ils correspondent à un programme interfaçable permettant de réaliser des fonctions logicielles [Arlat et al. 2000]. Avec l'essor de la programmation orientée objet, il est maintenant quotidien d'avoir recours à des classes dont le code et les performances ne sont pas connus en détail. Ceci introduit une nouvelle problématique lors des analyses de SdF.
- ***COTS matériel :*** Dans l'industrie actuelle, ce principe de composant unitaire technologique vendu prêt à l'emploi est de plus en plus répandu et s'étend aux composants matériels. La problématique quant aux analyses de SdF s'applique aussi pour les COTS matériels.

Quel que soit le type de COTS considéré, une classification de ces COTS est souvent réalisée en fonction de la connaissance que l'utilisateur a du composant qu'il doit intégrer à son système. Nous pouvons définir principalement 3 niveaux :

- ***Boite noire :*** le composant est livré avec une notice technique minimale et un mode d'emploi des fonctions du composant. Le code source, les plans de conception ne sont pas transmis. Dans le meilleur des cas, une spécification précise des interfaces et quelques données de fiabilité sont fournies.
- ***Boite grise :*** Aux informations fournies en boite noire est souvent ajoutée une description fonctionnelle de conception du système (voire le code source pour les COTS logiciels,

cependant non modifiable, car propriété du vendeur), un dossier technique complet, des données de REX de fiabilité.

- ***Boite blanche :*** Cette fois le composant est fourni dans son intégralité, avec toutes les informations de conception (plan de conception ou code source). Au niveau boite blanche, l'utilisateur est censé posséder toutes les informations sur le composant comme s'il l'avait développé lui-même. Cependant, la conception du COTS a été réalisée indépendamment du système qui l'intègre.

Dans la pratique, il est difficile de classer un COTS avec certitude dans l'une de ces catégories, c'est pourquoi on définit plusieurs points d'intérêt qui vont supporter nos réflexions par la suite :

- Informations de conception / Code source
- Données techniques
  - Interfaces
  - Fonctions
  - Technologie
- Données de fiabilité
  - Données de REX
  - Données de Test

Un autre point important qui doit être pris en compte si l'on veut avoir une vue globale de ce que représentent les COTS, est l'origine du composant. En effet, de plus en plus, les composants issus de développements antérieurs sont considérés comme des COTS. Ces composants qu'ils soient logiciels ou matériels devraient être classés au niveau boîte blanche, puisqu'ils ont été développés par l'entreprise. Cependant ces composants ne sont pas développés dans la démarche globale de conception du système, ils peuvent alors imposer des contraintes similaires à celles des composants commerciaux. De plus, lors de l'intégration de composants préalablement développés, ou développés dans d'autres services de l'entreprise, la perte d'informations ou l'impossibilité de pouvoir l'exploiter avec les outils actuels est parfois telle que le composant est plus souvent considéré au niveau boite grise, voire boite noire.

1. L'apport des COTS pour un projet industriel

Avant même de répertorier l'ensemble des risques inhérent à l'utilisation de COTS dans la conception d'un système critique, il est intéressant de se tourner vers les raisons qui incitent à les utiliser. Ces raisons, présentées par [Redmill 2004], gravitent autour de 3 axes :

- Une réduction des coûts de mise en œuvre.
- Un gain de temps de développement.
- Un accès à une technologie inaccessible par développement interne.

En effet, les COTS sont prévus pour être vendus à large échelle. Ce faisant, le coût de revient par unité est bien souvent inférieur au coût que représente un développement spécifique en interne. De la même façon, le choix se faisant sur catalogue, le produit est quasiment disponible dès que les exigences fonctionnelles sont formalisées. De plus, il n'est pas nécessaire de posséder en interne les compétences technologiques indispensables au développement des fonctions.

Et ce sont de ces qualités que naissent les défauts inhérents aux COTS. En effet, pour être rentable, le COTS est un composant générique, multifonction, produit à grande échelle. Bien que cela apporte la possibilité d'utiliser le COTS dans de multiples systèmes et configurations ce qui améliore encore sa rentabilité pour l'utilisateur, cela nuit à son intégration dans un système en particulier lorsque celui-ci est soumis à des contraintes strictes en matière de SdF :

- Le principe du composant générique implique des fonctionnalités qui ne sont pas spécifiquement développées pour répondre aux exigences. Il faudra apporter la preuve que le COTS respecte bien l'ensemble des exigences attribuées à la fonction qu'il doit remplir, sans pouvoir étudier finement son fonctionnement.
- Pour ce qui est des COTS multifonctions, on rencontre un autre problème, qui est celui des fonctions non utilisées. Il faudra prouver que l'impact de ces surcharges physiques et fonctionnelles sur la sûreté de fonctionnement est minime.

2. L'impact des COTS sur les études de SdF

Ainsi nous nous trouvons à devoir faire la preuve du fonctionnement d'un système complet tout en ignorant le fonctionnement particulier d'un de ses constituants. Dans ce cas, les études de SdF menées pour qualifier le système se feront à minima en fonction des informations livrées avec le COTS : par exemple, si l'on ne dispose que d'un taux de défaillance issu du REX, l'étude de SdF ne couvrira l'ensemble du système qu'en se limitant à une étude par arbre de défaillance. Même en nuançant cet exemple, il faut toutefois reconnaitre que le COTS aura un rôle de maillon faible, nivelant la qualité des études de SdF au niveau de celles avec lesquelles il est fourni.

On remarquera qu'en matière de COTS, les concepteurs ont tendance à contourner les risques que son utilisation implique en mettant en place des moyens de protections encapsulant le COTS. Cette démarche vise à reléguer les précisions de fonctionnement du COTS à un niveau de détails négligeable compte tenu de la sécurité qu'offre l'encapsulation du composant. Ce procédé s'apparente aux moyens de protection implantés en fin de conception pour pallier à des défaillances non anticipées. Or il est souvent possible d'éviter d'avoir recours à ce type de protection lorsque les études de sûreté de fonctionnement sont intégrées très tôt dans le processus de conception, ce qui permet de prévenir les défaillances plutôt que de devoir s'en protéger. Cependant, dans le cas des COTS, la prise en compte de contraintes de SdF n'est pas du ressort de l'assembleur. De plus, hormis dans le cas de COTS certifiés où la SdF est un argument de vente et de prix, mettre en place un processus de conception intégrant ces problématiques de SdF représente une perte de temps, et donc d'argent pour les vendeurs de COTS.

3. Intégration des COTS au système

Comment valider un système lorsqu'il intègre un COTS ? Quelles sont les études possibles qui restent valables lorsqu'un COTS est présent dans l'architecture de notre système ? De nombreux travaux essayent de répondre à ces questions. On trouve différentes approches selon que l'on considère les différentes catégories de COTS : matériel ou logiciel, selon la nature des informations considérées disponibles (boite noire, grise ou blanche), et selon le niveau de SdF à atteindre.

3.1. Validation des COTS logiciels

Selon [Rolland 1999], les COTS ne sont pas spécifiquement définis par les exigences, résultant en des incohérences lorsque ceux-ci doivent être interfacés avec le reste des composants du système. Il est proposé un moyen de pallier à ce problème sous forme d'une stratégie de représentation des exigences sous forme de carte permettant de montrer l'adéquation d'un COTS avec le reste du système. [Bishop et al. 2003] traitent de l'intégration des COTS dans des systèmes devant être certifiés SIL. La méthode définie repose sur une connaissance assez détaillée du COTS pour permettre la mise en forme d'une architecture fonctionnelle du composant. L'architecture fonctionnelle du COTS est alors mise en relation avec l'architecture fonctionnelle du système et le tout est analysé à l'aide de la méthode HAZOP, définie par le standard [IDE 2000]. La méthode HAZOP s'apparente à une AMDEC fonctionnelle où l'accent est mis sur l'analyse des flux entre composants logiciels à travers des MdD génériques.

Les travaux [Rolland 1999] et [Bishop et al. 2003] sont intiment liés aux activités d'IS (Définition des exigences et Analyse fonctionnelle). Bien que ces travaux soient définis dans le domaine de la conception logicielle, leur positionnement dans les premières phases de l'IS en fait des approches applicables à l'ensemble des domaines couvert par l'IS. De même, l'application de l'approche ISBM profiterait à chacun de ces travaux en permettant la connexion en un modèle cohérent des exigences système et du COTS comme le souhaite [Rolland 1999] et de même pour l'architecture fonctionnelle [Bishop et al. 2003].

3.2. Validation des COTS matériels

Comme nous venons de le voir, la validation des COTS matériels peut, en partie, être réalisée grâce à des méthodes type AMDEC fonctionnelle. En effet, l'architecture fonctionnelle détaillée d'un COTS matériel ne relève pas de la propriété du fabricant de COTS et sera parfois livrée.

Cependant, l'intégration de COTS matériel dans un système devant répondre à des contraintes de SdF passe par un minimum d'étude de fiabilité du composant. Or, le taux de défaillance d'un COTS n'est pas a priori connu. Les méthodes d'évaluation de taux de défaillance de composants sont principalement basées sur une connaissance assez fine du composant et de son comportement et deviennent donc inutilisables pour les COTS. Cependant, le guide FIDES, sorti en 2004 [FIDES 2004] et dont la dernière version normée date de 2011 [FIDES 2011] permet l'évaluation du taux de défaillance de COTS de types: cartes assemblées.

Le périmètre d'utilisation du guide FIDES représente déjà de nombreux domaines et ne cesse de s'étendre au cours des études sur les mécanismes de défaillances de nouveaux composants [Gireaudeau et al. 2010]. De plus, le nombre de cas pratiques validant la méthode FIDES croit lui aussi régulièrement [Charpenel et al. 2003] [Tourtelier et al. 2010]. Cela fait de la méthodologie FIDES une solution solide pour l'évaluation de la fiabilité des systèmes intégrant des COTS.

L'étude de la problématique des COTS fait finalement ressortir deux aspects :

- Nous ne possédons une connaissance fine du COTS que du point de vue fonctionnel
- Les méthodes d'évaluation de la fiabilité d'un COTS sont limitées.

C’est pourquoi nous nous efforcerons dans la suite de la thèse de nous focaliser sur les études de SdF qui restent réalisables et utiles lors de l’utilisation de COTS : l’AMDEC fonctionnelle et l’évaluation de fiabilité par la méthodologie FIDES.

V. Les études SdF adaptés aux COTS

Le cadre des études SdF utiles à l’intégration des COTS est maintenant délimité. Nous allons détailler ces deux méthodes spécifiquement afin de présenter l’ensemble des concepts manipulés par ces méthodes.

1. L’AMDEC fonctionnelle

1.1. Principes

Ce type d’étude est normalisé : MIL-STD1629A [MIL-STD1629A] et IEC 60 812 [IEC 60812]. Ces normes décrivent les différentes étapes que nous allons détailler dans la suite de cette section. Cette analyse consiste à étudier les unes après les autres les différentes fonctions conduisant le système à remplir sa mission. Ces fonctions ont été identifiées et définies au cours de l’analyse fonctionnelle du système, pendant laquelle les concepteurs ont dégagé la suite d’actions nécessaire pour réaliser la mission confiée au système développé. Pour chacune de ces fonctions, l’étude consiste à recenser les modes de défaillance conduisant à une dégradation du service rendu par la fonction. Pour chacun de ces modes, il s’agit de recenser leurs causes, leurs effets à différents niveaux (local, système), de qualifier la criticité du mode à travers sa gravité, sa probabilité d’occurrence et sa détectabilité et de définir les moyens de prévention, protection et détection permettant de réduire le risque. Les fonctions sont usuellement identifiées par un nom et éventuellement un identifiant (ex : freiner, F1.1). Les causes considèrent l’ensemble des phénomènes et mécanismes susceptibles de faire apparaître le mode. Ces phénomènes peuvent être indifféremment liés à des causes internes à la fonction étudiée ou à des fonctions périphériques. Les effets locaux décrivent les conséquences du mode sur les attributs du système, son comportement ainsi que sur son environnement. Les différentes cotations (gravité, occurrence, détectabilité) peuvent être vues comme des attributs des MdD détaillés. Elles sont fixées par les experts. Les moyens de réduction du risque sont enfin définis pour traiter les modes de défaillance dont la criticité dépasse le seuil d’acceptabilité. Cette description générique est souvent complétée par des points d’intérêt propre au système étudié, au domaine industriel ou à la pratique de l’entreprise qui mène l’étude.

L’AMDEC fonctionnelle est réalisée très tôt durant le processus de conception à un instant où les composants réalisant les traitements ne sont pas encore clairement définis. Ce type d’étude peut également être pertinent lorsque le nombre de composants à étudier n’est pas en adéquation avec le temps alloué et les résultats attendus pour l’étude. Enfin, ce type d’étude est primordial pour l’étude des COTS dont nous ne connaissons pas la composition organique détaillée. L’AMDEC fonctionnelle est une étude dont la représentation se fait le plus souvent sous forme tabulaire (Tableau I.2).

Tableau I.2 Forme tabulaire générique pour AMDEC fonctionnelle

Id	Fonction	Mode de Défaillance	Causes	Effets Locaux	Effets Systèmes	Gravité	Fréquence	Criticité	Moyens de Protection	Moyens de Prévention

1.2. Le processus de rédaction

Enfin, en se basant sur la syntaxe et la sémantique de l'AMDEC fonctionnelle, on peut déterminer le processus de réalisation de cette étude. Là encore, on peut voir que le standard MIL-STD1629A, bien que datant de 1980, proposait déjà une procédure afin de diriger l'analyse des modes de défaillance. Dans cette procédure, il est intéressant de constater que les premières activités proposées par la norme correspondent en fait à des activités d'IS : « Définir le système à étudier », « Définir les contraintes appliquées au système », « Définir les exigences de performances » et un point particulier est très développé, la description des mécanismes fonctionnels du système : « Définir l'architecture fonctionnelle », « Définir les interfaces fonctionnelles »,… La procédure prévoit même ensuite une activité de réification des connaissances élicitées précédemment sous forme de diagramme de bloc. Dans notre cas, ces étapes sont préalablement réalisées lors des processus d'analyse fonctionnelle supportés par SysML. Les activités suivantes de la norme MIL-STD1629A relèvent effectivement de l'étude de SdF : « Identifier tous les fonctions pouvant défaillir », « Identifier l'ensemble des modes de défaillance pour chacune », « Définir les causes et effets de ces MdD » et « Identifier l'ensemble des paramètres divers utiles pour l'étude ». Ainsi en reprenant la « procédure » de la norme MIL-STD1629A, on définit le processus de rédaction de l'AMDEC fonctionnelle (Figure I.8). Sur la Figure I.8, nous pouvons distinguer deux jeux de couleurs : les nuances de rouges correspondent aux données et activités relevant du domaine de la SdF, alors que les nuances de bleu relèvent du domaine de l'IS.

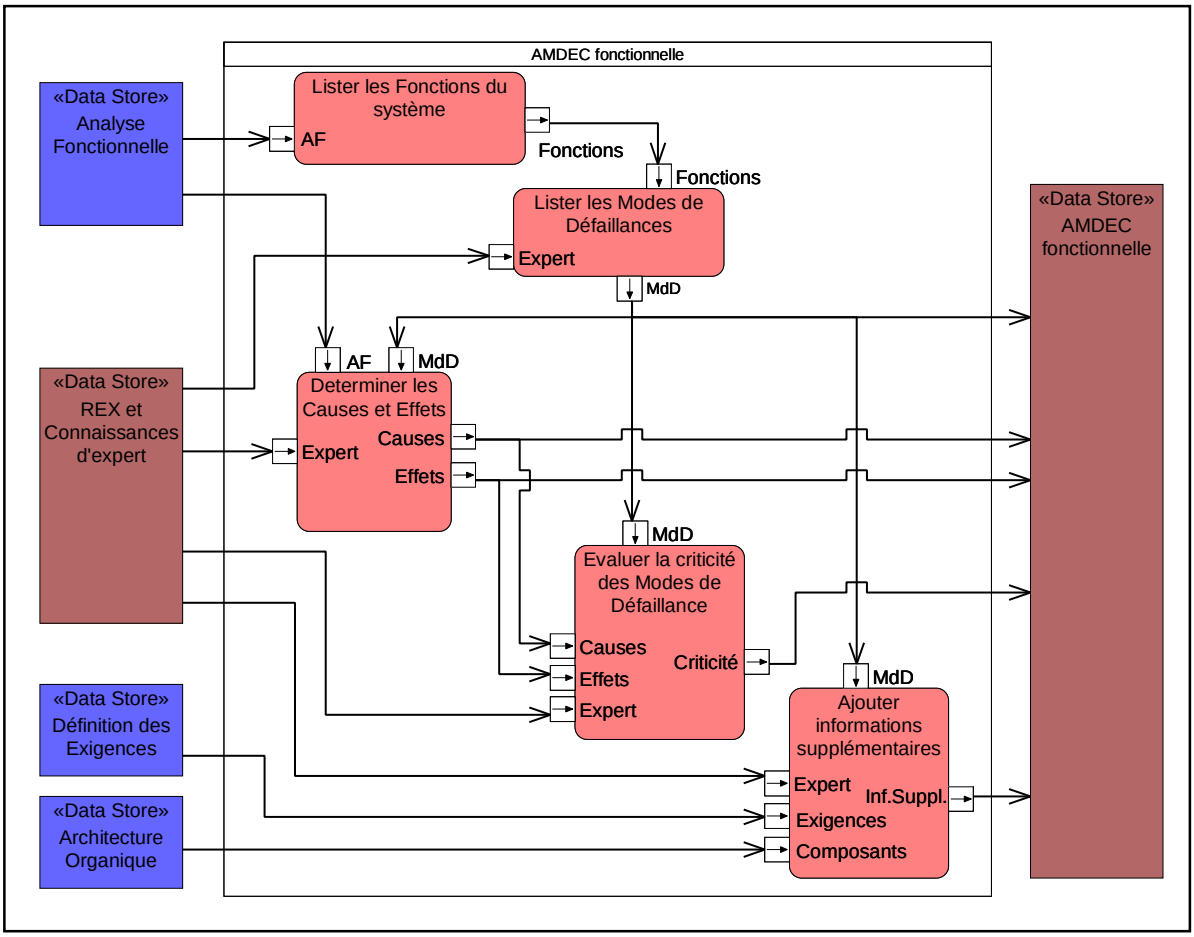


Figure I.8 Processus classique de rédaction d'AMDEC fonctionnelle

Lister les fonctions du système : Cette activité consiste à identifier l'ensemble des fonctions à inclure dans l'étude et à leur attribuer un éventuel identifiant qui permettra la lecture plus efficacement. Cette activité est d'autant plus aisée si les activités d'IS définissant l'architecture ont été réifiées et documentées convenablement.

Lister les Modes de Défaillance : La recherche des MdD s'appuie sur deux aspects complémentaires : l'application d'un raisonnement normalisé et le retour d'expérience. Le retour d'expérience peut exister sous différentes formes. Il peut s'agir de l'avis d'un expert de SdF, de l'analyse de résultat de l'étude d'un système similaire à celui étudié, ou bien l'utilisation d'un référentiel normalisé et formaté faisant état des fonctions et MdD connus. L'utilisation de référentiels est particulièrement utile pour les AMDEC composants. Cependant, dans le cas des études fonctionnelles, il est plus délicat d'identifier des unités fonctionnelles à travers de multiples projets. Pour compenser l'éventuelle absence de ce type de référentiel, avoir recours à une liste de MdD génériques est une solution. La norme MIL-STD1629A propose les MdD génériques suivants :

- Perte de la fonction.
- Fonction exécutée de façon intempestive.
- Retard d'exécution de la fonction.
- Démarrage de la fonction impossible.
- Arrêt de la fonction impossible.

- Fonction intermittente.
- Fonction dégradée.
- Autres conditions de défaillances propres à la fonction étudiée.

Notons que les « autres conditions de défaillances » relèvent de l'avis d'expert.

Déterminer les Causes et Effets : Il s'agit là d'étudier la naissance et la propagation d'un MdD et de mettre en évidence les éléments ayant causé la défaillance, ainsi que les éléments impactés à différentes échelles par la défaillance. Une fonction est une entité délivrant des sorties en fonction d'entrées et de paramètres d'influence. La fonction est considérée défaillante si les sorties obtenues sont en dehors de leurs domaines attendus. Les caractéristiques surveillées pour les sorties sont relatives à leur amplitude, leur variation ou leur délai et durée de mise à disposition. Les entrées peuvent être des sorties des fonctions amont ou être imposées par l'utilisateur. Les sorties peuvent être consommées par l'utilisateur ou être les entrées d'autres fonctions. Les fonctions fournissant des entrées ou utilisant des sorties de la fonction étudiée sont appelées fonctions périphériques. Les paramètres d'influences de la fonction sont eux aussi de diverses origines. Les caractéristiques des composants exécutant la fonction, ainsi que les conditions environnementales dans lesquelles se déroule la fonction étudiée représentent les principales sources de paramètres d'influence. Nous désignerons d'ailleurs les composants participant à la réalisation de la fonction par le terme de composants support. L'AMDEC fonctionnelle ne nécessite pas que les composants supports soient connus, mais lorsque les composants supports sont définis, de nouveaux paramètres d'influence peuvent alors émerger nécessitant de réévaluer l'AMDEC afin de s'assurer de l'apparition ou non de nouveaux MdD.

Ainsi, trouver les causes de la défaillance de la fonction, revient à identifier les paramètres d'influence et les entrées pouvant faire dévier les sorties de leurs valeurs attendues. Nous pouvons donc distinguer deux types de causes : « Cause interne » et « Cause externe ». Les causes internes sont dues aux paramètres d'influence de la fonction : paramètres environnementaux, caractéristiques des composants support, … Les causes externes correspondent aux entrées fournies par les fonctions périphériques. L'étude consiste ensuite à rechercher ce qui conduit ces paramètres à prendre des valeurs participant à la déviation des sorties de la fonction de leurs valeurs attendues. Ceci peut être, par exemple, la casse d'un élément des composants support, la dégradation des conditions environnementales ou la défaillance d'une fonction périphérique. Une fois les causes établies, il faut évaluer les effets que les sorties dysfonctionnelles vont avoir sur les fonctions périphériques et sur le système. C'est pourquoi, on distingue deux types d'effets : « Effets locaux » et « Effets Système ». L'impact des sorties fonctionnelles défaillantes sur les fonctions périphériques constitue les effets locaux. Les effets système quant à eux sont multiples. La première définition de ces effets correspond aux besoins fonctionnels de haut niveau qui ne sont plus exécutés convenablement à cause du MdD, auxquels peuvent être rajoutées des données jugées utiles par l'expert.

Évaluer la criticité des MdD : Il s'agit là de coter le risque que représente un MdD. La cotation passe par l'estimation de 3 composantes : la gravité, l'occurrence et la détectabilité. Obtenir les trois indices permet deux choses : fournir une indication sur le levier à utiliser pour réduire le risque et calculer la criticité qui donne les priorités dans le traitement des MdD. Différents critères

participent au calcul de la criticité et il existe selon les normes et pratiques industrielles plusieurs échelles de cotation pour chacun d'entre eux. Nous allons en détailler quelques exemples.

La norme MIL-STD1629A définit une classification de la gravité du mode de défaillance selon 4 niveaux qualitatifs :

1 : Mineure – Une défaillance pas assez grave pour causer des blessures, des dommages à l'environnement ou au système, mais qui peut engendrer une maintenance corrective rapide.
2 : Marginale – Une défaillance qui peut causer de légères blessures, des dommages mineurs à l'environnement ou au système qui résulterait en une perte de disponibilité du système ou une dégradation du fonctionnement attendu.
3 : Critique – Une défaillance qui peut causer de graves blessures, un endommagement majeur de l'environnement ou du système qui résulterait en une impossibilité de mener à bien la mission du système.
4 : Catastrophique – Une défaillance qui peut engendrer un décès ou la perte du système complet.

À quelques détails près, il s'agit de la classification la plus utilisée, cependant certains industriels complètent parfois cette approche qualitative par une approche quantitative en estimant la gravité d'un mode de défaillance en fonction du coût financier que celui-ci implique (dégâts, réparation, retards, rendement, image,…). Bien que l'étude quantitative de la gravité reste marginale, pour l'évaluation de la probabilité d'occurrence d'un mode de défaillance, l'approche qualitative et l'approche quantitative cohabitent. Encore une fois, la norme MIL-STD1629A [MIL-STD1629A] propose une échelle qualitative qui n'a que peu évolué dans les pratiques actuelles :

1 : Fréquent – La probabilité d'occurrence est élevée. Cela correspond à une estimation de la probabilité d'occurrence du mode de défaillance, pendant le fonctionnement du système, supérieure à 0.2.
2 : Raisonnablement probable – La probabilité d'occurrence est modérée. Cela correspond à une estimation de la probabilité d'occurrence du mode de défaillance, pendant le fonctionnement du système, comprise entre 0.1 et 0.2.
3 : Occasionnel – La probabilité d'occurrence est considérée occasionnelle. Cela correspond à une estimation de la probabilité d'occurrence du mode de défaillance, pendant le fonctionnement du système, comprise entre 0.01 et 0.1.
4 : Rare – La probabilité d'occurrence est faible. Cela correspond à une estimation de la probabilité d'occurrence du mode de défaillance, pendant le fonctionnement du système, comprise entre 0.001 et 0.01.
5 : Extrêmement Rare – La probabilité d'occurrence est extrêmement faible. Cela correspond à une estimation de la probabilité d'occurrence du mode de défaillance, pendant le fonctionnement du système, inférieure à 0.001.

Cette échelle se voulant qualitative, les probabilités données en exemple sont à ajuster en fonction du système étudié et du domaine industriel considéré. L'approche qualitative quant à elle, fera appel à des études de fiabilité plus complète pour déterminer la probabilité d'occurrence du mode de défaillance. Ces études pourront par exemple s'appuyer sur des recueils de données de REX tel que la MIL-HDBK-217F [MIL-HDBK-217F] ou le guide FIDES [FIDES 2011].

La détectabilité n'est quant à elle pas systématiquement évaluée. La MIL-STD1629A, par exemple, ne propose aucune échelle de cotation de la détectabilité. Cependant, la détectabilité est de plus en plus prise en compte en utilisant une échelle qualitative telle que :

1 : Aisément détectable – Il existe un signe avant-coureur de la défaillance que l'opérateur ou la maintenance préventive pourront observer. Une action préventive pourra alors éviter l'incident.
2 : Détectable – Il existe un signe avant-coureur de la défaillance, mais il y a un risque que ce signe ne soit pas perçu par l'opérateur ou les équipes de maintenance préventive.
3 : Difficilement détectable – Il existe un signe avant-coureur de la défaillance, mais qui est difficilement décelable.
4 : Indétectable – Il n'existe aucun signe avant-coureur de la défaillance.

Enfin, lorsque ces 3 critères ont été évalués, il est possible de calculer la criticité du mode de défaillance. La criticité correspond au produit de ces critères :

$$\text{Criticité} = \text{Gravité} \times \text{Probabilité} \times \text{Détectabilité}$$

Le calcul de la criticité permet ensuite de planifier la prise en compte des modes de défaillance. Il est usuel d'établir une matrice de criticité comme illustrée par le Tableau I.3. Cette matrice est cohérente avec la norme MIL-STD1629A et par conséquent ne s'applique pas aux études où la détectabilité est évaluée.

Tableau I.3 Matrice de criticité des modes de défaillance

			Niveau de Gravité			
			1	**2**	**3**	**4**
			Mineur	**Marginal**	**Critique**	**Catastrophique**
Probabilité d'occurrence	**5**	**Fréquent**	Indésirable	Inacceptable	Inacceptable	Inacceptable
	4	**Probable**	Acceptable	Indésirable	Inacceptable	Inacceptable
	3	**Occasionnel**	Acceptable	Indésirable	Indésirable	Inacceptable
	2	**Rare**	Négligeable	Acceptable	Indésirable	Indésirable
	1	**Extrêmement Rare**	Négligeable	Négligeable	Acceptable	Acceptable

L'allocation des niveaux d'acceptabilité des modes de défaillances est une donnée qui est paramétrée par l'expert en fonction du produit, de l'environnement d'utilisation du produit et des éventuelles exigences de sûreté de fonctionnement s'appliquant à l'étude des modes de défaillance. L'équivalent du tableau précédent pourrait être réalisé en 3 dimensions pour prendre en compte la détectabilité, cependant il est plus pratique d'effectuer une catégorisation en fonction de la valeur de criticité. En effet, on peut voir que le Tableau I.3 correspond à l'échelle :

- Criticité ≤ 2 : Mode de défaillance Négligeable
- $3 \leq$ Criticité ≤ 4 : Mode de défaillance Acceptable
- $5 \leq$ Criticité ≤ 9 : Mode de défaillance Indésirable
- Criticité ≥ 10 : Mode de défaillance Inacceptable

Ainsi pour inclure la détectabilité, une échelle de ce type devra être décidée. Cette échelle a pour vocation de déterminer le niveau de sûreté nécessaire à la validation du système. Cette échelle pourra donc faire l'objet d'une exigence de SdF.

Ajouter les informations supplémentaires : Cette activité est multiple et sera décomposée en plusieurs sous activités variant d'un expert à l'autre, d'un système à un autre, d'un domaine industriel à un autre. Nous allons détailler les activités les plus représentatives et notamment celles qui nous semblent utiles pour le projet LEA et pour notre problématique de maximiser l'apport de l'ISBM pour la SdF.

Traditionnellement, la recherche de moyens de réduire le risque est réalisée. Ces moyens de réduction du risque sont en rapport aux critères de cotation précédents :

- Moyens de prévention pour réduire la probabilité d'occurrence de la défaillance.
- Moyens de protection pour réduire la gravité de la défaillance.
- Moyens de détection pour optimiser la détectabilité de la défaillance.

Les moyens de réduction du risque doivent être déterminés par la collaboration des équipes de conception et de sûreté de fonctionnement. L'importance de cette étape varie pour chaque mode de défaillance en fonction de sa criticité et de la catégorie d'acceptation auquel il appartient. En effet, il est inutile de dépenser du temps à trouver des moyens de réduction de risque pour les modes de défaillance considérés comme Négligeable, autant investir ce temps pour réduire le risque des défaillances classées Inacceptable.

Par définition, ces moyens de prévention, protection et détection influent sur les critères de criticité évalués précédemment. Ainsi, on trouvera parfois une réévaluation de la criticité et des critères qui permettent de la calculer, après avoir défini les moyens de réduction du risque pour pouvoir apprécier l'impact de ces moyens sur le risque. Syntaxiquement, cela se retrouve sous la forme d'un groupe de colonne « Gravité », « Probabilité », « Détectabilité » et « Criticité » après les colonnes des Moyens de réduction du risque. Cette dernière colonne définissant la criticité post-réduction des risques devra par exemple comporter uniquement des niveaux Négligeable ou Acceptable pour valider la SdF du système. Si la réévaluation de la criticité n'est pas réalisée, alors les moyens de réduction du risque proposés sont transmis aux experts conception qui les étudient pour faire évoluer les spécifications du système de façon appropriée. L'impact des moyens de réduction des risques sera alors apparent uniquement lors de l'AMDEC suivante. Ainsi on voit que l'AMDEC doit être réalisée à chaque modification du système. Il s'agit d'une analyse qui est propre à une version unique du système. Cette étape permet aussi de montrer l'intérêt de réaliser l'AMDEC très tôt lors des spécifications du système quand celles-ci peuvent encore être modifiées sans surcoût trop important. Si l'AMDEC est réalisée après la conception du système, le champ des possibilités pour les moyens de réduction du risque diminue considérablement et se restreint à des moyens externes au système.

Dans cette activité d'ajout d'informations supplémentaires, nous pouvons ajouter deux points qui sont particulièrement adaptés dans le cadre d'un projet où le système est spécifié avec une approche ISBM supportée par un langage de modélisation comme SysML. En effet, les fonctions, les composants et les exigences sont reliés dans le modèle et permettent d'établir des matrices d'allocation et de satisfaction utiles aux experts de SdF pour la rédaction de l'AMDEC. Ainsi, lors de l'analyse des modes de défaillance, il est possible de rechercher pour chaque fonction étudiée le

composant qui doit exécuter la fonction et qui peut être à l'origine de la défaillance (nous avons vu comment les paramètres d'influences, pris en compte dans la détermination de la cause de défaillance, pouvaient être liés au composant support). De même, pour chaque fonction, il est possible de rechercher les éventuelles exigences qui sont impactées par la défaillance.

Nous verrons dans le chapitre III, que les concepts présentés ici et manipulés lors de la rédaction fonctionnelle sont intimement liés aux concepts de l'ingénierie système et que l'approche ISBM permet un gain d'efficacité de l'expert non négligeable. Nous allons avant cela effectuer ce même travail de présentation des concepts manipulés pour la méthodologie FIDES.

2. Le guide FIDES

Le Guide FIDES [FIDES 2011] représente une méthodologie globale d'ingénierie de la fiabilité en électronique. Il est constitué de deux parties :

- un guide d'évaluation prévisionnelle de la fiabilité,
- un guide de maîtrise et d'audit du processus fiabilité.

Les objectifs du Guide FIDES sont d'une part de permettre une évaluation réaliste de la fiabilité des équipements électroniques, y compris dans les systèmes qui rencontrent des environnements sévères (système de défense, aéronautique, électronique industrielle, transport…), et d'autre part de fournir un outil concret pour la construction et la maîtrise de cette fiabilité. L'approche fiabilité de FIDES est basée sur la prise en compte des trois composantes Technologie, Processus et Utilisation. Ces composantes sont considérées pour l'ensemble du cycle de vie depuis la phase de spécification du produit jusqu'à la phase d'exploitation et de maintenance. La Technologie couvre aussi bien celle de l'article lui-même que celle de son intégration dans le produit. Le Processus considère toutes les pratiques et règles de l'art depuis la spécification du produit jusqu'à son remplacement. L'Utilisation prend en compte à la fois les contraintes d'emploi définies par la conception du produit et celles en exploitation chez l'utilisateur final.

Le guide FIDES prévoit un modèle mathématique spécifique d'évaluation de la fiabilité de cartes électroniques COTS dans sa méthodologie. Ce modèle est destiné aux cartes du commerce qui remplissent des fonctions électroniques standards. Ce modèle est défini pour :

- Estimer la fiabilité de cartes COTS dont le fabricant n'a pas donné d'information de fiabilité.
- Estimer la fiabilité de cartes COTS dans des environnements autres que celui pour lequel le fabricant a donné la fiabilité.
- Estimer la fiabilité d'un ensemble de cartes COTS de différentes origines dans un référentiel commun, sachant que les fabricants de cartes COTS, quand ils donnent une information de fiabilité, n'en précisent pas forcément ni l'origine, ni les conditions dans lesquelles elle s'applique.

Ce modèle est prévu pour être mis en œuvre à partir des informations directement disponibles sur la carte COTS. Le guide FIDES considère que cela se limite à la fiche de données techniques de la carte. Le prérequis à l'utilisation de ce modèle est de posséder une bonne connaissance des fonctions électroniques réalisées par le COTS. Ainsi, là encore, l'étude du COTS repose sur une connaissance de la décomposition fonctionnelle de celui-ci.

Cependant, dans le cas d'un COTS en boite grise où une décomposition organique succincte de la carte est fournie, la méthode FIDES originale peut être utilisée. Le guide FIDES propose plusieurs niveaux de détails permettant d'avoir recours à une méthode simplifiée lorsque la connaissance du système n'est pas totale permettant ainsi des études moins précises, mais suffisantes pour orienter les efforts de conception. Cette méthode simplifiée peut s'appliquer aux cartes COTS lorsque :

- Les cartes COTS sont constituées de composants électroniques dans le périmètre de FIDES.
- La liste des composants utilisés dans la carte COTS est connue.
- Optionnellement : la qualité de l'étude sera accrue si l'organisation interne générale des composants est connue (composants en contact direct avec le reste du système ou non).

Finalement, la méthodologie présentée par le guide FIDES représente un outil important pour la prédiction de fiabilité des composants de type COTS, quel que soit le niveau de détails (boite blanche à boite noire). Ainsi, elle représente une solution complémentaire à l'AMDEC fonctionnelle qui délaisse l'aspect organique du système souvent méconnu lors de l'utilisation de COTS.

VI. Conclusion

Dans ce premier chapitre, nous avons présenté le cadre de nos travaux : la conception et l'étude de sûreté de fonctionnement d'un système complexe embarqué intégrant des COTS. Nous avons défini le cadre méthodologique qui nous semble être la meilleure réponse à nos besoins :

- Une approche d'ingénierie système à base de modèles,
- Le langage SysML comme langage système central,
- Le cadre méthodologique MéDISIS pour la valorisation de l'ISBM.

Nous avons vu que les besoins actuels de l'industrie impliquent le développement de systèmes complexes et parfois critiques. Les attentes en termes de coûts, de temps de développement et la complexité des systèmes conçus nécessitent l'adoption de nouvelles méthodes d'Ingénierie Système. L'approche à base de modèle a été choisie, car elle offre une meilleure structuration et expressivité des concepts manipulés lors des activités d'ingénierie système. L'ISBM permet aussi l'établissement d'un modèle système, façonné et détaillé, au fur et à mesure des itérations au sein du processus d'IS. Ce modèle système offre la possibilité d'être le support central à l'ensemble des activités d'un projet, y compris les activités propres à des domaines d'expertise spécifiques tels que la SdF.

Nous avons identifié le langage SysML comme étant le meilleur langage système pour réaliser ce modèle système. Ses larges possibilités de modélisation des concepts et des relations entre concepts en fait un langage particulièrement adapté à l'approche ISBM. SysML reposant sur une structure orientée objet et n'imposant pas de sémantique, c'est un langage générique qui pourra être adapté à différents processus industriels par définition d'une sémantique, c'est-à-dire par définition de règles de modélisation des concepts de l'IS. C'est pourquoi nous présenterons dans le chapitre suivant les activités d'IS et la sémantique SysML que nous considérons pour l'ensemble de la thèse. De plus, nous avons pu voir que la littérature répertorie de nombreux travaux sur l'utilisation de SysML pour faciliter ou améliorer des études de SdF.

C’est le cas de la méthodologie MéDISIS présentée par [David 2009] qui construit autour d’un modèle système en SysML un framework d’outils de rédaction de documents et de génération de modèles propres à la SdF. Ces outils de la méthodologie MéDISIS sont soutenus par une base de données des comportements dysfonctionnels, elle-même supportée par SysML, qui permet la pérennisation des données de SdF au travers de multiples projets ayant recours au cadre méthodologique MéDISIS. Ces différents points nous poussent à nous inscrire dans ce cadre pour la suite de la thèse.

Une difficulté particulière devant être prise en compte dans les travaux de thèse a été évoquée : l’utilisation de composants sur étagère. Les problématiques spécifiques aux COTS ont été présentées. Par cette description, nous avons pu définir les études de SdF proposée par la littérature qui permettent actuellement de surpasser les limitations imposées par l’utilisation de COTS. Ainsi, la suite de la thèse va s’efforcer de valoriser l’approche d’ingénierie système à base de modèles SysML pour les études de SdF qui restent réalisables lorsque des COTS sont présents dans le système étudié.

Pour cela, nous allons voir dans un premier temps comment les principes de MéDISIS présentés par [David 2009] peuvent être adaptés à des études portant sur l’architecture fonctionnelle du système. Nous présenterons donc des travaux qui permettent la rédaction de pré-AMDEC fonctionnelle améliorant l’efficacité de l’expert SdF dans le cadre de la rédaction d’une AMDEC fonctionnelle. L’étude des aspects matériels d’un COTS étant toutefois inévitable, nous verrons comment la méthodologie MéDISIS et le guide FIDES peuvent être accordés. En effet, à travers la réification de la méthodologie FIDES en entités SysML, nous mettrons en avant le lien possible de la Base de données des Comportements Dysfonctionnels de MéDISIS et la méthodologie d’évaluation de fiabilité de FIDES afin de permettre l’intégration de celle-ci dans une approche ISBM.

Enfin, nous appliquerons l’ensemble des méthodes et processus qui ont été déployés au cours de la thèse dans le cadre du projet industriel LEA. Nous verrons comment les processus décrits dans les travaux originels de MéDISIS [David 2009] et les travaux de la thèse permettent une étude complète du système.

Chapitre II. Processus d'ingénierie système supporté par SysML

I. Introduction

Dans le chapitre précédent, un grand nombre de travaux traitant de la conception et de l'analyse de sûreté de fonctionnement des systèmes complexes ont été présentés. Les défis spécifiques que représentent l'utilisation et l'intégration de COTS dans ces systèmes ont aussi été abordés. Nous avons finalement statué sur l'intérêt d'adopter l'approche d'ingénierie système supportée par les modèles afin de profiter des avantages que propose une telle approche et plus particulièrement la possibilité d'utiliser le modèle système comme pivot aux différentes activités qui interviennent au cours d'un projet.

L'ingénierie système est constituée de différentes activités qui peuvent être réalisées selon différents processus. Dans le chapitre précédent, à la Figure I.2 nous présentions de manière générale le processus d'ingénierie système que nous allons maintenant détailler. Les concepts manipulés au sein de chaque activité d'IS : Elicitation des besoins, Définition des exigences, Analyse fonctionnelle et Description organique, sera détaillée. Une activité étant elle-même décomposable en un processus de sous-activités, ces processus seront définis en s'inspirant des différentes normes d'ingénierie système et plus particulièrement de l'IEEE 1220. Cette description des processus de chaque activité sera réalisée en tenant compte de notre approche à base de modèles et du contexte de conception d'un système complexe critique.

SysML est le langage de modélisation système retenu pour les activités d'IS et nous allons donc aussi présenter dans ce chapitre les moyens syntaxiques et les règles de modélisation permettant le respect de la sémantique. Ainsi, les concepts manipulés et résultants des activités d'ingénierie système pourront être représentés et identifiés dans un unique modèle SysML qui servira de fondation aux processus MéDISIS décrits par [David 2009] et à ceux présentés dans les chapitres III et IV de la thèse.

Ce chapitre présente en grande partie des travaux relevant de la recherche bibliographique. Cependant, la présentation du langage SysML, encore peu connu à ce jour, permet de s'assurer de la compréhension de celui-ci, pour la lecture de la suite des travaux. De plus, la sémantique SysML proposée pour l'ensemble des activités d'IS considérées est originale et permet de poser les fondations des chapitres III à V. Ces deux points justifient, à notre sens, la présence de ce chapitre dans le manuscrit.

Notons que les travaux de la thèse se fondent sur la version 1.2 de SysML [OMG 2010]. L'essentiel des travaux présentés reste vrai avec la dernière version de SysML (1.3) [OMG 2012] à l'exception de la notion de *flowport* qui a été revue et que nous utilisons dans le §V.4.a de ce chapitre. De plus, l'ensemble des termes propres au langage SysML sera utilisé dans leur langue originale, en anglais, et rédigé en italique.

II. Elicitation des besoins

1. Description détaillée de l'activité

L'élicitation des besoins n'est pas une activité reconnue par la plupart des standards d'ingénierie système [IEEE 1220][EIA 632][IEC 15288]. De fait, ils considèrent la description des exigences

comme activité initiale du processus d'ingénierie système. Cependant, dans l'ensemble de ces standards la description des attentes des parties prenantes est préalablement nécessaire. La norme IEC 15288 [IEC 15288] définit même un processus de définition des exigences des parties prenantes à part du processus de définition des exigences du système. L'existence de ce processus met en évidence la nécessité de spécifier les attentes des parties prenantes en tant que telles. Ce processus, tel qu'il est décrit dans la norme IEC 15288 [IEC 15288], se décompose en 3 étapes :

1. Eliciter les exigences.
2. Définir les exigences.
3. Analyser et maintenir les exigences.

L'étape 1 consiste principalement à lister les acteurs extérieurs au système ayant une interaction avec le système au cours de l'ensemble de son cycle de vie, depuis la spécification jusqu'à son utilisation, à nommer l'ensemble des attentes des parties prenantes (qui représentent un sous-ensemble des acteurs extérieurs) et à relier les attentes aux acteurs correspondants.

L'étape 2 a pour but de définir plus finement les besoins énoncés lors de l'élicitation. Cette étape concorde avec la définition que fait la norme IEEE 1220 [IEEE 1220] des besoins des parties prenantes :

- ce que le système doit accomplir,
- avec quelle performance chaque fonction doit être accomplie,
- les environnements naturels et induits dans lesquels le système devra opérer ou être utilisé,
- les contraintes liées au contexte de production (ex : financement, objectif de coût, calendrier, interfaces externes, …).

L'étape 3 aborde l'analyse et la validation de la cohérence des exigences des parties prenantes entre elles ainsi que leur maintien à jour au cours de l'évolution du projet. Une dernière référence aux besoins des parties prenantes est faite dans la partie validation des exigences système de la norme IEEE 1220 [IEEE 1220]. Il y est précisé que la validation des exigences se fera par comparaison avec les attentes des parties prenantes, les contraintes projets et entreprises, et les contraintes externes. Cette référence met de nouveau en avant la nécessité de structurer et de formaliser l'expression des besoins des parties prenantes afin de permettre la validation de l'ensemble des exigences du système. Finalement, en corrélant les diverses descriptions des besoins qui sont faites, nous pouvons élaborer une définition de ce que l'élicitation des besoins doit couvrir :

- la description des attentes fonctionnelles du système,
- la description des interfaces externes du système, c'est-à-dire, l'ensemble des acteurs avec lequel il doit interagir,
- la définition du cadre environnemental du système, sous-entendu la dénomination des contraintes physiques, technologiques, financières, normatives et de sûreté de fonctionnement auxquels le projet doit se plier. La description de ces contraintes n'est pas obligatoire à ce stade.

L'ensemble de ces informations doit être spécifié pour l'ensemble des phases de vie du système, aussi bien lors de sa production que pendant son utilisation, son stockage éventuel voire son démantèlement. La description des phases de vie peut être structurée par décomposition des phases en sous-phases permettant d'obtenir des unités temporelles et fonctionnelles pour l'étude du

système. Chaque phase de vie devra être justifiée par rapport aux phases du même niveau hiérarchique. Ainsi deux phases de vie doivent être séparées si l'une des deux comporte un attribut ou une contrainte qui lui est propre ou bien si la valeur d'un attribut commun aux deux est différente pour chacune des phases.

Maintenant que nous avons défini les informations nécessitant d'être formalisées, nous pouvons aborder les techniques qui permettent effectivement de créer cette information. Pour cela, nous pouvons prendre exemple sur [Vanderperren 2005] qui décrit plusieurs techniques d'élicitation des exigences. Cependant, ces travaux évoquent deux techniques applicables à l'élicitation des besoins :

- Les techniques d'élicitation non structurées (brainstorming, entretien ouverts, …),
- Les techniques d'élicitation structurées (entretiens organisés, groupes de réflexion par domaine spécifique, analyse des facteurs de réussite, …).

Des méthodes ayant pour but de formaliser la pensée pour exprimer les besoins existent aussi, notamment la méthode APTE [Bretesche 2000] ou la méthode MISME [AV].

2. Réification de cette activité en SysML

Notre hypothèse de travail étant d'utiliser SysML pour l'ensemble du modèle système, nous allons nous intéresser aux éléments de modélisation proposés par la norme [SysML 1.2] qui permettent de réifier les résultats de l'activité d'élicitation des besoins. Nous optons pour la combinaison de deux types de diagrammes SysML afin de réaliser l'élicitation des besoins : les *Block Definition Diagrams* (BDD : diagrammes de bloc) et les *Use Case Diagrams* (UCD : diagrammes de cas d'utilisation). Les BDD permettent la description de contexte afin de spécifier, dès les prémices du projet, le contexte de développement et d'utilisation du système étudié. Ce principe est présenté en annexe dans le standard SysML [OMG 2010], sous la forme d'un exemple de « diagramme de contexte ». Il est alors question d'utiliser un *Internal Block Diagram* (IBD : diagramme interne de bloc) et des *stéréotypes*. Cette solution comporte deux inconvénients : la création d'IBD nécessite un niveau de structuration et de détails trop important à ce stade du projet, et l'utilisation d'un *profile* implique la perte de l'aspect standard de la méthode. En effet, le mécanisme de *profile* proposé par le standard SysML consiste en la définition d'un ensemble de *stéréotypes*. L'utilisation de *stéréotypes* est un moyen de masquer des mécanismes d'héritage à l'utilisateur afin de lui proposer une des entités spécifiques sans lui laisser percevoir leur origine en entités standards SysML.

Finalement, nous optons pour le BDD qui est plus abstrait. Nous devons alors nous appuyer sur les mécanismes d'héritage pour pallier à l'utilisation de *stéréotypes*. Ainsi cela permet de se soustraire à l'utilisation de *profile*, mais n'empêchera pas à chacun de créer ses propres *profiles* pour masquer ces informations auprès de ces utilisateurs. Ce type de BDD utilisé pour l'élicitation des besoins sera référé sous le nom de diagramme de contexte dans la suite de la thèse.

Un exemple de diagramme de contexte est présenté à la Figure II.1. Le système se trouve au centre et est relié à 3 types d'entités : les acteurs extérieurs interagissant avec le système, les phases de vie du système et les contraintes appliquées au système. Les *blocks* représentant les phases de vie héritent tous d'un *block* générique « Phase de vie ». Il est possible d'organiser les phases de vie entre elles en utilisant des relations de *composition*. Ainsi une phase de vie pourra être décomposée

en plusieurs sous-phases, permettant ainsi de structurer ces phases. Seules les phases de vie de plus bas niveau sont associées à notre système. De même les contraintes sont représentées par *constraint block* héritant d'un *constraint block* générique « Contraintes d'environnement ». D'autres *constraint blocks* génériques tels que : « Contraintes normatives » et « Contraintes technologiques » peuvent être ajoutés pour compléter l'ensemble des domaines à couvrir lors de l'analyse de besoins. Une allocation des contraintes aux phases de vie est possible pour exprimer le domaine d'application d'une contrainte grâce à la relation SysML *allocatedTo*. Enfin, les acteurs interagissant avec le système sont représentés par des *actors* et sont associés au système par l'*association* SysML. Les détails de modélisation des *blocks* dans un BDD seront expliqués dans la partie description organique.

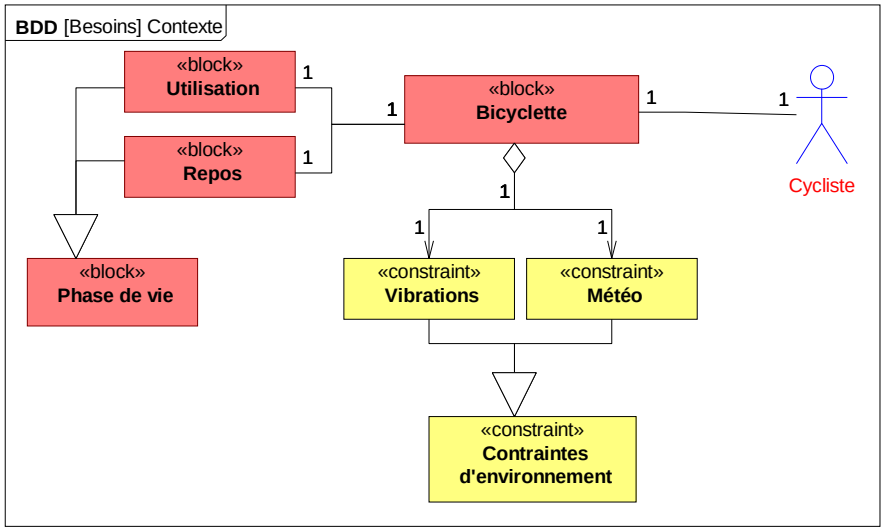


Figure II.1 : Exemple de diagramme de contexte pour l'élicitation des besoins

Le diagramme de contexte permet donc de décrire le cadre opérationnel, environnemental et normatif du système, mais il ne permet pas d'étudier les besoins en termes de fonctionnalités. Cette description des attentes fonctionnelles du système par les parties prenantes est réalisée avec le diagramme de cas d'utilisation pour compléter l'analyse des besoins, dont la Figure II.2 représente un exemple. Selon [OMG 2012], le diagramme de cas d'utilisation se compose principalement de 3 éléments : les cas d'utilisation représentés par des ellipses libellés du nom du cas d'utilisation, les acteurs extérieurs représentés comme des petits personnages (auxquels on peut substituer des rectangles pour un effet purement graphique) et le système étudié représenté par un rectangle englobant les cas d'utilisation. Les acteurs sont reliés aux *uses cases* par un lien de communication représenté par un trait plein. Les *use cases* entre eux peuvent être reliés par une connexion de type *include*, *extend* ou *generalization*. La relation *include* transcrit une factorisation de fonctionnalités communes partagées par plusieurs *uses cases* et indispensable à la réalisation du *use case* d'origine. La relation *extend* correspond quant à elle a des fonctionnalités optionnelles qui permettent une extension du *use case* d'origine. Des *extensions point* et des *conditions* d'extensions peuvent être définies pour raffiner l'utilisation de relation *extend*. Le mécanisme d'héritage/généralisation permet de définir des variantes d'un même *use case* d'origine. Le rappel graphique des phases de vie peut être fait par l'utilisation de *frame box* représentées par des rectangles en pointillés, permettant ainsi d'établir la cohérence entre les deux vues (BDD et UCD) utilisées pour cette activité.

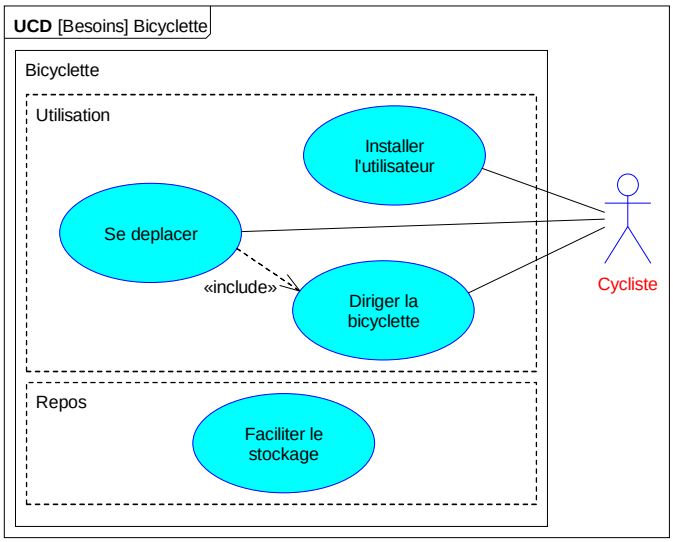


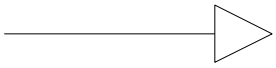
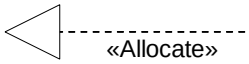
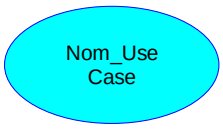
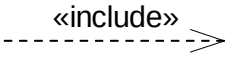
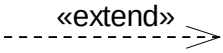
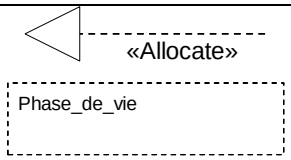
Figure II.2 : Exemple de UCD pour l'élicitation des besoins

Chaque cas d'utilisation représentera une fonctionnalité attendue par les parties prenantes. Cette fonctionnalité sera exprimée par rapport aux acteurs qui devront interagir avec elle et par rapport aux autres fonctionnalités du système. Enfin, chaque *use case* pourra être alloué aux phases de vie qui lui correspond.

Ainsi, au final, l'ensemble des concepts manipulés et des informations produites lors de l'analyse des besoins est réifié en SysML selon le Tableau II.1suivant :

Tableau II.1 : Représentation SysML des concepts de l'élicitation des besoins

Concept	**Représentation SysML**		
	Diagramme	**Élément**	**Entité graphique SysML**
Système	Contexte	*Block*	«block» **Nom_Block**
Phases de vies		*Block*	«block» **Nom_Block**
Décomposition des phases de vie		*Composition*	1 — 1
Lien Système \ Phase de vie		*Association*	1 — 1
Acteurs extérieurs		*Actor*	Nom_Acteur
Lien Système \ Acteurs		*Association*	1 — 1
Contraintes		*Constraint block*	«constraint» **Nom_contrainte**
Lien Système \ Contraintes		*Aggregation*	1 — 1

Héritage		*Generalization*	
Lien Contraintes \ Phase de vie	Use Case	*Allocation*	
Fonctionnalités principales		*Use Case*	
Inclusion de fonctionnalités		*Include relationship*	
Lien fonctionnalités optionnelles		*Extend relationship*	
Lien Fonctionnalité \ Phase de vie		*Allocation (formel) Frame Box (graphique)*	
Lien Fonctionnalité \ Acteur		*Association / Interaction*	

III. Définition des exigences

1. Description détaillée de l'activité

Cette activité est bien plus renseignée et définie que l'activité d'élicitation des besoins. L'ensemble des normes d'ingénierie système précédemment citées [IEEE 1220] [EIA 632] [IEC 15288] décrit l'activité de définition des exigences. Les informations structurées lors de l'élicitation des besoins sont indispensables pour réaliser la définition des exigences et l'analyse fonctionnelle du système. La définition des exigences est vastement étudiée et donc nous nous appuierons sur les standards d'ingénierie système pour définir les termes employés. Selon [IEEE 1220], les exigences d'un système peuvent être décomposées en 3 catégories : opérationnelle, fonctionnelle et conception. La vue opérationnelle représente la description des exigences quant aux services rendus par le système. Cette vue définit les exigences qui ont attrait aux opérateurs éventuels du système, au cycle de vie du système et avec quelles performances et dans quelles conditions le système doit être utilisé. Ces exigences opérationnelles vont formaliser les attentes préalablement décrites par le diagramme de contexte présenté lors de l'élicitation des besoins. La vue fonctionnelle doit définir la façon de fonctionner du système, le comportement du système, pour rendre les services décrits dans la vue opérationnelle. Enfin la vue conception transcrit les exigences en termes de technologie, en termes d'interface avec d'autres systèmes, avec des COTS et/ou avec des humains. La norme IEEE 1220 décrit le processus de définition sur lequel se base le schéma de la Figure II.3.

Les 15 étapes de définition des exigences détaillées dans cette figure sont associées selon une hiérarchie particulière qui illustre les paliers de précision et raffinement des exigences. En effet, les 4 premières étapes numérotées 1 à 4, correspondent aux exigences émergeant directement de l'élicitation des besoins. Ces étapes seront donc réalisées très tôt et consisteront à une formalisation sous forme d'exigences des attentes formulées lors de l'élicitation des besoins. Les étapes 5 à 8 raffinent les exigences précédentes en dépassant le niveau de détails développé lors de l'élicitation

des besoins. L'étape 9 : « Définir les exigences fonctionnelles » est quant à elle une formalisation en exigences de l'analyse fonctionnelle (que nous décrivons en détail au §IV). Cette interaction directe avec l'analyse fonctionnelle est d'ailleurs schématisée par le cadre : « Vers : Analyse Fonctionnelle ». L'étape 10 : « Définir les exigences de performance », est quant à elle reliée à la description du système. La définition de ces exigences impacte le choix des composants du système. De même, les choix et contraintes de conception peuvent nécessiter une revue de ces exigences. Les dernières étapes, numérotées 11 à 14 consistent en des raffinements des exigences fonctionnelles (9) et des exigences de performances (10). Enfin, l'étape 15 de la Figure II.3 correspond à l'organisation des exigences définies à travers toutes les étapes décrites auparavant selon les trois catégories d'exigences.

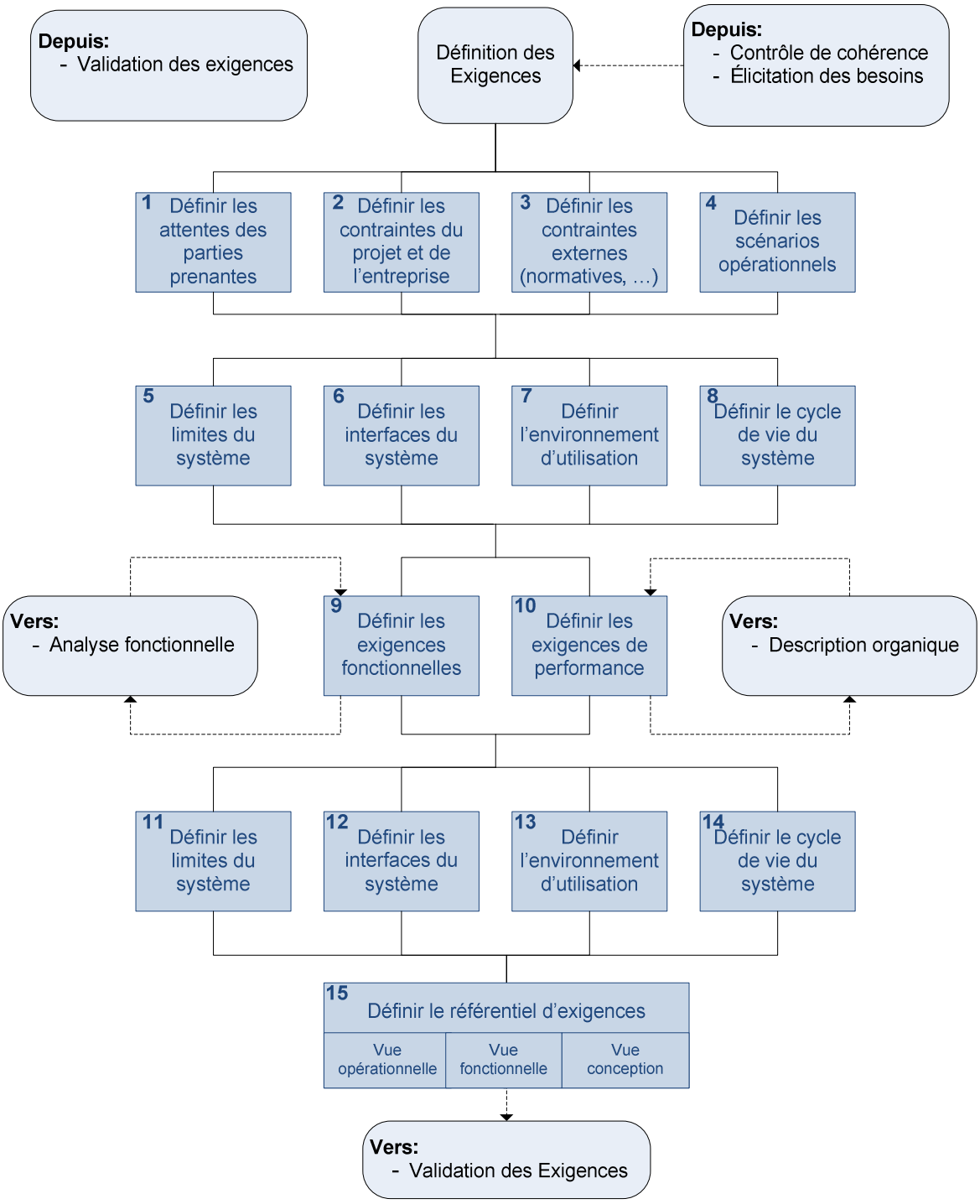


Figure II.3 : Processus de définition des exigences

À noter que la classification des exigences présentée ici est propre à la norme IEEE 1220. Il existe de nombreuses classifications des exigences, la norme EIA632 par exemple propose 33 catégories très spécifiques pour classer les exigences. Ces répartitions sont avant tout présentées et utilisées pour organiser les activités de définition et validation des exigences. Ainsi, quelles que soient les catégories utilisées, ce seront toujours les mêmes mécanismes qui interviendront lors de la définition des exigences : la décomposition, le raffinement et la dérivation [Petin et al. 2010] [Friedenthal et al. 2008]. La décomposition consiste à découper une exigence en 2 ou plus sous-exigences. C'est le mécanisme principal de la définition des exigences. Il permet à partir des besoins exprimés de constituer un référentiel d'exigence d'un détail suffisant pour les autres activités d'ingénierie système : analyse fonctionnelle et définition organique du système. Le raffinement consiste à détailler la définition d'une exigence, par modification de cette exigence ou en lui associant une propriété. En effet, le standard IEEE 1220 et la norme IEC 15288 évoquent la nécessité d'exprimer les exigences le plus clairement possible au moment de leur définition en ayant recours si nécessaire à des propriétés formulées dans un langage métier spécifique : descriptions formelles, algorithmes informatiques, équations mathématiques, schémas électriques, etc. Enfin, la dérivation représente la relation qui relie une exigence A à une exigence B, lorsque B existe du fait du respect de A dans un contexte particulier. Par exemple, lorsqu'une exigence technique existe par application d'une exigence normative, il existe une relation de dérivation entre ces deux exigences.

De même que pour l'élicitation des besoins, des techniques et bonnes pratiques existent pour optimiser le travail de rédaction des exigences [AFIS 2001] afin d'assurer la qualité des exigences rédigées. Cet aspect « qualité des exigences » appelle donc une activité connexe à la définition des exigences : la validation des exigences. Selon les normes d'ingénierie système précédemment citées [IEEE 1220] [EIA 632] [IEC 15288], la validation des exigences consiste en deux aspects distincts :

- Le référentiel complet des exigences doit être cohérent. Les exigences doivent respecter les contraintes définies lors de l'élicitation des besoins, transcrire l'ensemble des besoins élicités, …
- Le référentiel des exigences en tant que partie du projet doit être cohérent avec le reste des informations du projet.

Le premier aspect représente une sous-activité de la définition des exigences, en général réalisée en fin de processus. Le second est quant à lui lié à une activité de contrôle de cohérence de l'ensemble du projet qui vérifiera que les exigences sont effectivement respectées dans le reste de la spécification du système. Cette étape correspond plus à une validation du système par rapport aux exigences qu'à une étape de validation des exigences.

La validation des exigences correspond donc principalement au premier aspect décrit ci-dessus. Le schéma de la Figure II.4 issu de la norme IEEE 1220 représente justement la validation des exigences par rapport aux besoins élicités par les parties prenantes.

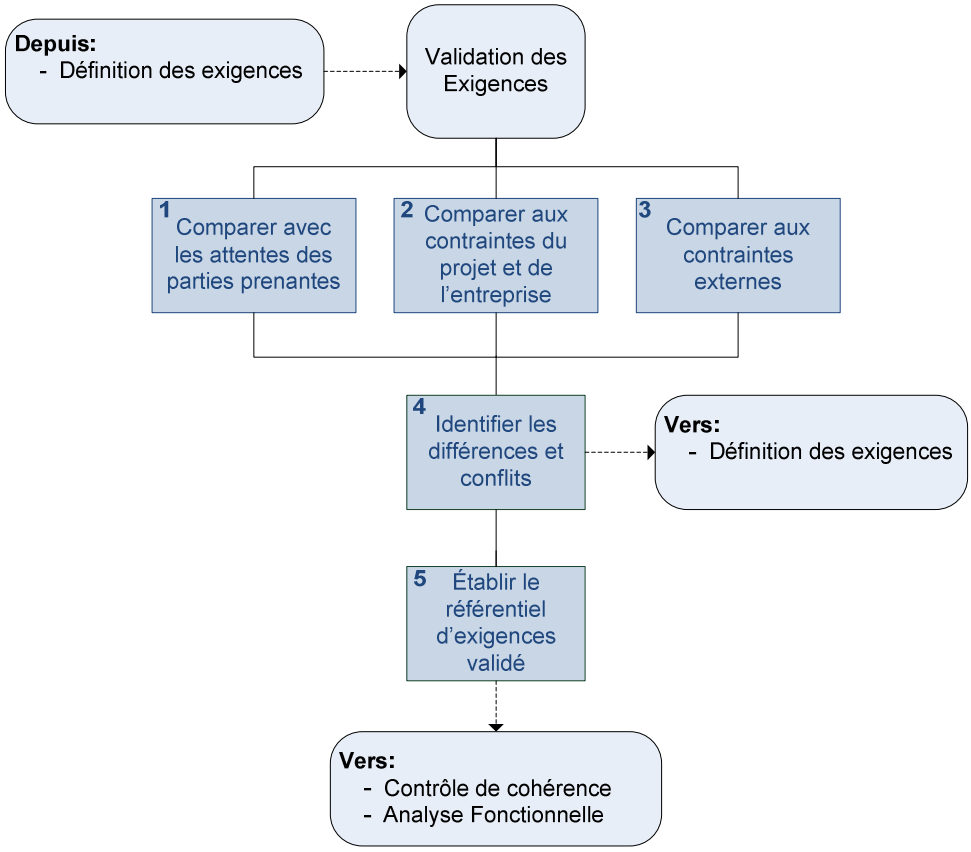


Figure II.4 : Processus de validation des exigences

Les exigences définies sont en entrée des activités de validation : « Depuis : Définition des exigences ». En cas de conflits, les activités de validation bouclent sur les activités de définition d'exigences : « Vers : Définition des exigences ». Après un nombre suffisant d'itérations entre les activités de définitions des exigences et de validation, les exigences sont validées unitairement et alors le processus continue vers les activités de contrôle de cohérence de l'ensemble des exigences entre elles et vis-à-vis du reste du projet : « Vers : Contrôle de cohérence ». L'importance du lien qui existe entre les exigences et les autres entités qui permettent de décrire notre système est telle qu'il est nécessaire de spécifier ces liens, ne serait-ce que pour faciliter les activités de contrôle et de validation. Deux types de relations peuvent alors intervenir :

- Relation de satisfaction : cette relation est utilisée pour définir les entités qui respectent une exigence particulière. Par exemple : une roue de bicyclette peut satisfaire à une exigence opérationnelle de diamètre de roue.
- Relation de vérification : cette relation est utilisée pour définir les entités qui permettent de tester le respect d'une exigence. Par exemple : la description d'un test de puissance électrique permet la vérification d'une exigence de performance d'une pile à combustible.

Ces relations permettent donc la réalisation des activités de validation des exigences vis-à-vis de l'ensemble du projet. La validation pourra notamment contrôler que toutes les exigences sont satisfaites et vérifiables.

2. Réification de cette activité en SysML

Plusieurs travaux [Petin et al. 2010] [Friedenthal et al. 2008] [Holt & Perry 2008] étudient la problématique de la gestion des exigences en SysML. En nous appuyant sur leurs travaux et sur le standard SysML, nous pouvons définir les mécanismes SysML nécessaires et suffisants à la réalisation des activités de définition et validation des exigences décrites auparavant. Le standard SysML [OMG 2012] contient un élément de modélisation propre à la représentation d'une exigence et une panoplie de mécanismes dédiés. On retrouve les mécanismes décrits précédemment : la décomposition, le raffinement, et la dérivation. Le Tableau II.2 présente les représentations graphiques en SysML des exigences et des relations utiles.

Tableau II.2 : Représentation SysML des concepts de définition des exigences

Concept	**Représentation SysML**		
	Diagramme	**Élément**	**Entité graphique SysML**
Exigence	Requirement	*Requirement*	«requirement» Nom de l'exigence
Décomposition		*Containment*	
Raffinement	Multiple	*Refine*	«refine»
Dérivation		*DeriveReqt*	«deriveReqt»
Satisfaction		*Satisfy*	«satisfy»
Vérification		*Verify*	«verify»

L'exigence elle-même est représentée sous la forme d'un rectangle portant l'en-tête *«requirement »* ainsi que le nom de l'exigence. Une exigence en SysML possède deux propriétés principales : *text* et *id*. La propriété *text* correspond à l'intitulé de l'exigence en langage naturel. La propriété *id* permet d'attribuer à chaque exigence un identifiant unique pour permettre sa traçabilité. Cet *id* est notamment utilisé par les différents outils logiciels de modélisation SysML pour permettre une interaction avec des outils d'ingénierie des exigences comme DOORS [DOORS] ou Reqtify [Reqtify]. La relation de décomposition présente entre deux *requirements*, permet de structurer la hiérarchie des exigences. La relation *refine* correspond au besoin de décrire une exigence avec des entités diverses ne relevant pas de la pure définition des exigences. Par conséquent, la flèche SysML représentant la relation *refine* relie toujours une exigence à la pointe et n'importe quel autre élément qui raffine l'exigence à l'origine (techniquement, il est possible que la pointe soit connectée à une autre exigence). La relation de dérivation permet de mettre en avant la connexion de deux exigences. La relation SysML *deriveReqt* connecte donc deux *requirements* : l'origine de la flèche représente l'exigence qui existe à cause de l'exigence qui est à la pointe de la flèche. La satisfaction des exigences est représentée par une flèche *satisfy* dont l'origine représente n'importe quelle entité qui satisfait l'exigence qui se trouve à la pointe de la flèche. Enfin, la relation de vérification relie n'importe quelle entité à l'origine de la flèche qui permet la vérification de l'exigence sur laquelle pointe la flèche. L'ensemble des relations représenté par des flèches peuve

aussi être représenté par des cartouches rattaché à chacun des éléments. Ainsi une relation de type *refine* devient deux cartouches : « RefinedBy » et « Refines ». Ce principe peut aussi s'appliquer aux relations de type *deriveReqt*, *satisfy* et *verify*. De plus, ces cartouches peuvent aussi être intégrés à l'élément concerné sous forme de propriétés. Un exemple des ces 3 formes est présenté en Figure II.5.

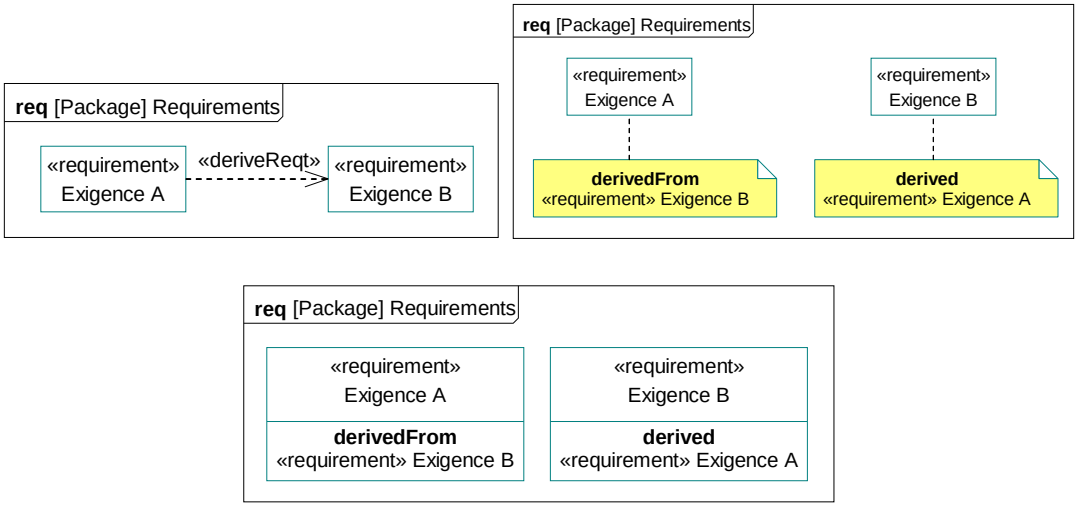


Figure II.5 : Différentes représentations de la relation deriveReqt

Le standard SysML propose une extension non normative contenant un ensemble de stéréotypes dédiés à la définition des exigences. Comme pour le diagramme de contexte, cet exemple fourni par le standard sort du cadre standardisé de la norme SysML mais permet toutefois d'établir une nécessité de spécifier et d'organiser les différents types d'exigences que contient un projet. De façon similaire aux principes utilisés pour l'élicitation des besoins, nous pouvons utiliser les mécanismes de décomposition de façon à remplacer les stéréotypes (Figure II.6). L'organisation des exigences par package comme cela est proposée par [Friedenthal et al. 2008] permet aussi de limiter l'utilité de stéréotype et est compatible avec l'organisation par décomposition proposée en Figure II.6.

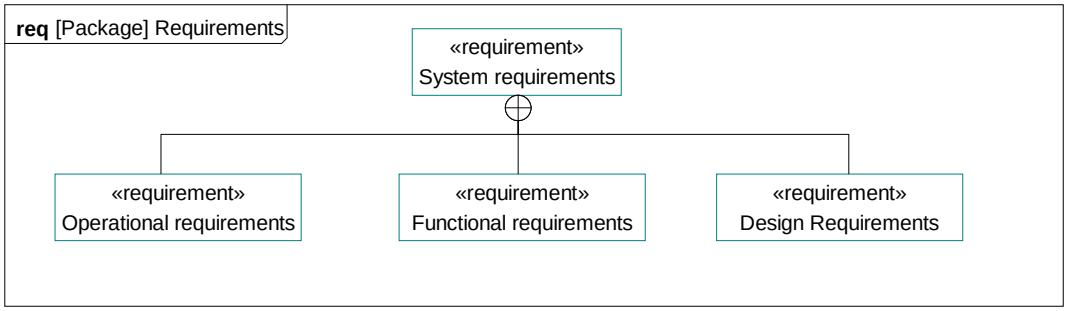


Figure II.6 : Organisation des types d'exigences par décomposition

IV. Analyse Fonctionnelle

1. Description détaillée de l'activité

L'analyse fonctionnelle est une pratique assez répandue, cependant elle est souvent associée aux activités de gestion de risque et des études de FMDS (Fiabilité, Maintenabilité, Disponibilité, Sécurité). En effet, la plupart des standards cités précédemment n'évoquent pas l'analyse fonctionnelle [EIA632] [IEC15288] [INCOSE 2010], ou alors ne décrivent que l'aspect description des interfaces logiques des composants [NASA 2007]. Cependant, la norme IEEE 1220 [IEE1220] décrit l'analyse fonctionnelle à travers un processus dédié et plusieurs sous-activités (Figure II.7).

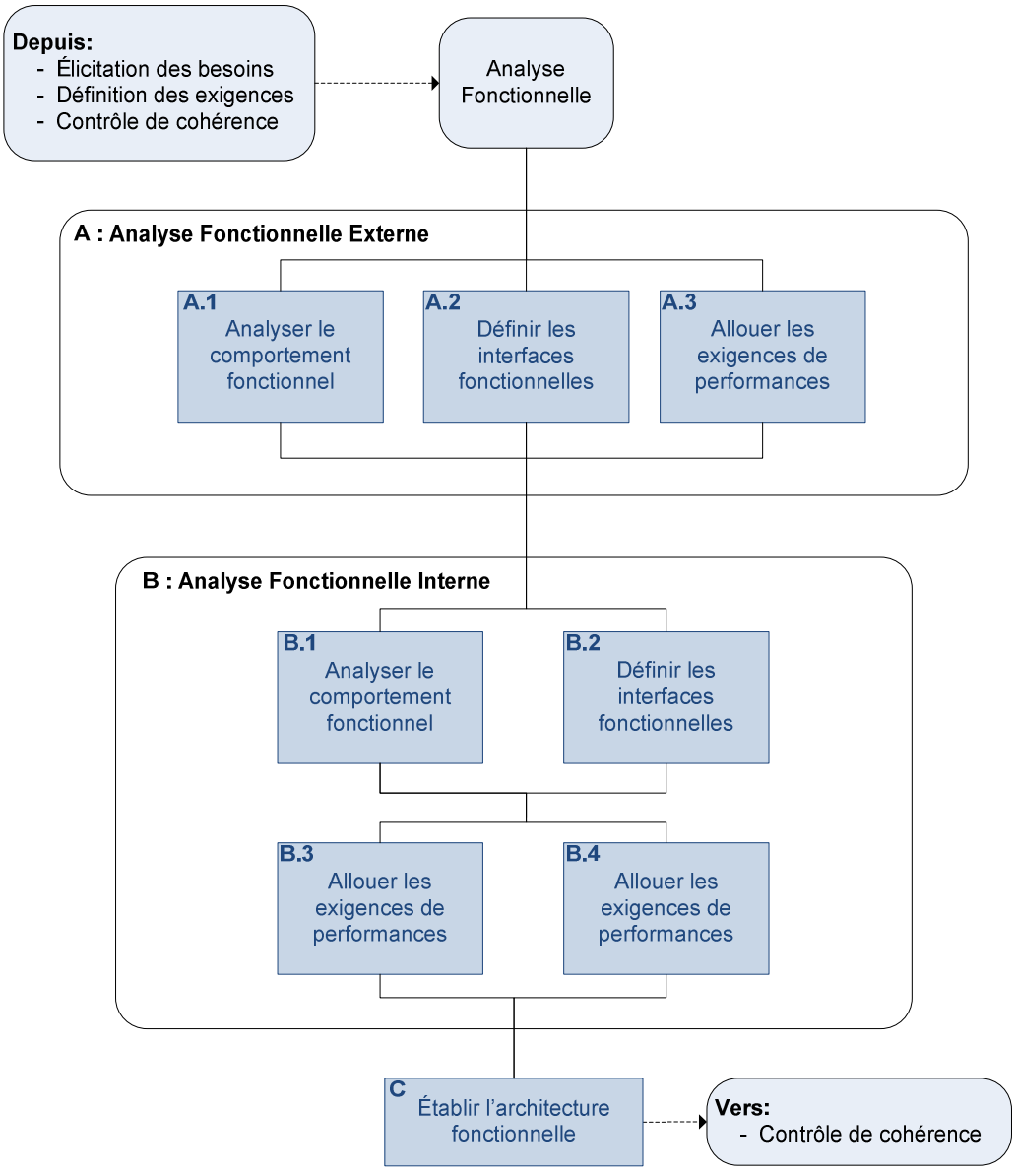


Figure II.7 : Processus d'analyse fonctionnelle

Il est intéressant de détailler les deux grandes phases qui composent ce processus :

- Analyse Fonctionnelle Externe. Cette étape se concentre sur la définition des fonctions au niveau système et des interfaces logiques de ces fonctions avec l'extérieur du système. Ces

activités doivent être cohérentes avec les exigences fonctionnelles décrites lors de la définition des exigences et ainsi, il est nécessaire de connecter les activités d'analyse fonctionnelle externe avec la définition des exigences. De plus, une sous-activité d'allocation des exigences de performance aux fonctions du système est ajoutée pour permettre la traçabilité des exigences.

- Analyse Fonctionnelle Interne. Cette étape va permettre de raffiner les fonctions de niveau système en sous-fonctions. Chaque sous-fonction sera décrite de façon plus complète que lors de l'analyse fonctionnelle externe. Les interfaces sont définies comme précédemment. Les flux connectant les sous-fonctions sont précisés et détaillés (type de flux : logique, physique,…). Le comportement interne des sous-fonctions et vis-à-vis les unes des autres est lui aussi décrit (états de fonctionnement, échange de données, logique d'échange, …). Par raffinement successif des fonctions et sous-fonctions, l'objectif est d'atteindre un niveau de détails correspondant aux fonctions d'un composant physique. Une relation de décomposition fonctionnelle similaire à la relation de décomposition des exigences peut être introduite pour faciliter la traçabilité des relations hiérarchiques existantes entre les fonctions et sous-fonctions.

En considérant le processus proposé jusqu'à présent, l'analyse fonctionnelle se déroule alors que les activités d'élicitation des besoins et de définition des exigences ont été initiées. Ainsi les besoins et les exigences fonctionnelles constituent les entrées de ces activités. La continuité et la cohérence de toutes ces activités entre elles doivent être assurées par des correspondances entre les différentes entités manipulées. Nous avons déjà présenté les relations propres aux exigences qui permettent de valider et contrôler celles-ci (satisfaction, vérification, …). Au cours de l'analyse fonctionnelle, une nouvelle relation utile à la traçabilité est introduite : l'allocation fonctionnelle. Cette relation permet de décrire le lien existant entre la fonction définie lors de l'analyse fonctionnelle et le composant décrit lors de l'activité de description organique qui exécute cette fonction. Une fonction peut être allouée à plusieurs composants et plusieurs fonctions peuvent être allouées à un unique composant. Le niveau de détail lors de la description des fonctions d'un système peut varier d'un utilisateur à l'autre. Trois cas sont observés :

- Une fonction du système est une entité plus englobante que l'unité organique. Une fonction du système sera donc réalisée par un ensemble de composants.
- La fonction représente l'unité fonctionnelle et doit correspondre à l'unité organique : le composant. Ainsi, on s'efforce dans ce cas de tendre vers : « une fonction par composant ».
- Les composants du système ne sont pas aussi détaillés que les fonctions et alors plusieurs fonctions sont allouées à un seul composant.

En général, le point numéro deux correspond à un idéal à atteindre et ponctuellement des exceptions apparaissent pour certaines fonctions particulières ou pour certains composants. Notamment, la plupart des COTS utilisés dans l'industrie sont des composants dont la description détaillée n'est pas disponible, embarquant la possibilité d'exécuter plusieurs fonctions.

2. Réification de cette activité en SysML

La représentation graphique recommandée par le « NASA System Engineering Handbook » [NASA 2007] pour cette activité d'analyse fonctionnelle est l'eFFBD (enhanced Functional Flow Block

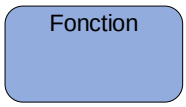
Diagram). De même [Long 2002] décrit l'eFFBD comme étant l'outil graphique le plus représentatif de cette activité d'analyse fonctionnelle. Or, les diagrammes d'eFFBD [Seidner 2009] sont très proches des *Activity Diagrams* (AD) de SysML, aussi bien dans leur représentation que dans les possibilités de représentations qu'ils offrent. Ainsi, il est assez aisé de réaliser l'activité d'analyse fonctionnelle en utilisant les AD de SysML. Divers travaux évoquent aussi la possibilité d'effectuer cette analyse fonctionnelle en utilisant des IBD. Cependant cette solution, même si elle est viable en théorie, s'avère délicate d'utilisation, car ce type de diagramme est en général utilisé pour la description de l'architecture organique. En effet, l'utilisation d'un même élément syntaxique pour des concepts différents impose une sémantique plus complexe et donc plus délicate à visualiser pour le modélisateur et le lecteur du modèle. Bien que des artefacts graphiques puissent être ajoutés à l'utilisation de certains logiciels de modélisation, cela ne constitue pas une solution viable dans la plupart des cas, comme l'évoquent [Pétin et al. 2010].

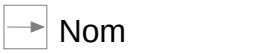
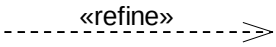
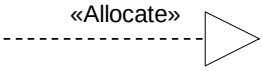
Ainsi nous considérons l'*Activity Diagram* de SysML comme étant le meilleur choix pour la description fonctionnelle. L'entité principale qui représente la fonction dans un *AD* est l'*opaqueAction*. Chaque *opaqueAction* peut posséder des *pins* d'entrée et de sortie : *InputPin* et *OutputPin*. Le nom d'un *pin* définit le nom du flux fonctionnel qui sera véhiculé à travers ce *pin*. Les fonctions d'un même niveau hiérarchique sont alors connectées par des flux fonctionnels établis entre des *pins* d'entrée et de sortie. La nature de l'objet SysML qui relie un *pin* d'entrée et un *pin* de sortie est un *object flow*. Pour modéliser les différents niveaux hiérarchiques de l'analyse fonctionnelle, une fonction peut elle-même être définie par un AD organisant ces sous-fonctions. La relation SysML entre l'*opaqueAction* mère et l'AD est une relation de raffinement, similaire à celle utilisée lors de la définition des exigences. La relation *refine* réifie différentes relations selon les entités auxquelles elle est reliée. Lors de la modélisation, il faut s'assurer que les entrées et sorties d'une fonction mère sont respectées lors de la description de son contenu à l'aide d'un AD. Ainsi les *InputPins* et *OutputPins* de l'*opaqueAction* mère doivent correspondre à des *InputPins* et *OutputPins* en bordure de l'AD (les *pins* de bordure d'un AD indiquent des flux fonctionnels qui s'étendent hors du périmètre représenté par l'AD).

La relation de *refine* sert aussi à lier un besoin système décrit par un *use case* lors de l'élicitation des besoins et l'un des AD de plus haut niveau, pour initier la hiérarchie fonctionnelle (Figure II.8). Un *use case* ne possédant pas de *pin*, la règle précédente ne s'applique pas dans ce cas particulier.

Enfin, l'allocation de la fonction à un ou plusieurs composants est réalisée en SysML grâce à la relation *allocate* déjà évoquée lors de l'élicitation des besoins. La relation *allocate* relie donc dans ce cas un *opaqueAction* avec un *part* (les *parts* seront détaillés dans la partie description organique). Le Tableau II.3 résume l'ensemble minimal des entités SysML utiles à l'analyse fonctionnelle.

Tableau II.3 : Représentation SysML des concepts de l'analyse fonctionnelle

Concept	**Représentation SysML**		
	Diagramme	**Élément**	**Entité graphique SysML**
Fonction	Activity Diagram	*Opaque Action*	

Interfaces : Entrées et Sorties		*Input Pin* *Output Pin*	
Nom du flux fonctionnel		*Pin Name*	
Flux fonctionnel		*Object Flow*	
Hiérarchie fonctionnelle	Multiple	*Refine*	
Allocation Fonctionnelle		*Allocate*	

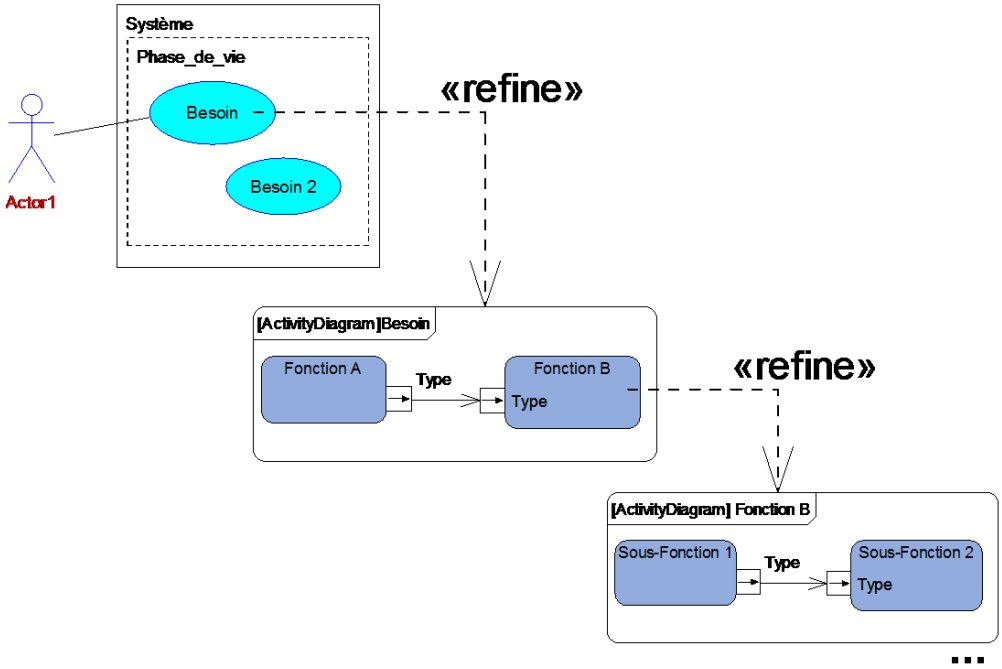


Figure II.8 : Organisation de la hiérarchie fonctionnelle

V. Description Organique

1. Description détaillée de l'activité

La description organique est la dernière étape de spécification d'un système. Après cette étape, des activités de contrôle, cohérence et traçabilité peuvent encore s'ajouter, mais l'ensemble de la définition de notre système est réalisé lors de ces 4 étapes : Elicitation du besoin, définition des exigences, analyse fonctionnelle et description organique.

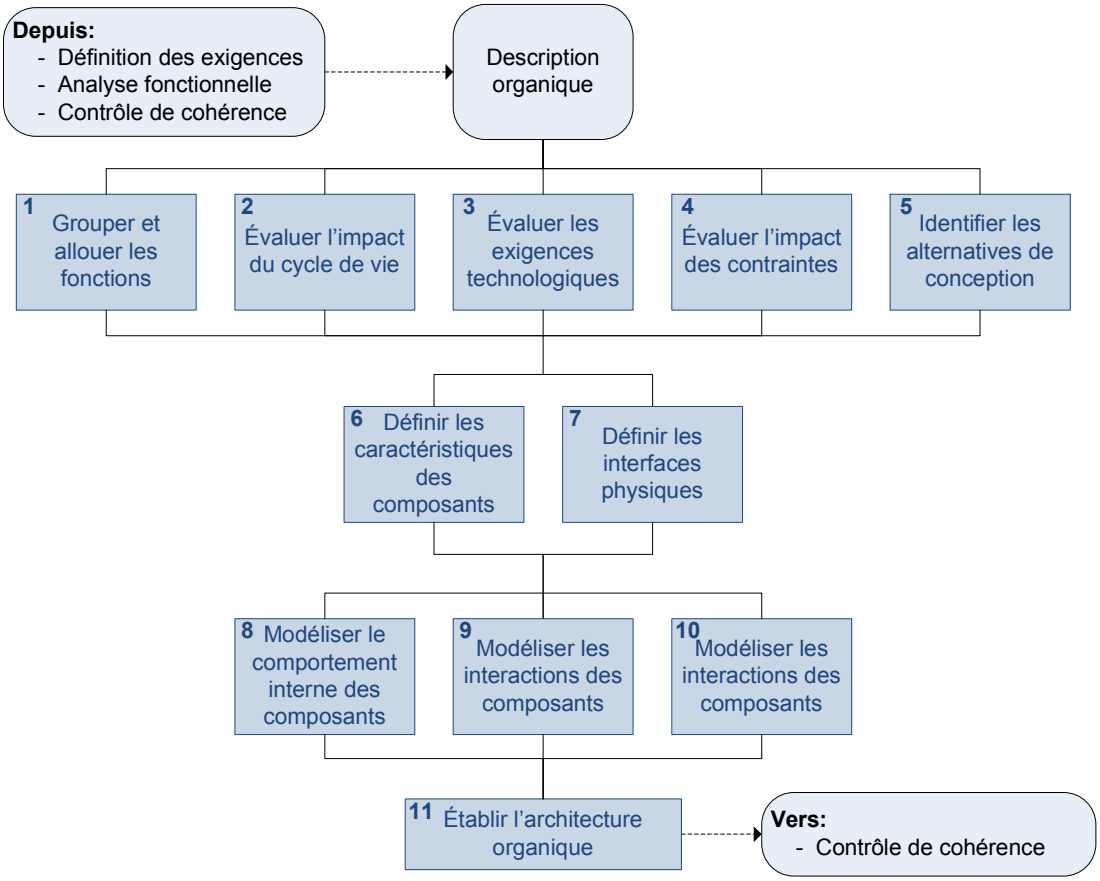


Figure II.9 : Processus de description de l'architecture organique

Le processus décrit par la Figure II.9 est inspiré du standard IEEE1220 [IEEE 1220]. Ce processus décrit les différentes activités permettant, à partir des informations produites par les étapes précédentes d'ingénierie système, l'établissement de l'architecture physique et logique du système. Cette activité de description organique est sans doute la plus conséquente des étapes considérées jusqu'à présent. C'est lors de ce processus qu'un grand nombre de paramètres doivent être pris en compte. Ainsi la plupart des activités présentées dans la Figure II.9 sont des activités effectuées en parallèle et de façon itérative afin de pouvoir aboutir finalement à une architecture répondant à tous les critères. Parmi ces activités, nous pouvons percevoir deux catégories : les activités de prise en compte des besoins fonctionnels, des exigences et des contraintes (1 à 4) et les activités de choix d'architecture (5 à 10).

Quand ces activités sont réalisées, l'architecture organique peut alors être spécifiée dans un formalisme quelconque, ce qui est l'objet de l'activité 11. Notons que les activités 8 à 10 correspondent elles aussi à des activités de formalisation de l'architecture, mais ces activités ne visent pas à la spécification, mais plutôt à l'analyse spécifique d'un composant ou de l'interaction de certains composants entre eux pour permettre la vérification. Le processus de la Figure II.9 prévoit aussi la possibilité que des changements de spécification interviennent au cours d'un projet c'est pourquoi à partir de l'activité 11, une connexion existe avec le processus de contrôle qui permet la gestion des changements de spécification. Ce processus de contrôle ne sera pas détaillé. Cependant, il faut noter que ce processus peut impliquer un retour aux étapes d'élicitation des besoins, de définition des exigences ou même d'analyse fonctionnelle.

Les informations attendues lors de la description organique du système sont diverses. A minima, les types de composants et composants sont décrits (activité 5). Les interfaces de chacun de ses composants sont spécifiées : nom et nature des échanges possibles avec chaque composant (activité 7). Le fonctionnement de chaque composant est défini, mais non détaillé (activité 6). À noter que les fonctions unitaires réalisées par le composant physique seront appelées « opérations » pour les distinguer des fonctions du système décrites lors de l'analyse fonctionnelle qui ne sont pas forcement corrélées avec les opérations, comme évoqué précédemment à propos de l'allocation fonctionnelle dans la partie Analyse Fonctionnelle.

Notons que ces étapes correspondent à la description d'un composant au niveau Boite Noire : seules ces interfaces et la définition de ses opérations sont connues. C'est en général à ce niveau que la description d'un COTS se termine.

Cependant, pour les composants dont nous possédons une connaissance plus grande, ils pourront être détaillés et chaque interaction et flux entre composants pourra l'être aussi. Plusieurs niveaux de détails peuvent être utiles/nécessaires et la nature des informations utilisées peut varier. Les composants peuvent être décomposés en ensemble de sous-composants. Pour les interactions entre les composants (à différents niveaux de détails), on distinguera tout de même trois principales possibilités :

- Description détaillée de la logique interne d'un composant,
- Description détaillée des interactions entre composants,
- Description des contraintes appliquées à un ou plusieurs composants.

L'ajout de ces informations détaillées correspond aux activités 8 à 10.

En parallèle à ces activités de choix d'architecture, les activités de traçabilité sont réalisées. L'activité 1 consiste à réaliser l'allocation fonctionnelle : allouer les fonctions systèmes au(x) composant(s) qui les réalise(nt). De même, l'activité 3 renseigne le lien de satisfaction qui doit exister entre les exigences techniques et les composants. Des relations de satisfaction doivent aussi exister entre les exigences d'interfaces et les interfaces de chaque composant ce qui pourra être renseigné ou complété à l'activité 7 et entre les exigences environnementales et les contraintes appliquées aux composants ce qui sera renseigné aux activités 2 et 4.

Quand ces activités sont terminées, l'architecture du système est finalisée en connectant entre eux les différents composants et modules, et une première définition du système est terminée. En accord avec le processus global d'Ingénierie Système présenté dans le chapitre I, la seule activité non détaillée est l'activité de contrôle de cohérence du système. Cette activité ne sera pas détaillée, car elle n'impacte pas ou peu les processus MéDISIS que nous décrivons dans la suite de la thèse. Cependant, l'activité de contrôle de cohérence est décrite abondamment dans la littérature [EIA 632] [IEC 15288] [IEEE 1220] [NASA 2007]. L'activité de contrôle de cohérence est facilitée par le renseignement des relations de traçabilité disséminées au travers de l'ensemble des activités d'IS. L'ensemble de ces relations est représenté dans la Figure II.10.

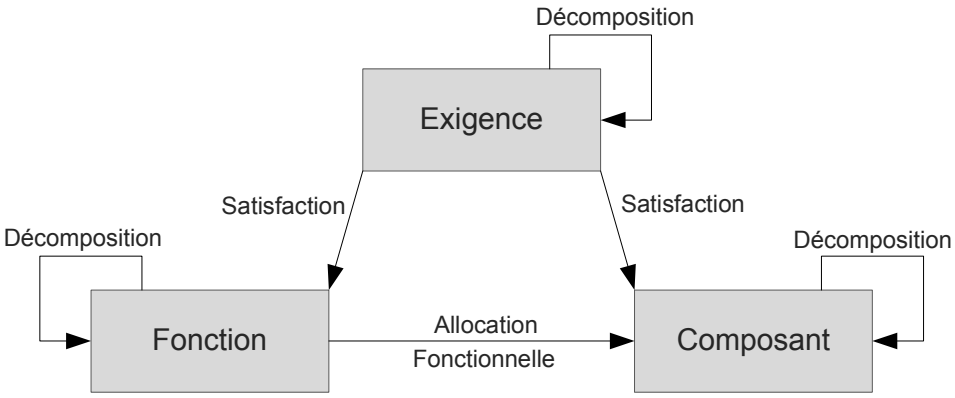


Figure II.10 : Relations de traçabilités entre les éléments du modèle système

Voyons maintenant comment cette activité de description organique peut être réifiée avec le langage système.

2. Réification de cette activité en SysML

Comme nous l'avons vu précédemment, la description organique peut se décomposer en deux parties :

- Définition des composants (nom et interface) et organisation architecturale des composants entre eux.
- Description détaillée des composants et de leurs échanges.

Nous allons décrire comment utiliser les éléments de la syntaxe SysML afin de définir et raffiner les composants du système et les relations entre eux.

2.1. Définition et organisation des composants

La définition et l'organisation des composants est l'activité la plus renseignée et référencée des activités d'ingénierie système [Friedenthal et al. 2008] [Roques 2009b]. Ainsi, lors de la mise au point de SysML, deux types de diagrammes ont été prévus spécifiquement pour le cœur de cette activité : Les *Block Definition Diagram* (BDD) et les *Internal Block Diagram* (IBD) [OMG 2012].

En SysML, deux entités vont être utilisées pour définir les composants : les *blocks* et les *parts*. Les *blocks* correspondent à des types de composant et les *parts* à des composants physiques uniques. Par exemple, on pourra décrire un type « Roue » modélisé en SysML par un *block* et pour modéliser une bicyclette, nous trouverons 2 *parts* de type « Roue » : « Roue avant » « Roue arrière ». Le nom d'un *part* est aussi appelé *rôle*, car la « Roue avant » est une « Roue » qui occupe le rôle de « Roue avant ». En SysML, le nom d'une entité est souvent suivi de « : » et du nom de son type. Dans notre exemple, il sera écrit : « RoueAvant : Roue ». Pour modéliser l'organisation des composants, il existe deux relations en SysML : la relation de composition : *PartAssociation* et la relation d'agrégation : *SharedAssociation*. La composition est utilisée pour représenter les entités qui constituent le *block* étudié. L'agrégation est utilisée pour représenter les entités qui sont partagées avec d'autres *blocks*. Ces relations possèdent pour chacune de leurs extrémités une multiplicité. La multiplicité représente le nombre de composants qui sont considérés dans la relation. La multiplicité

peut spécifier un nombre ou une plage de nombre de composants considérés. Dans l'exemple Figure II.11, on voit qu'une bicyclette possède aucun ou plusieurs portes-bagages : 0..* , alors qu'une Bicyclette possède exactement deux Roues : 2. Côté *block* parent, la multiplicité par défaut est 1, indiquant que la décomposition présentée correspond à un composant. Le sens sémantique de toutes autres multiplicités côté *block* parent devra être défini.

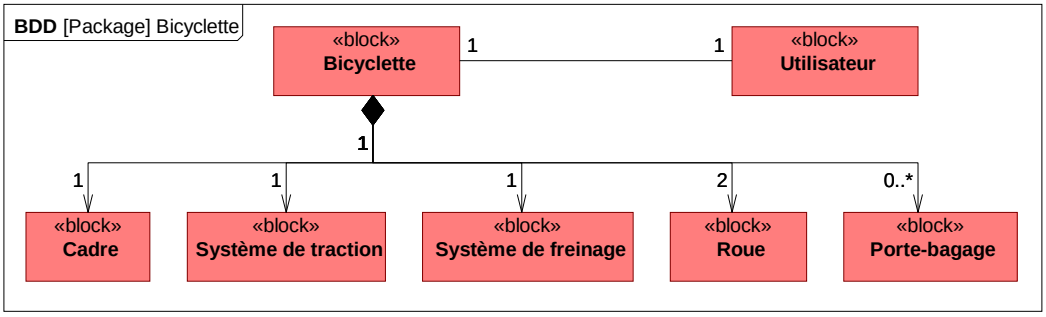


Figure II.11 : Block Definition Diagram: Bicyclette

Il existe aussi une relation d'association simple : *Association* qui permet de faire état de relation entre *blocks* qui ne relève pas de la décomposition organique, par exemple la relation entre un « Utilisateur » et la « Bicyclette » (Figure II.11). Cette relation d'association simple sera aussi utilisée pour représenter la relation d'un *block* avec un acteur extérieur (les mêmes *actor* que ceux présentés lors de l'élicitation des besoins). Dans ce cas, le *block* « Utilisateur » pourrait être remplacé par une entité *actor*.

Pour chaque type de composant, il est possible de définir des propriétés : *properties* qui lui sont propres. Quatre types de *property* sont utilisables :

- Les *value properties* sont les propriétés simples et valuables, par exemple « taille = 12 » (taille pouvant lui-même être typé, notamment pour indiquer l'unité et la dimension de la *value property*).
- Les *parts* sont les composants qui composent ce type de composant, par exemple, un *block* bicyclette possédera 5 *parts* : « Cadre », « Roue avant », « Roue Arrière », « Système de freinage » et « Système de traction » (et éventuellement un ou plusieurs *part*s « Porte-Bagage »). Ces propriétés doivent être cohérentes avec les relations d'agrégation et de composition précédentes.
- Les *constraint properties* permettent de relier les *value properties* de différent *block*. Ces *constraint properties* permettent d'exprimer les contraintes qui impactent les propriétés des composants. Ces *constraints properties* intègrent des relations mathématiques ou logiques pour spécifier ces contraintes.
- Les *ports* permettent de définir les interfaces (logiques, physiques, électriques,…) d'un *block*.

Chaque propriété est typée. Les *parts* le sont par d'autres *blocks*. Les *value properties* et les *ports* sont typés par des *value types*. Les *constraint properties* sont typées par des *constraint blocks*. *Value type* et *constraint block* sont tous deux des formes spécialisées de *block*. Ainsi le *value type* représente un type de *value property* comme le *block* représente un type de *part*. Dans l'exemple de la Figure II.12, on distingue que le *value type* se distingue du *block* car il possède deux propriétés qui lui sont propres : la dimension (*dimension*) définissant la nature de la mesure de ce type de

propriété et l'unité (*unit*) utile à la mesure de ce type de propriété. Sur la Figure II.12, le *value type* « Diamètre » est utilisé pour typer la propriété « D » du *block* « Roue ». Dans l'exemple la valeur par défaut de D est 35, cette valeur pourra être modifiée pour chaque *part* de « Roue ».

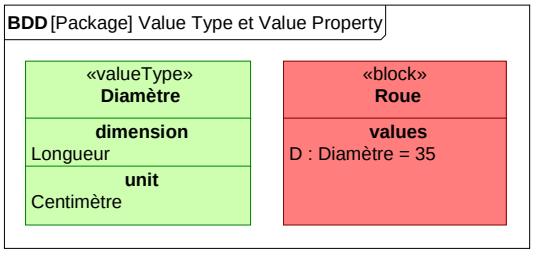


Figure II.12 : Value Type et Value Property

Les *blocks* peuvent aussi posséder des *opérations*. Ces *opérations* servent à définir les actions que le composant peut réaliser. Il s'agit en fait d'une fonction technique du composant. Ces *opérations* se définissent classiquement par un nom et des paramètres d'entrée et de sortie. Ces paramètres correspondent à des *value properties* du *block* ou à des *ports* du *block*. Le détail du fonctionnement d'une opération peut être réifié à l'aide d'un *constraint block*. Il n'existe pas obligatoirement une relation de cohérence entre ces *opérations* et les fonctions décrites au niveau de l'analyse fonctionnelle. Cependant si cela semble nécessaire, il est possible d'utiliser une relation *AllocatedTo* des *opérations* aux *opaqueActions* qui définissent les fonctions de bas niveau lors de l'analyse fonctionnelle. Cette relation d'*allocation* permet alors de réifier les *opérations* qui réalisent une fonction. Ce mécanisme d'*allocation* peut être utilisé en parallèle ou en remplacement des *allocations* composant/fonctions.

Une dernière relation existe pour hiérarchiser les types de composants (*blocks*) et les type de propriétés (*value types*), il s'agit de la relation de généralisation/héritage. Cette relation permet de définir un héritage de propriétés entre deux *blocks*. Par exemple, un *block* « Roue Cloutée » hérite du *block* « Roue », alors « Roue Cloutée » possédera au minimum les mêmes propriétés que « Roue ». Dans notre cas, « Roue Cloutée » possédera donc une *value property* « D » de type « Diamètre », cependant la valeur par défaut de D pourra être différente de celle de « Roue ». De plus « Roue Cloutée » pourra aussi définir des propriétés qui lui sont propres, comme la densité de clous. Cette relation d'héritage telle qu'elle est décrite en SysML est appelée *generalization* et elle relie le *block* fils au *block* père sous forme d'une flèche avec une pointe de flèche blanche (dans notre exemple Figure II.13, la flèche part de « Roue Cloutée » et pointe vers « Roue »).

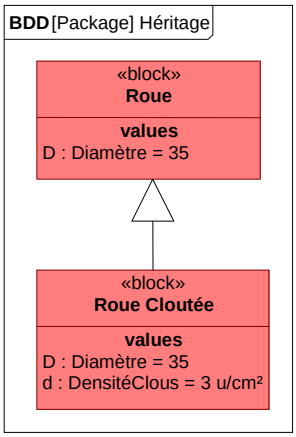


Figure II.13 : Relation d'héritage

Pour organiser les *blocks* et les *value types*, le diagramme utilisé est le BDD. Bien que cela ne soit pas décrit dans le standard SysML, nous recommandons de séparer la représentation des *value types* et la décomposition organique des *blocks*. La déclaration des types de données manipulées au cours d'un projet sera faite dans un même diagramme : « Dictionnaire de données ». La décomposition des types de composants pourra par contre s'étendre sur plusieurs diagrammes par souci de clarté de la représentation.

Le Tableau II.4 résume les représentations graphiques des entités SysML évoquées jusqu'à présent dans cette partie.

Tableau II.4 : Représentation SysML des concepts de la description organique (définitions et interfaces)

Concept	**Représentation SysML**		
	Diagramme	**Élément**	**Entité graphique SysML**
Type de Composant	Block Definition Diagram	*Block*	«block» **Nom_Block**
Type de Propriété		*Value Type*	«valueType» **Nom_Value_Type** **dimension** Nom_Dimension **unit** Nom_Unit
Propriété		*Value Property*	«block» **Nom_Block** **values** Nom_Property : Nom_Value_Type = default_Value
Fonction technique		*Operation*	«block» **Nom_Block** **operations** Opération (in Paramètre1, in Paramètre2, out Résultat)
Décomposition organique « forte »		*Composition*	1 1
Décomposition organique « faible »		*Aggregation*	1 1
Héritage		*Generalization*	

Avec un BDD, notre système est défini de façon générique. Ce niveau de détails permet déjà, à un expert de SdF, d'initier des études de types APR (cf. Annexe 1). Cependant, les multiplicités des relations de compositions peuvent représenter des plages de valeurs, ce qui implique donc une multitude d'alternatives possibles. C'est pourquoi la description détaillée de l'architecture du système étudié doit être réalisée avec un autre type de diagramme présentant les composants et non plus les types de composants : l'*Internal Block Diagram*. L'IBD est donc un diagramme qui représente les composants, c'est-à-dire les *parts*. Un IBD doit être cohérent avec le ou les BDD déjà

définis, et tout particulièrement les multiplicités représentées sur les liaisons de composition (et agrégation). En reprenant l'exemple de la Figure II.11, l'IBD de la bicyclette devra présenter 1 *part* de type « Cadre », 1 *part* de type « Système de traction », 1 *part* de type « Système de freinage », 2 *parts* de type « Roue » et 0 ou plus *part* de type « Porte-Bagage ». Quand on considère un système important possédant un grand nombre de sous-composants, il est envisageable que différents IBD représentent différents points de vue et se complètent. Ainsi, chaque IBD sera exempté de représenter tous les *parts*. Il est aussi possible dans l'IBD de modéliser les relations des sous-composants avec les composants externes au système. Cependant, ces relations doivent être décrites au niveau BDD par une association et la multiplicité de l'association doit être respectée au niveau IBD, selon la syntaxe de SysML. Lorsque des composants externes sont représentés, nous conseillons de représenter le système englobant les sous-composants pour permettre une identification visuelle rapide du périmètre du système (Figure II.14).

Les IBD représentant un choix d'implémentation du BDD, il est aussi nécessaire de valuer les *value properties* définies au niveau BDD. Enfin, la dernière étape de complétion d'un IBD consiste à modéliser les flux échangés entre les composants. Ces flux existent entre deux *flowport* [SysML 2010]. Ces *flowports* sont des propriétés des composants qui sont typés par des *value types* pour définir la nature de l'échange. Chaque *flowport* possède une direction : *in*, *out* , *inout*. Cette direction définit dans quel sens l'échange se produit (entrant, sortant ou les deux). La connexion de deux *flowports* par un flux sera possible si les *flowports* sont bien du même type et si leurs directions sont cohérentes. La Figure II.14 représente un IBD de la bicyclette qui était spécifiée dans le BDD de la Figure II.11.

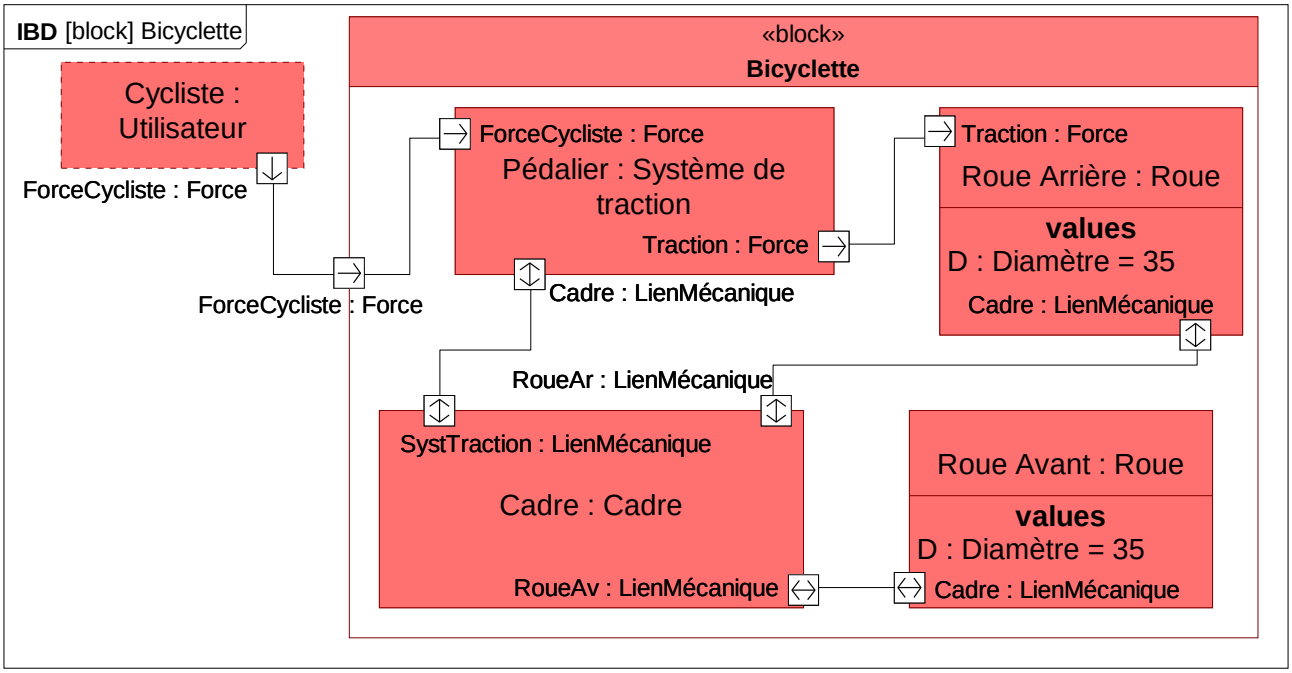
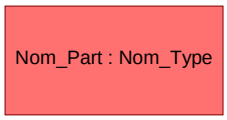
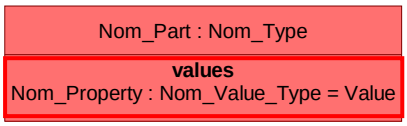
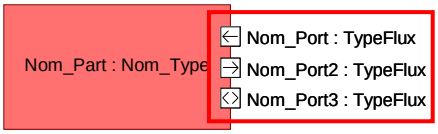


Figure II.14 :Internal Block Diagram : Bicyclette

Le Tableau II.5 résume les représentations graphiques des entités SysML utiles à la modélisation d'un IBD.

Tableau II.5 : Représentation SysML des concepts de la description organique (hiérarchie et interactions)

Concept	**Représentation SysML**		
	Diagramme	**Élément**	**Entité graphique SysML**
Composant	Internal Block Diagram	*Part*	
Propriétés		*Value Property*	
Port		*Flow Port*	

2.2. Description détaillée des composants

La description détaillée du comportement d'un composant, aussi bien, interne qu'en échange avec ses composants voisins s'effectue à l'aide de 3 diagrammes SysML : le diagramme d'état (*StateMachine Diagram*), le diagramme de séquence (*Sequence Diagram*) et le diagramme paramétrique (*Parametric Diagram*). Les *StateMachine Diagrams* (STM) permettent de modéliser les états internes des composants, de définir les conditions de passages d'un état à un autre ainsi que les opérations réalisées dans chaque état. Les *Sequence Diagram* (SD) permettent de modéliser les échanges entre composants (synchrone asynchrone,…) et les enchainements d'opérations qui sont liées à ces échanges. Enfin, le *Parametric Diagram* (ParD) est utilisé pour modéliser les *constraints properties* qui sont utilisées dans de diverses occasions afin de transcrire les relations mathématiques qui lient différentes propriétés.

L'utilisation des SD et des STM est très bien documentée dans la littérature puisque ce sont des diagrammes existants depuis UML 1.0 et abondamment utilisés aussi bien pour la conception logicielle que pour la conception des systèmes en général avec SysML. La norme SysML [OMG 2012] et certains ouvrages de référence traitant de la modélisation SysML tels que [Friedenthal et al. 2008],[Hause 2006] et [Roques 2009b] ou des ouvrages traitants de la modélisation UML tel que [Roques 2009a] permettent d'appréhender les mécanismes de la modélisation organique à l'aide de ces trois diagrammes. Nous allons cependant détailler l'utilisation des ParD qui nous seront particulièrement utiles dans la suite de la thèse.

Nous avons détaillé dans ce chapitre deux utilisations possibles des *constraints properties* :

- Spécifier les contraintes (environnementales, normatives,…) qui impactent sur les propriétés des composants,

- Détailler le fonctionnement d'une opération et comment elle impacte sur les propriétés des composants.

Les *constraints properties* ont donc pour objectif principal de décrire des relations en *value properties*. La Figure II.15 présente un exemple de *Parametric Diagram* (ParD) modélisant le calcul de la vitesse de notre bicyclette d'exemple en fonction des paramètres de ses roues.

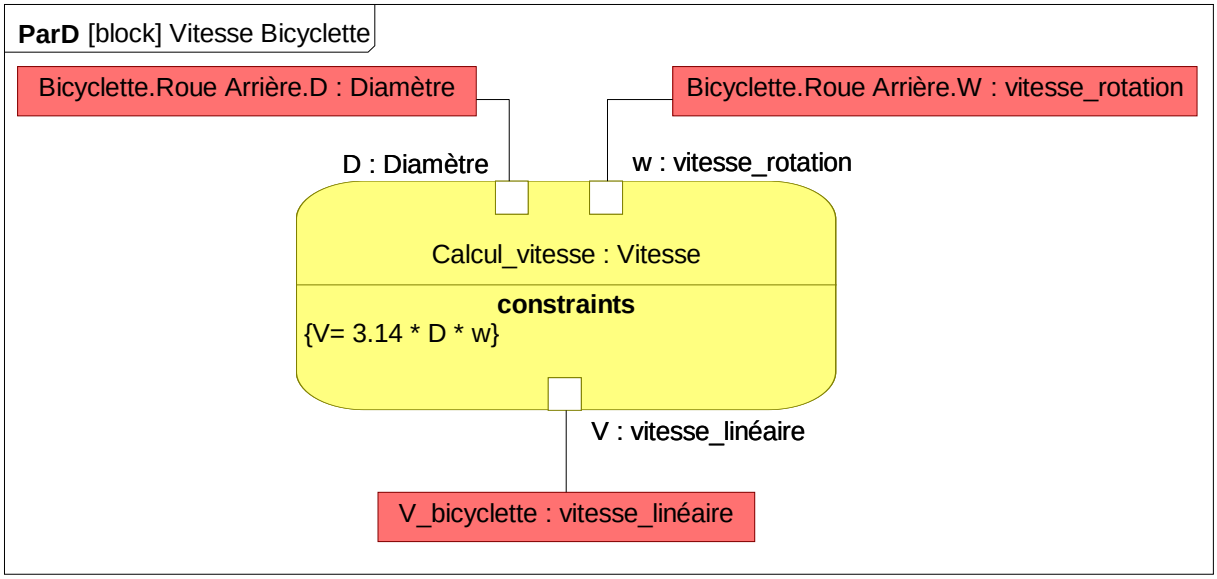


Figure II.15 : Calcul de vitesse de la bicyclette

La *constraint property*, en jaune sur la Figure II.15, dispose d'un nom, d'un type, de ports (*constraint parameters*), et de l'intitulé de la contrainte. Comme pour les *blocks* et les *parts*, il existe des types et des instances pour les contraintes. Un type de contrainte est exprimé avec un *constraint block* et l'instance de ce type dans un système particulier est exprimée avec une *constraint property*. Les *constraint parameters* sont les « ports » de la *constraint properties*, ils représentent l'interface de la contrainte, c'est-à-dire les paramètres impliqués dans la contrainte. La contrainte elle-même est définie dans la partie basse du rectangle jaune entre accolades. Ici, elle est exprimée sous forme mathématique, cependant la norme SysML n'impose pas de forme particulière. Des langages tels que le M-Code [MathWorks], le C ou le Python sont aussi régulièrement utilisés. Les *constraint parameters* sont ensuite reliés, soit à d'autres *constraints parameters* permettant ainsi de représenter les interconnections des contraintes entre elles, soit à des *value properties*. Dans l'exemple de la Figure II.15, on distingue trois *value properties* connectées à la *constraint property* « Calcul_vitesse ». D et w sont des *value properties* de « Roue Arrière » et V est une *value properties* de « Bicyclette ». La parenté de ces *value properties* est exprimée par le texte de chaque rectangle rouge : « Bicyclette.Roue Arrière.D : Diamètre ». « Bicyclette.Roue Arrière.D » exprime la parenté de la propriété, à savoir ici : le paramètre D de la Roue Arrière de Bicyclette. « : Diamètre » indique le type du paramètre en question, ici un « Diamètre ».

Cette façon de représenter les *constraint properties* dans un ParD reste vraie que ce soit pour exprimer une simple contrainte ou pour détailler une opération. Mais dans le cas où une *constraint property* définie spécifiquement une *opération*, alors il faudra l'indiquer par une relation d'*allocation* de la *constraint property* à l'*opération* en question. Dans ce cas, il faudra s'assurer que les propriétés impactées par la *constraint property* correspondent effectivement aux interfaces définies par l'*opération*.

La syntaxe de description de contraintes proposées par SysML est très efficace et fait l'objet de nombreux travaux et innovations. Ainsi Artisan Studio [ATEGO], propose par exemple de spécifier les contraintes en utilisant des modèles Simulink [MathWorks] exécutables.

Les règles de modélisation d'un système en SysML s'achèvent avec cette étape. Rappelons que le processus d'ingénierie système est un processus itératif (cf. Chapitre I §I).

VI. Conclusion

Dans ce chapitre, la description des activités d'ingénierie système et des règles permettant la réification de leur résultat en SysML a été réalisée. La sémantique SysML a été détaillée pour chaque activité d'IS considérée : Élicitation des besoins, Définition des exigences, Analyse fonctionnelle et Description organique. Les résultats de ces activités sont réifiés avec une vaste panoplie de symbole SysML et la cohérence des activités d'IS est elle aussi réifiée grâce à un ensemble de relations entre symboles bien défini. Ainsi, nous disposons donc d'un couple syntaxe / sémantique permettant de finaliser la description du langage système que nous avions évoqué dans le chapitre I et qui repose sur SysML. À présent, ce socle de modélisation système va permettre de créer des passerelles vers des DSLs et vers leurs outils qui permettent une étude approfondie du système.

Lors de la description de la problématique de l'intégration des COTS à un système complexe critique, deux méthodes d'évaluation de la SdF ont été retenues : l'AMDEC fonctionnelle et la méthode de prédiction de fiabilité FIDES. En nous appuyant sur un modèle système établi dans le respect de la syntaxe SysML et de la sémantique présentée dans ce chapitre, nous nous efforcerons de mettre au point des passerelles vers ces méthodes de SdF. Ces passerelles seront définies pour intégrer le cadre méthodologique MéDISIS et ainsi participer à la valorisation de l'approche ISBM pour les études de SdF.

À travers l'extraction et le traitement des données systèmes, les processus présentés aux chapitres III et IV vont permettre des études plus rapides, efficaces et cela au plus près des évolutions du système. Bien que ces processus soient motivés par l'étude de système intégrant des COTS, ils restent utiles pour une majorité de projets de conception. Ces processus s'inscrivent donc dans l'objectif final de MéDISIS : la validation plus efficiente d'un système et le rapprochement des domaines de la SdF et de l'IS.

Chapitre III. Génération de pré-AMDEC fonctionnelles

I. Introduction

Le chapitre I a mis en avant le besoin de mener les études de SdF à partir de la description fonctionnelle d'un système, particulièrement lors de l'étude d'un système intégrant des COTS. De même, nous avons vu que le cadre méthodologique de MéDISIS permet une valorisation du temps investi en ingénierie système en permettant l'extraction et la mise en forme des données du modèle système pour les études de SdF. Le principe de rédaction automatique de documents et modèles facilitant le travail de l'expert SdF est un avantage indéniable pour l'intégration des études de SdF au plus près des activités d'IS.

C'est pourquoi, en conservant cet objectif, nous allons étudier dans cette partie la possibilité de rédiger des pré-AMDEC fonctionnelles à partir des diagrammes SysML réifiant la description fonctionnelle du système. Par l'étude des méta-modèles des concepts de l'IS et de l'AMDEC fonctionnelle, nous proposons un outil de rédaction de pré-AMDEC permettant d'optimiser le travail de l'expert de SdF. Les algorithmes de traitement des données extraites du modèle système pour la rédaction de la pré-AMDEC seront présentés ainsi que l'utilisation que l'expert peut faire des pré-AMDEC fonctionnelles.

Cet outil pourra être utilisé très tôt dans le processus d'IS, dès l'analyse fonctionnelle, mais aussi jusqu'aux dernières phases de conception. L'outil devra donc être en mesure d'être exécuté de façon itérative et cohérente malgré les différentes évolutions du modèle système. Afin de maintenir les données dysfonctionnelles apportées par l'expert à chaque AMDEC successive, la mise en place d'une base de données permettant la sauvegarde des données dysfonctionnelles et de leurs relations avec le modèle système est nécessaire. Cette Base de données des Dysfonctionnements des Fonctions (BDF) sera structurée en s'inspirant de la BCD de MéDISIS en veillant à maximiser la restitution de l'intégralité de données d'expert d'une itération à l'autre.

De façon similaire aux avantages présentés par MéDISIS dans [David 2009] pour la génération d'AMDEC composants, l'utilisation de cet outil permet une meilleure identification des risques tôt dans le projet, réduisant ainsi les risques de modifications tardives alors très coûteuses. Dans le cas où des COTS sont utilisés, l'outil s'avère essentiel puisqu'il permet d'initier rapidement des études adaptées. Enfin, la capacité de modélisation de SysML permet en outre d'accéder à de nouvelles dimensions d'analyse permettant par exemple la prise en compte des exigences lors de l'AMDEC fonctionnelle.

Dans la deuxième partie de ce chapitre, nous détaillons l'implémentation de cet outil jusqu'alors méthodologique en un outil logiciel. Pour cela, des choix techniques quant à l'implémentation de l'outil ont dû être fait, par exemple, le choix du format informatique du modèle système pour son analyse par l'outil. Nous détaillerons la solution retenue utilisant le format d'échange standardisé de modèle UML et SysML : XMI, censé être commun à l'ensemble des logiciels d'édition de modèle SysML. Nous illustrerons succinctement le résultat final de l'outil lors de la rédaction de la pré-AMDEC de notre exemple d'étude à partir de fichier XMI issu d'un logiciel de modélisation standard.

Enfin sera présenté un retour d'expérience de notre démarche et de l'emploi de ce processus de rédaction de pré-AMDEC fonctionnelle dans le cadre de l'emploi d'une autre sémantique SysML.

II. Exploitation de l'ISBM supporté par SysML pour l'AMDEC fonctionnelle

Le processus de rédaction de l'AMDEC fonctionnelle (cf. Chapitre I) fait appel aux connaissances créées pendant les activités d'IS et qui sont réifiées, dans le cadre d'une approche ISBM, sous forme d'un modèle SysML. La méthodologie MéDISIS a déjà montré comment aller plus loin dans l'utilisation de SysML en fournissant un outil de génération de pré-AMDEC composant. Cet outil permet à l'expert SdF de passer un minimum de temps à l'analyse du modèle SysML pour se concentrer sur son cœur de métier : l'évaluation des modes de défaillances et la cotation du risque. Le terme pré-AMDEC introduit par [David 2009] désigne un tableau d'AMDEC pré-rempli avec l'ensemble des données extraites du modèle SysML et mise en forme pour coller à la syntaxe de l'AMDEC. La pré-AMDEC composant de [David 2009] contient l'ensemble des composants du système et pour chacun d'entre eux l'ensemble des causes et effets possibles déduit du modèle fonctionnel en SysML. En nous inspirant de ce processus MéDISIS, nous allons décrire le processus de rédaction de pré-AMDEC fonctionnelles permettant à l'expert de se concentrer sur son cœur de métier.

1. Les entités SysML utiles à l'AMDEC

Afin d'analyser les similitudes entre les différents concepts manipulés lors de l'AMDEC fonctionnelle et lors de la modélisation système, la sémantique et la syntaxe de chacun doivent être étudiées. Le chapitre II présente un ensemble des règles de modélisation SysML et les syntaxes du langage utiles pour la description du système. Nous décrivons donc ici, les relations et les propriétés des entités décrites au cours de l'AMDEC fonctionnelle à travers un méta-modèle (Figure III.1) permettant de formaliser la sémantique de l'AMDEC fonctionnelle. Ce méta-modèle s'appuie sur celui présenté par [David 2009] et sur la norme MIL-STD1629A [MIL-STD1629A].

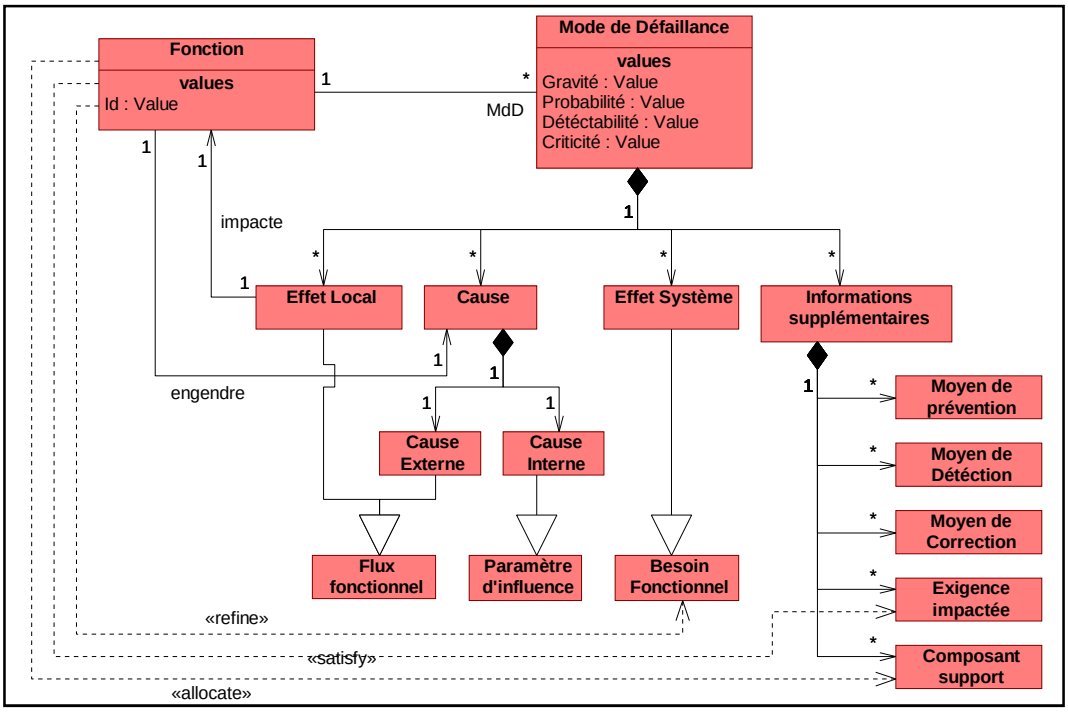


Figure III.1 : Méta-Modèle de l'AMDEC fonctionnelle

Le méta-modèle met en évidence les deux principaux éléments fondateurs de l'AMDEC : la fonction et le mode de défaillance. La fonction possède un identifiant utilisé afin de permettre une lecture rapide d'un document d'AMDEC. Le mode de défaillance comporte avant toute autre chose les propriétés valuées : Gravité, Probabilité, Détectabilité et Criticité. Un mode de défaillance est aussi constitué d'autres éléments :

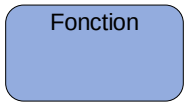
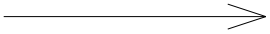
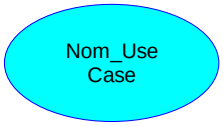
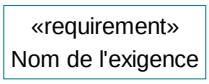
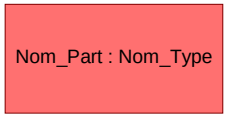
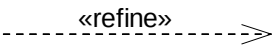
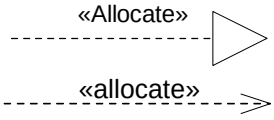
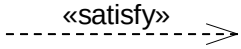
- Un effet local est un flux fonctionnel résultant du mode de défaillance impactant une fonction voisine.
- Une cause est un flux fonctionnel à l'origine du mode de défaillance, engendré par une fonction voisine (dans le cas d'une cause interne, la fonction qui engendre ce flux est la même fonction que celle dont on étudie le MdD).
- Un effet système est un besoin fonctionnel impacté par le MdD. Ce besoin est détaillé sous forme d'ensembles de fonctions lors des activités d'IS.
- Des informations supplémentaires de diverses natures :
    - Les moyens de réduction du risque qui sont propres au MdD étudié.
    - Les exigences impactées par le MdD, c'est-à-dire les exigences qui sont liées par un lien de satisfaction à la fonction dont on étudie le MdD.
    - Les composants supports qui sont les composants à qui sont allouées les fonctions dont on étudie les MdD.

Les activités d'IS (cf. Chapitre II) conduisent à définir certains de ces éléments :

- Fonction,
- Flux fonctionnel,
- Besoin fonctionnel,
- Composant,
- Exigence.

Cependant, le méta-modèle de l'AMDEC présente aussi les relations particulières nécessaires à sa rédaction. Il s'agit des relations d'allocation fonctionnelle, les relations de satisfaction d'exigence et les relations de composition fonctionnelle. Le Tableau III.1 présente l'ensemble des entités utiles à l'AMDEC et leur représentation en SysML. On y retrouve les entités issues de l'analyse fonctionnelle ainsi que les entités reliées aux fonctions par une des relations citées précédemment.

Tableau III.1 : Entités SysML utiles à l'AMDEC fonctionnelle

Élément issu de l'IS	**Représentation SysML**			**Élément de l'AMDEC**
	Entité SysML	**Entité graphique SysML**	**Diagramme**	
Fonction	*Opaque Action*		**Activity Diagram**	Fonction
Interfaces : Entrées et Sorties	*Input Pin* *Output Pin*		**Activity Diagram**	Causes externes / Effets locaux
Nom du flux fonctionnel	*Pin Name*		**Activity Diagram**	Causes externes / Effets locaux
Flux fonctionnel	*Object Flow*		**Activity Diagram**	Causes externes / Effets locaux
Besoins fonctionnels	*Use Case*		**Use Case Diagram**	Effets Système
Exigence	*Requirement*		**Requirement Diagram**	Exigences Impactées
Composant	*Part*		**Internal Block Diagram**	Composants support
Hiérarchie fonctionnelle	*Refine*		**Multiple**	Causes externes / Effets locaux / Effet système
Allocation Fonctionnelle	*Allocate*		**Multiple**	Composants support
Satisfaction	*Satisfy*		**Multiple**	Exigences Impactées

Ce tableau permet de caractériser l'ensemble des données disponibles dans un modèle SysML utiles à l'AMDEC fonctionnelle. Il est maintenant possible de définir un processus automatisable d'extraction de données d'un modèle SysML permettant de présenter à l'expert SdF des données cohérentes qui lui seront plus faciles à étudier que le modèle SysML. Pour cela le processus de génération de pré-AMDEC fonctionnelle est défini en s'inspirant du processus de rédaction d'AMDEC fonctionnelle présentée dans le chapitre I. La pré-AMDEC sera présentée dans un format similaire à celui de l'AMDEC fonctionnelle (Tableau I.2).

Tableau III.2 Forme tabulaire générique pour la pré-AMDEC fonctionnelle

Id	Fonction	MdD	Causes	Effets Locaux	Effets Système	Gravité	Fréquence	Détectabilité	Criticité	Moyens de Prévention	Moyens de Protection	Moyens de Détection	Composants support	Exigences impactées

2. Le processus de génération de pré-AMDEC fonctionnelle

2.1. Lister les fonctions du système

Cette activité consiste à lister l'ensemble des fonctions résultant de l'analyse fonctionnelle du système. Les fonctions présentées dans une AMDEC sont les fonctions du niveau de détail le plus bas afin d'obtenir le même niveau de détail que celui de l'analyse fonctionnelle. Ainsi, pour effectuer la liste des fonctions à étudier dans l'AMDEC, il faut parcourir l'ensemble des branches de décomposition fonctionnelle.

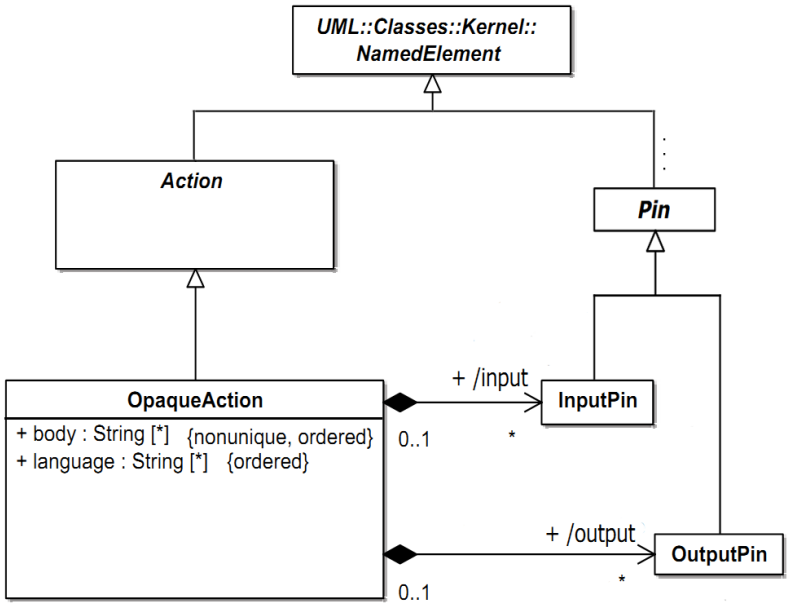


Figure III.2 : Méta-modèle de l'OpaqueAction

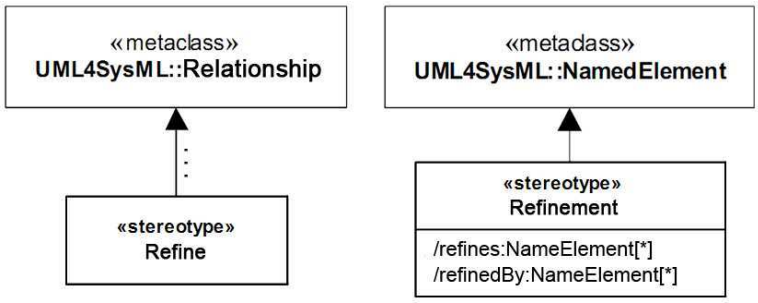


Figure III.3 : Méta-modèle de la relation Refine

Les fonctions sont modélisées en SysML par des *opaqueActions*. Elles sont organisées à un même niveau de détail dans des *Activity Diagram* et elles sont hiérarchisées en utilisant une logique de décomposition réifiée par les relations *refine* (cf Chapitre II). En étudiant le méta-modèle de l'*opaqueAction* présenté Figure III.2, et de la relation *refine* présentée Figure III.3, on observe que les *opaqueActions* héritent de l'entité UML *NamedElement*, qui désigne un élément de modélisation UML (ou SysML) qui possède au moins un nom, et que la relation *refine* hérite (à travers différents héritages non détaillés) de l'entité UML *Relationship* qui est l'entité de référence des relations en UML (et SysML). L'entité *NamedElement* est générique et peut représenter une grande variété d'entités de modélisation (notamment : *block*, *opaqueAction*, *part*, *use case*, …). Ainsi, la relation *refine* est utilisable pour une grande variété d'entités puisqu'elle est spécifiée pour relier deux *NameElement* (/refines :NameElement[*] et /refinedBy : NameElement, [*] signifie que plusieurs entités peuvent être reliés par cette relation). Nous nous intéresserons uniquement aux relations de raffinement d'une *opaqueAction* par un *activity diagram*. Nous pouvons alors formaliser les données que nous extrayons du modèle SysML pour chaque fonction f et chaque *activity diagram* AD.

$$f = \begin{Bmatrix} nom \\ AD \end{Bmatrix},\ AD = \begin{Bmatrix} nom \\ niveau \end{Bmatrix},\ niveau \in \{1,...,n\}$$

Nous pouvons aussi définir l'opération qui permet d'obtenir l'ensemble des fonctions représentées dans un *activity diagram* :

listFcn(AD) = {f / f ∈ AD}

De plus, nous définissons aussi pour un couple (f, AD) lié par une relation *refine*, les opérations refines et refinedBy permettant de trouver chaque extrémité de la relation :

AD = refinedBy(f)
f = refines(AD)

Ainsi la recherche des fonctions utiles au renseignement de la colonne « Fonctions » de l'AMDEC fonctionnelle correspond à l'ensemble de fonctions {f / refinedBy(f) = Ø}, signifiant qu'elles ne sont pas raffinées à un niveau de détail plus grand. La colonne « Fonctions » pourra alors être peuplée par les chaines : f.nom.

Lors de cette étape, nous allons aussi attribuer à chaque fonction son identifiant comme cela est réalisé lors de la rédaction traditionnelle de l'AMDEC (Colonne « Id »). Pour déterminer cet identifiant, plusieurs possibilités se présentent :

- Un identifiant s'incrémentant à chaque nouvelle fonction identifiée.
- Un identifiant composé dont les différentes composantes s'incrémentent selon les niveaux hiérarchiques des fonctions identifiées.

La première solution a le mérite d'être simple et d'assurer l'objectif minimum : avoir un identifiant unique pour chaque fonction. La deuxième solution offre en plus une facilité de relecture par niveau hiérarchique tel que défini lors de l'analyse fonctionnelle. En définitive, nous optons pour la seconde solution en utilisant une syntaxe simple : **$Fx_1.x_2.$ … $.x_i$**, où x_i correspond au numéro de la fonction dans un *activity diagram* donné, cet *activity diagram* étant au niveau de détail i (Figure III.4). Notons que le niveau 0 correspond aux besoins fonctionnels représentés dans le *use case diagram*.

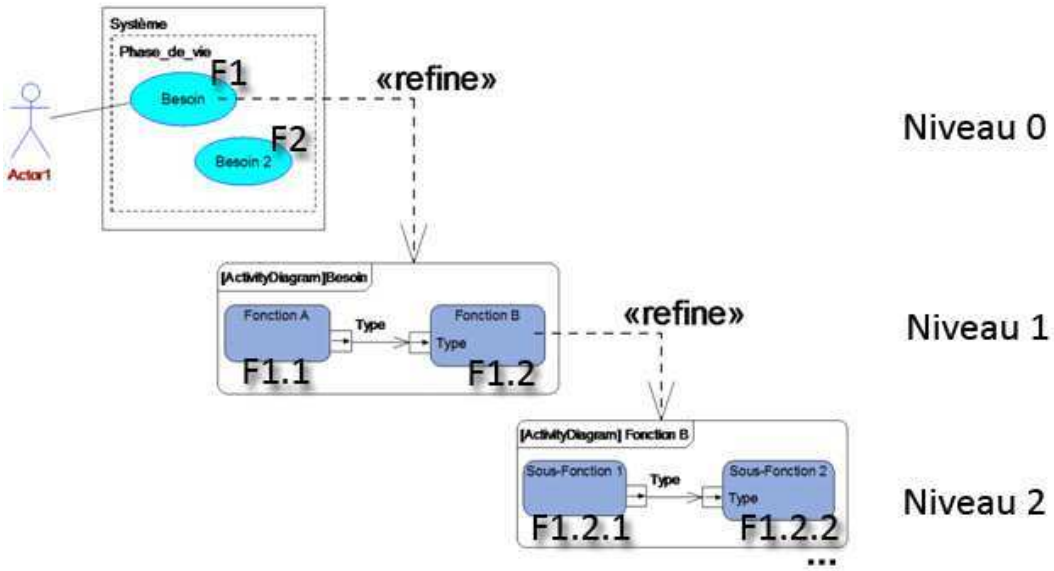


Figure III.4 : Attribution des id de fonction

À un niveau de détail donné, plusieurs *activity diagram* existent, cependant, par concaténation des x, deux fonctions ne peuvent pas avoir le même identifiant. Par la suite, on notera n le niveau de détail maximal du modèle.

L'expert dispose, dans la pré-AMDEC, d'une liste exhaustive des fonctions que l'AMDEC doit étudier, optimisant ainsi l'effort de travail de l'expert sur l'analyse des fonctions plutôt que sur leur identification dans le modèle système.

2.2. Établissement des modes de défaillances

Comme nous l'avons vu dans le chapitre I, lors de la rédaction d'AMDEC fonctionnelle, il existe un certain nombre de modes de défaillance génériques qui doivent être considérés pour tout type de fonction. Ces modes de défaillances étant en nombre fini [MIL-STD1629A], nous pouvons les lister dans la pré-AMDEC. Cette pratique amènera systématiquement l'expert à considérer ces MdD avant de choisir de les maintenir ou de les supprimer ce qui apporte à la pratique de l'expert un support méthodologique pour optimiser la qualité de l'analyse. Les modes de défaillances génériques proposés dans la pré-AMDEC sont les suivants :

- Perte de la fonction,

- Fonction exécutée de façon intempestive,
- Retard d'exécution de la fonction,
- Démarrage de la fonction impossible,
- Arrêt de la fonction impossible,
- Fonction intermittente,
- Fonction dégradée.

Les modes de défaillances spécifiques à une fonction ne sont par contre ni présents dans le modèle système, ni aisément déductibles des données système par une analyse algorithmique. Ainsi, ils ne pourront pas être renseignés dans la pré-AMDEC et seul l'expert pourra les détailler. Cependant, dans le chapitre I, nous avons vu que ces modes de défaillances spécifiques peuvent être liés au composant support de la fonction étudiée. Ce composant support est quant à lui présent dans le modèle système et peut être extrait pour faciliter le travail de l'expert. L'extraction des informations du composant support est abordée dans le §2.5 de ce chapitre.

2.3. Identifier l'environnement périphérique de chaque fonction

Cette activité a pour but d'examiner les entrées et sorties de chaque fonction et d'identifier les entités reliées à ces entrées et sorties. Cela permet donc d'identifier pour chaque fonction les flux fonctionnels entrant et sortant ainsi que les fonctions périphériques. Ainsi il sera possible d'établir l'ensemble des causes externes et effets locaux potentiels d'une défaillance de la fonction étudiée. Comme nous l'avons détaillé dans le chapitre II, lors de l'analyse fonctionnelle, il est naturel d'avoir recours à une présentation hiérarchisée des fonctions qui composent le système. Pour l'AMDEC, nous nous intéressons au niveau de détail le plus bas de chaque branche fonctionnelle. En conséquence, lors de la recherche des fonctions périphériques de la fonction étudiée, il est nécessaire de suivre les flux fonctionnels à travers les différents niveaux de décomposition.

En SysML, les *opaqueActions* possèdent des *inputs* et *output pins* pour modéliser les entrées et sorties d'une fonction. Sur la Figure III.2, le méta-modèle des *pins* est aussi représenté. Nous pouvons observer que les *InputPin* et les *OutputPin* appartiennent à une *opaqueAction*. Ils héritent tous deux de *pin*s et à travers d'autres héritages, ils héritent de *NamedElement*. Ces *pins* possèdent donc un nom qui permet de nommer le flux fonctionnel. Le *pin* est ensuite connecté à un autre *pin* par un *objectFlow*. L'autre *pin* du couple appartient à l'*opaqueAction* qui représente la fonction périphérique. La syntaxe SysML oblige la cohérence des *pins* connectés vis-à-vis de leurs directions respectives. Ainsi la direction de chaque *pin* (*InputPin* et *OutputPin*) permettra d'ajouter le sens du flux fonctionnel considéré. La Figure III.5 illustre ces principes avec un cas d'étude générique.

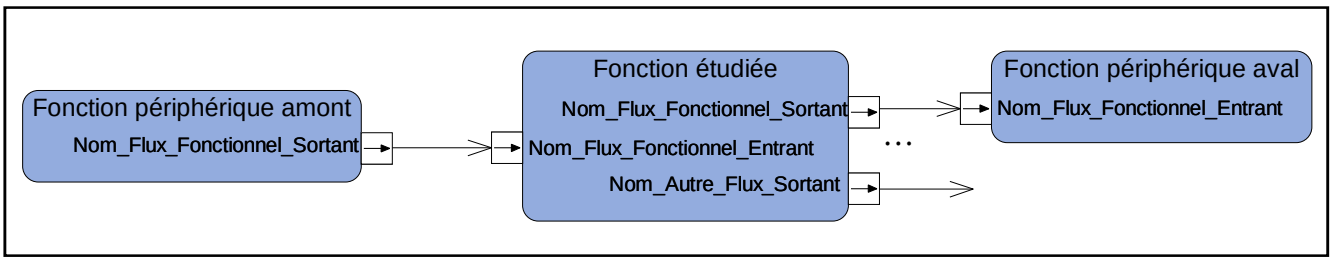


Figure III.5 : Détermination de l'environnement périphérique d'une opaqueAction

Il existe quelques cas particuliers liés à des flux fonctionnels traversant différents niveaux de détails de la hiérarchie fonctionnelle.

Notamment, le flux entrant peut provenir d'un niveau hiérarchique supérieur ou bien un flux sortant se diriger vers un niveau hiérarchique supérieur comme illustré dans la Figure III.6. Au niveau de la syntaxe SysML, cela se traduit par la présence de *pin* (in ou out) sur la bordure de l'AD.

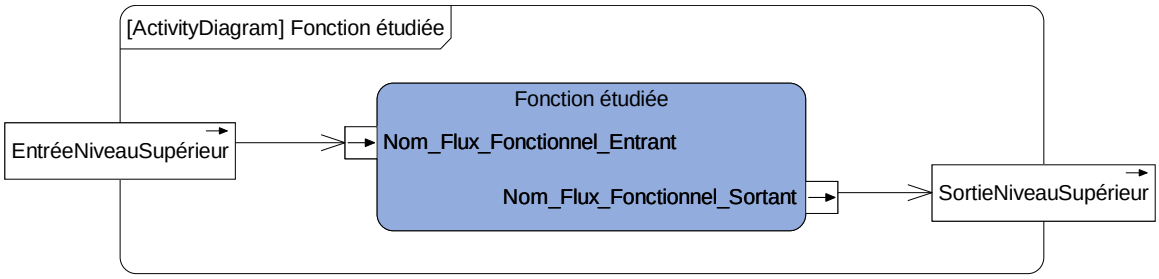


Figure III.6 : Entrée et Sortie d'un niveau hiérarchique supérieur

Dans ces deux cas particuliers, la recherche des fonctions périphériques est légèrement plus compliquée. Dans un premier temps, il est nécessaire de trouver la fonction de niveau supérieur. En effet, nous avons décrit dans le chapitre II (Figure II.8) comment les relations *refine* sont utilisées pour décrire la hiérarchie fonctionnelle. Les *pins* de bordure de l'AD correspondent aux *pins* de la fonction supérieure si les règles de modélisation du chapitre II sont respectées. Quand la fonction supérieure est identifiée, la recherche de fonction périphérique est exécutée comme dans le cas normal en débutant du *pin* portant le nom identique à celui du niveau inférieur.

À l'inverse, la recherche de fonction périphérique peut mener au *pin* d'une fonction qui est elle-même raffinée par un AD. Cela peut se produire en recherche aval ou amont de façon identique. En accord avec les règles de modélisation, un *pin* de bordure au niveau inférieur doit correspondre au *pin* identifié au niveau supérieur. Ainsi la recherche de fonction peut alors se dérouler normalement à partir de ce *pin* de bordure.

Enfin, il est possible que les deux cas de figure précédent (flux périphérique vers le niveau supérieur et fonction périphérique possédant une décomposition fonctionnelle) s'appliquent sur un même flux fonctionnel voire qu'ils s'imbriquent pour monter ou descendre de plusieurs niveaux à la suite.

Pour clarifier ces mécanismes de recherche, nous considérons donc qu'un flux fonctionnel est identifié par 2 *pins* P. Un *pin* appartient à une fonction ou à un AD. Dans les deux cas, un *pin* possède un nom et une direction (in ou out).

$$flux = \begin{Bmatrix} P_1 \\ P_2 \end{Bmatrix}, \quad P = \begin{Bmatrix} nom \\ direction \\ possesseur \end{Bmatrix}$$

Le possesseur est la fonction qui possède le *pin* P. Dans le cas particulier des *pins* de bordure d'AD, le possesseur est un AD. Nous définissons donc aussi les opérations :

listPin(f) = {P / P ∈ f}
listPin(AD) = {P / P ∈ AD}

La recherche de fonction périphérique fp consiste donc à suivre les flux fonctionnels de la fonction étudiée fe jusqu'à trouver la fonction fp où refinedBy(fp)=Ø, signifiant une fonction de bout de branche fonctionnelle (Algorithme III.1).

L'ensemble des fonctions fp : {fp}, que l'algorithme permet d'identifier représente l'ensemble des fonctions périphériques. Pour classer les fonctions périphériques entre causes et effets potentiels, il faut s'attacher à la direction du *pin* Pi de la fonction étudiée.

- Cause externe : Pi.direction = in. La cause externe sera alors rédigée : [Pi.nom]fp.nom : Ø. En effet la description détaillée de la cause reste vide et sera renseignée ensuite par l'expert.
- Effet local : Pi.direction = out. L'effet local sera alors rédigé : [Pi.nom]fp.nom : Ø. De même, la description détaillée reste vide et sera renseignée ensuite par l'expert.

En effet, les flux entrants associés aux fonctions périphériques amont représentent l'ensemble de toutes les causes externes possibles d'une défaillance de la fonction étudiée. De même, les flux sortants associés aux fonctions périphériques aval représentent l'ensemble de tous les effets locaux possibles d'une défaillance de la fonction étudiée. Finalement, il est donc possible de remplir les colonnes « Cause » et « Effet Local » de la pré-AMDEC fonctionnelle.

Algorithme III.1 Recherche des fonctions périphériques

```
Parcourir ( {Pi} ← listPin(fe) )
        Parcourir {flux}
                Si flux.P1=Pi
                        Alors Pp ← flux.P2
                Si flux.P2=Pi
                        Alors Pp ← flux.P1
        Tant que fp= Ø
                Si (Pp. possesseur = type(f)) et (refinedBy(Pp.possesseur) = Ø)
                        Alors fp ← Pp.possesseur
                Si (Pp. possesseur = type(f)) et (refinedBy(Pp. possesseur) ≠ Ø)
                        Alors Parcourir ( {Pj} ← listPin(refinedBy(Pp.possesseur)) )
                                Parcourir {flux}
                                        Si flux.P1=Pj
                                                Alors Pp ← flux.P2
                                        Si flux.P2=Pj
                                                Alors Pp ← flux.P1
                Si Pp. possesseur = type(AD)
                        Alors Parcourir ( {Pk} ← listPin(refines(Pp.possesseur)) )
                        Si Pk.nom=Pp.nom
                                Alors Pp ← Pk
        Ajouter fp à {fp}
        fp ← Ø
```

2.4. Rechercher le besoin fonctionnel dont dépend la fonction étudiée

Dans la partie précédente, nous rappelions le mécanisme de raffinement modélisant la hiérarchie fonctionnelle. Au chapitre II, nous avions expliqué comment les besoins (modélisés sous forme de *use case*) se trouvent au sommet de la hiérarchie fonctionnelle. Ainsi, à partir de la fonction étudiée, il suffit de suivre les relations *refines* des différents AD pour trouver le *use case* représentant le besoin de haut niveau duquel dépend la fonction étudiée. C'est ce besoin qui risque d'être touché si la fonction défaille, donc la non-réalisation de ce besoin est un effet système potentiel.

En SysML, un *use case* (uc) est identifié par son nom et le *Use Case Diagram* (UCD) dans lequel il se trouve. L'UCD quant à lui est identifié par son nom.

$$uc = \begin{Bmatrix} nom \\ UCD \end{Bmatrix}$$

Les *use cases* sont liés par des liens *refine* avec des AD comme pour les fonctions. Cependant, les *use case*s se trouvent à l'origine des branches de l'architecture fonctionnelle. Par conséquent, les AD qui raffinent un uc sont de niveau 1. Le niveau 0 est réservé aux UCD. On peut donc ajouter l'opération :

uc = refines(AD), ssi AD.niveau = 1.

Ainsi la recherche du besoin fonctionnel : ucbf, présent à la racine de la branche fonctionnelle de la fonction étudiée : fe, consiste à suivre de façon itérative l'ensemble des liens *refine* de la branche fonctionnelle comme le suggère l'Algorithme III.2.

Algorithme III.2 : Recherche du besoin fonctionnel

```
f ← fe
Tant que ucbf = Ø
        Si type(refines(f.AD)) = type(uc)
                Alors ucbf = refines(f.AD)
        Sinon f ← refines(f.AD)
```

Les liens de décomposition fonctionnelle étant uniques et non recouvrables (un AD ne peut pas raffiner la description de deux fonctions), pour une fonction étudiée fe, il n'y a donc qu'un seul *use case* ucbf au sommet de la branche fonctionnelle. Ainsi dans la pré-AMDEC, nous pouvons renseigner la colonne « Effet Système » avec la chaine : [Besoin] ucbf.nom. Le préfixe : [Besoin] est nécessaire pour différencier cet effet système des éventuels effets système spécifique qui seraient ajoutés par la suite par l'expert.

En plus de ces informations, nous pouvons aussi envisager d'ajouter une autre donnée utile à l'expert : la phase de vie à laquelle ce besoin est alloué. En effet, lors de l'élicitation des besoins, un diagramme de contexte est réalisé en association avec l'UCD pour permettre la réification des phases de vie du système. La relation d'*Allocation* existant entre la phase de vie et le besoin permet d'accéder à l'entité qui décrit la phase de vie et donc à son nom (cf §2.5). La colonne « Effet

système » de la pré-AMDEC fonctionnelle peut alors être renseignée avec la chaine précédente détaillée ainsi : [Besoin] ucbf.nom *(Phase : phase.nom)*. Notons que contrairement à la colonne Effets Locaux qui contient l'ensemble des effets possibles, cette colonne peut contenir des informations spécifiques, autres que la liste des besoins menacés, qui pourront être apportées par l'expert.

2.5. Rechercher les composants support et les exigences impactées

Ces informations représentent tout l'intérêt d'utiliser l'approche ISBM supporté par SysML. La syntaxe et la sémantique de SysML présentée dans le chapitre II permettent à travers l'utilisation de relations SysML, de réifier les interconnexions des différents concepts entre eux. Ainsi l'identification de ces informations (composants supports et exigences impactées) est possible par simple lecture du modèle système. Pour cela, nous étudions le méta-modèle de SysML centré sur les entités et relations qui nous intéressent.

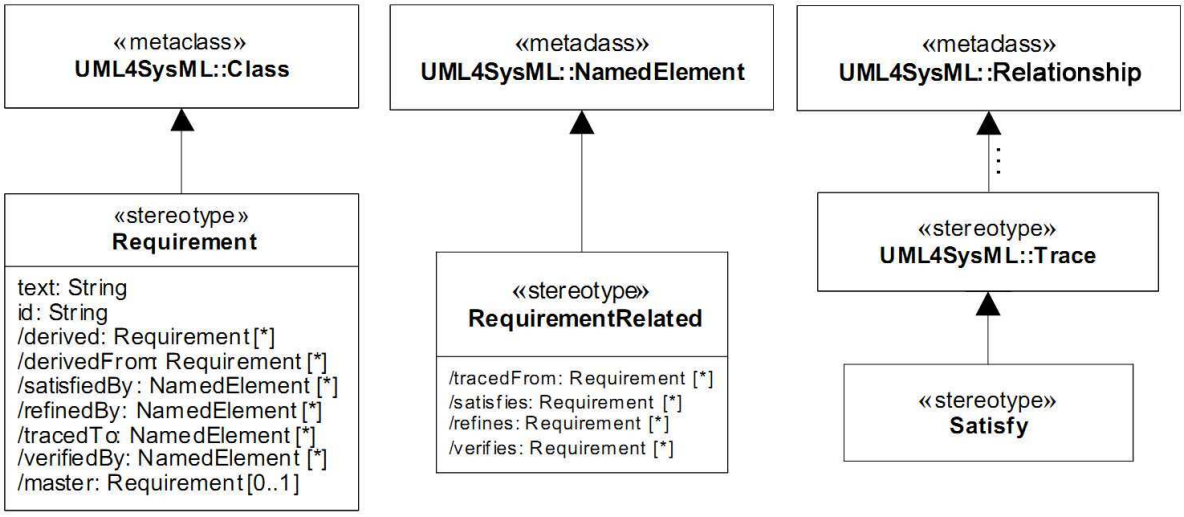


Figure III.7 : Méta-modèle de Requirement SysML

Pour retrouver les exigences potentiellement impactées par une défaillance de la fonction étudiée, il faut regarder s'il existe des relations *satisfies* allant de la fonction étudiée vers une exigence. Cette exigence est modélisée en SysML par un *requirement* dont le méta-modèle est présenté dans la Figure III.7. Ce *requirement* se caractérise par un texte de description (text :String), un identifiant (id :String) et un ensemble de relations possibles avec d'autres éléments du modèle. Nous voyons que les relations : *derived* et *derivedFrom* sont possibles avec d'autres *requirements*. La relation *master* présentée dans le méta-modèle indique la relation de décomposition des exigences présentée dans le chapitre II. Pour la génération d'AMDEC, nous étudierons plus particulièrement la relation *satisfy* qui relie un *requirement* SysML à un *NamedElement*. Notons que s*atisfy* hérite de la relation UML *trace* qui elle-même à travers différents héritages trouve son origine auprès de l'entité UML *Relationship*. Selon la sémantique présentée au chapitre II, cette relation *satisfy* entre une exigence et une fonction réifie effectivement les exigences satisfaites par une fonction qui seront donc impactées en cas de défaillance. Nous définissons alors, le *requirement* req et les opérations satisfiedBy() et satisfy() :

$$req = \begin{Bmatrix} nom \\ id \\ text \end{Bmatrix},$$

f = satisfiedBy(req),

req = satisfy(f).

Les résultats de ces opérations peuvent aussi bien correspondre à un élément unique ou bien à un ensemble d'éléments. Finalement, l'ensemble {reqi} des exigences impactées par un mode de défaillance de la fonction étudiée fe, correspond à l'ensemble des exigences vérifiant {reqi}=satisfy(fe). Ainsi la colonne « Exigences impactées » de la pré-AMDEC pourra être renseignée par l'ensemble des phrases constituées selon le schéma suivant : [reqi.id] reqi.nom *(reqi.texte)*. Le rappel du texte de l'exigence pourra être omis par soucis de présentation graphique dans le tableau.

En SysML, les composants sont réifiés par des *parts*. À ces *parts* sont allouées les fonctions qu'ils exécutent (cf Chapitre II). Ce sont donc les éventuelles relations *allocatedTo* qu'il nous faut étudier afin d'identifier le *part* support. La Figure III.8 présente le méta-modèle de la relation *allocate*. Elle permet de relier deux *NamedElement*. L'allocation est donc applicable à un grand nombre d'entités comme l'était la relation *refine*. Cependant, nous considérons donc en accord avec les règles de modélisation du chapitre II que les allocations entre *opaqueAction* et *part* correspondent toujours à l'allocation fonctionnelle et nous nous limiterons à l'étude de ces allocations.

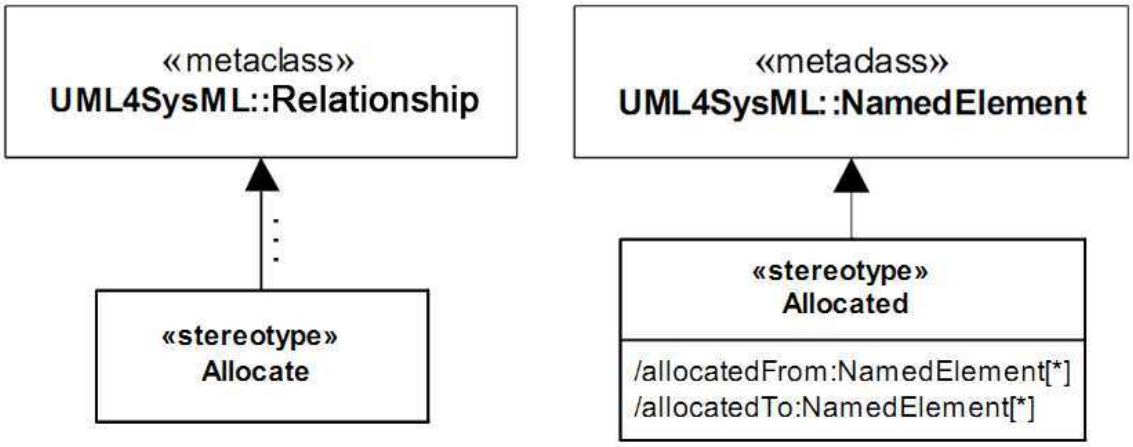


Figure III.8 : Méta-modèle d'Allocation SysML

Les *parts* représentant les composants sont typés par des *blocks*. Pour l'AMDEC fonctionnelle, les informations utiles sont : le nom du composant (le nom du *part*) et le nom du type de composant (le nom du *block*). Nous pouvons donc à présent définir la notion de composant et les opérations représentant la relation d'allocation :

$$c = \begin{Bmatrix} nom \\ type \end{Bmatrix},$$

c = allocatedTo(f)

f = allocatedFrom(c)

L'ensemble des composants supports de la fonction étudiée, fe sera donc l'ensemble :{cs} = allocatedTo(fe). Enfin, dans la colonne « Composants Support » de la pré-AMDEC sera indiquée l'ensemble des phrases constitué selon le schéma suivant : cs.nom *(Type : cs.type)*.

2.6. Modes de défaillances spécifiques, causes internes, critères de criticité, moyens de réduction du risque

Si nous reprenons le méta-modèle de l'AMDEC fonctionnelle présenté Figure III.1, nous pouvons lister l'ensemble des données qui manque à ce stade : les modes de défaillances spécifiques, les causes internes de défaillance, les 3 critères de criticité et la criticité et l'ensemble des propositions de réduction du risque (moyens de prévention, protection et détection). Ces données sont issues uniquement du retour d'expérience et des connaissances d'un expert de SdF et donc l'analyse du modèle système SysML ne permet en aucun cas d'extraire ces informations. Nous pouvons cependant noter que la possibilité d'extraire les composants supports de chaque fonction facilitera tout de même le travail de l'expert lors de la description des causes internes de défaillances puisque nous avions établi qu'elles sont en partie dues à des paramètres d'influence d'origine organique. Ce point nous amène donc à définir comment l'expert de SdF va pouvoir exploiter la pré-AMDEC ainsi rédigée pour produire une AMDEC fonctionnelle complète.

3. Utilisation de la pré-AMDEC par l'expert de SdF

À l'issue des étapes précédentes, l'expert dispose d'une pré-AMDEC qu'il doit modifier et compléter pour obtenir une AMDEC fonctionnelle convenable. Si l'on considère notre exemple de la bicyclette, l'élicitation des besoins résulte en un UCD (Figure II.2). On considère que le *use case* « Se déplacer » est lié à l'AD ci-dessous Figure III.9 par une relation *refine*.

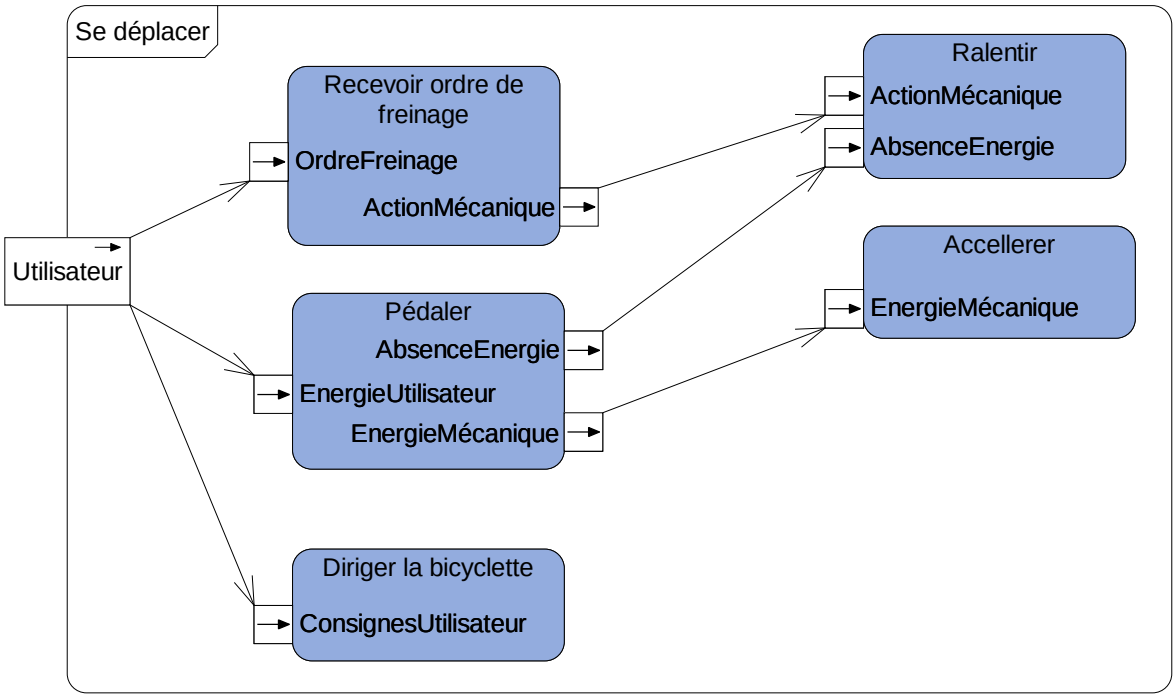


Figure III.9 : Diagramme d'activité appartenant à l'analyse fonctionnelle de la bicyclette

Les relations d'allocation fonctionnelle et de satisfaction d'exigences sont également réalisées (cf. Chapitre II). En appliquant les algorithmes d'extraction de données précédents, nous obtenons une pré-AMDEC dont le Tableau III.3 est un extrait. Nous pouvons voir l'analyse de la fonction : « Recevoir consigne de freinage ». Pour cette fonction, on dispose de 8 lignes MdD : les 7 MdD génériques proposés par la MIL-STD1629A et une ligne, où la case MdD est vierge, que l'expert pourra utiliser pour ajouter des MdD spécifiques. Ensuite, on retrouve dans la colonne « Causes » les causes externes déduites du modèle, ici il n'y a qu'une cause possible provenant du *pin* de bordure Utilisateur par le flux fonctionnel : « OrdreFreinage ». De même, dans la colonne « Effets Locaux », on retrouve l'ensemble des effets locaux possibles, ici il n'y en a qu'un seul : Effet sur la fonction « Ralentir » par le flux fonctionnel « ActionMécanique ». Sans détailler, nous pouvons observer que les colonnes « Effets Système », « Composants support » et « Exigences impactées » sont aussi renseignées en accord avec les algorithmes définis précédemment.

Tableau III.3 : Extrait de pré-AMDEC fonctionnelle

Id	Fonction	Modes de Défaillance	Causes	Effets Locaux	Effets Système	Gravité	Fréquence	Détectabilité	Criticité	Moyens de Prévention	Moyens de Protection	Moyens de Détection	Composants support	Exigences impactées
F1.1	Recevoir consigne de freinage	Perte de la fonction	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer *(phase : Utilisation)*								Manette droite de frein (*Type : mannette de frein)*	[Req2] Freinage *(Le système doit être capable de freiner sur action de l'utilisateur)*
		Fonction exécutée de façon intempestive	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer								Manette droite de frein	[Req2] Freinage
		Retard d'exécution de la fonction	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer								Manette droite de frein	[Req2] Freinage
		Démarrage de la fonction impossible	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer								Manette droite de frein	[Req2] Freinage
		Arrêt de la fonction impossible	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer								Manette droite de frein	[Req2] Freinage
		Fonction intermittente	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer								Manette droite de frein	[Req2] Freinage
		Fonction dégradée	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer								Manette droite de frein	[Req2] Freinage
			[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer								Manette droite de frein	[Req2] Freinage
F1.2	Ralentir	Perte de la fonction	[ActionMécanique] Recevoir ordre de freinage	Ø	[Besoin] Se déplacer *(phase : Utilisation)*								Patin de frein arrière *(Type : patin de frein)*	[Req2] Freinage *(Le système doit être capable de freiner sur action de l'utilisateur)*

NB : les informations en italique dans le tableau sont des informations facultatives qui ne sont pas répétées dans chaque ligne

Comme nous l'avons déjà évoqué à de multiples reprises, la pré-AMDEC (extrait Tableau III.3) est constituée à l'excès, c'est-à-dire avec toutes les informations qui pourraient *a priori* s'avérer utiles à l'expert. En effet, dans la pré-AMDEC, toutes les colonnes sont renseignées avec l'ensemble des informations potentiellement utiles à l'expert et extractibles du modèle système. Ainsi, à première vue, la pré-AMDEC rédigée est imposante inutilement. L'expert a cependant devant les yeux l'ensemble des données utiles à l'AMDEC fonctionnelle. À ce stade, l'expert va pouvoir exercer son métier de la manière qu'il l'entend. Cependant, en utilisant notre outil nous avons pu détecter une marche à suivre globale permettant d'adopter la pré-AMDEC de façon efficiente.

Dans un premier temps, il est utile de prendre connaissance de la fonction étudiée, des éventuelles exigences et composants supports associés afin d'avoir une bonne compréhension de la fonction. Notons cependant que tout au long de la rédaction de l'AMDEC fonctionnelle (finale), la présence du modèle SysML et des documents d'IS est nécessaire comme support à l'étude. De même, l'expert devra avoir pris conscience du système étudié préalablement comme c'est toujours le cas traditionnellement. L'outil de rédaction de pré-AMDEC optimise le travail de l'expert, mais ne se substitue pas à son jugement.

La liste des modes de défaillances génériques étant conséquente et parfois redondante appliquée à certaines fonctions, l'expert pourra d'abord déterminer les modes de défaillances non révélateurs pour la fonction étudiée. L'ajout des modes de défaillances spécifiques viendra ensuite. Pour chaque mode de défaillance générique maintenu, un tri similaire pourra être effectué au niveau des causes et effets locaux. Lors de ce tri, l'expert étant penché sur la problématique des causes de défaillance, il pourra naturellement ajouter les causes internes de défaillance. Pour cela nous proposons la syntaxe suivante : [Interne] Nom de la cause interne. Cette syntaxe bien que superflue à ce stade se révélera utile lors des itérations successives de l'outil de rédaction au cours du projet. De façon similaire, des effets systèmes peuvent être ajoutés. Ces effets systèmes qui ne correspondent pas à des besoins fonctionnels pourront être indiqués selon la syntaxe : [Autre] Nom de l'effet système.

Nous considérons que les colonnes Composants support et Exigences impactées n'ont pas de raisons d'être modifiées. Si cela est nécessaire, alors le modèle système est incomplet ou incohérent, et une demande de modification de celui-ci devra être émise.

Enfin, les colonnes de critères de risques, de criticités et de moyens de réduction des risques peuvent être remplies par l'expert comme dans toute AMDEC fonctionnelle traditionnelle. Au terme de ce travail, l'AMDEC fonctionnelle est alors terminée (extrait Tableau III.4).

Tableau III.4 : Extrait de l'AMDEC fonctionnelle complétée

Id	Fonction	Modes de Défaillance	Causes	Effets Locaux	Effets Système	Gravité	Fréquence	Détectabilité	Criticité	Moyens de Prévention	Moyens de Protection	Moyens de Détection	Composants support	Exigences impactées
F1.1	Recevoir consigne de freinage	Démarrage de la fonction impossible	[Interne] Manette bloquée ouverte	[ActionMécanique] Ralentir	[Besoin] Se déplacer *(phase : Utilisation)*	4	2	3	24	Maintenance Préventive		Maintenance Préventive	Manette droite de frein *(Type : mannette de frein)*	[Req2] Freinage *(Le système doit être capable de freiner sur action de l'utilisateur)*
		Arrêt de la fonction impossible	[Interne] Mannette bloquée fermée	[ActionMécanique] Ralentir	[Besoin] Se déplacer	2	2	3	12	Maintenance Préventive		Maintenance Préventive	Manette droite de frein	[Req2] Freinage
		Fonction dégradée	[Interne] Usure de la manette	[ActionMécanique] Ralentir	[Besoin] Se déplacer	1	3	3	9	Maintenance Préventive		Maintenance Préventive	Manette droite de frein	[Req2] Freinage
F1.2	Ralentir	Perte de la fonction	[Interne] Câble de frein sectionné	Ø	[Besoin] Se déplacer	4	2	3	24				Patin de frein arrière	[Req2] Freinage
		Démarrage de la fonction impossible	[ActionMécanique] Recevoir ordre de freinage [Interne] Patin bloqué ouvert	Ø	[Besoin] Se déplacer	4	2	3	24				Patin de frein arrière	[Req2] Freinage
		Arrêt de la fonction impossible	[ActionMécanique] Recevoir ordre de freinage [Interne] Patin bloqué fermé	Ø	[Besoin] Se déplacer	2	2	3	12				Patin de frein arrière	[Req2] Freinage
		Fonction dégradée	[ActionMécanique] Recevoir ordre de freinage [Interne] Usure du patin	Ø	[Besoin] Se déplacer	1	3	3	9				Patin de frein arrière	[Req2] Freinage

Le processus de rédaction de pré-AMDEC fonctionnelle présenté dans ce chapitre vise à s'intégrer au cadre méthodologique MéDISIS. Ce cadre méthodologique a pour objectif d'inclure les analyses de sûreté de fonctionnement au plus près du processus de spécification et conception du système. Ainsi, les processus de rédaction et de génération de modèle existant dans MéDISIS doivent être utilisés à chaque évolution du système et de sa modélisation, afin d'évaluer l'impact des nouvelles spécifications sur la sûreté de fonctionnement. Or, malgré les évolutions du système, les études précédentes ne sont pas totalement obsolètes, et l'expert ne repart pas de zéro à chaque évolution, puisqu'il possède l'expérience et les résultats des AMDEC des précédentes versions. Il est donc indispensable que les processus du cadre MéDISIS résolvent cette problématique. Les travaux de [David 2009] se sont confrontés au problème. Le moyen de pérennisation des informations dysfonctionnelles s'est concrétisé sous la forme de la Base de données des Comportements Dysfonctionnels (BCD). Les objectifs de la BCD alors définie sont doubles :

- À chaque nouvelle version des spécifications du système, l'expert n'a pas à tout redéfinir, mais uniquement à mettre à jour les défaillances des fonctions modifiées et à analyser les nouvelles fonctions introduites.
- Pour chaque nouveau projet, les données communes avec les projets précédents sont réutilisées afin de focaliser le travail de l'expert sur les parties innovantes et nouvelles.

La BCD proposée par [David 2009] remplit parfaitement ces objectifs. Cependant, elle est adaptée à un raisonnement centré sur les composants. Ainsi la structure de la BCD, que nous détaillerons au chapitre IV, est définie pour permettre le stockage des comportements dysfonctionnels utiles à la rédaction d'AMDEC composant et à la génération de modèle Altarica DF. Or pour la rédaction de pré-AMDEC fonctionnelle, nous manipulons principalement des fonctions.

De plus, dans le cas de la description d'architecture fonctionnelle, il s'agit bien souvent d'entités qui sont propres à un projet et à un système. En effet, il est rare de retrouver une fonction d'un projet à un autre. Même lorsque des fonctions de projets différents sont proches, nous ne retrouverons pas le même intitulé pour définir ces fonctions. Ainsi il n'est pas raisonnable d'envisager de pérenniser des données de SdF pour de multiples projets dans le cadre de la génération d'AMDEC fonctionnelle. Néanmoins, la nécessité de s'intégrer à un processus d'IS projet avec de multiples itérations successives reste fondamentale. C'est pourquoi nous allons définir notre Base de données des Dysfonctionnements des Fonctions (BDF) permettant au niveau projet de maintenir les informations apportées par l'expert SdF lors de la rédaction de l'AMDEC fonctionnelle.

4. Base de données des Dysfonctionnements des Fonctions : BDF

4.1. Les informations utiles

Le besoin principal de stockage des données dysfonctionnelles provient avant tout de l'optimisation du temps de l'expert SdF qui ne doit pas effectuer plusieurs fois le même travail. Ainsi, au minimum l'ensemble des données qui sont apportées par l'expert (qui correspondent donc à celles qui ne sont pas extraites du modèle SysML) doit être réifié dans la BDF :

- Modes de défaillances spécifiques.
- Gravité, Probabilité, Détectabilité, Criticité de tous les MdD.

- Causes internes de tous les MdD.
- Moyens de Prévention, Protection, Détection de tous les MdD.

Cependant l'expert de SdF n'apporte pas seulement des données, il effectue aussi un travail de tri et de filtrage des données extraites du modèle SysML. Aussi, il est nécessaire de pouvoir mémoriser quelles informations ont été filtrées pour ne pas les réafficher lors des générations suivantes. Enfin, un autre point est à prendre en compte, c'est la nécessité de pouvoir maintenir la cohérence entre les informations maintenues d'une génération à l'autre avec les informations du modèle SysML. Ainsi aux données précédentes il sera nécessaire d'ajouter :

- Nom de fonction
- Modes de défaillance
- Causes externes et Effet locaux
- Effets Système
- Relations de dépendances entre toutes les données (satisfaction, allocation, décomposition,…)
- Informations de filtrage ou non des données générées

Finalement, il s'agit de stocker la quasi-intégralité des données de la pré-AMDEC et de l'AMDEC finale. Nous pouvons donc associer à notre BDF un méta-modèle similaire à celui présenté Figure III.1, cependant la BDF est supportée par SysML pour optimiser la comparaison des données avec le modèle système, ainsi le méta-modèle de la BDF devient celui présenté Figure III.10.

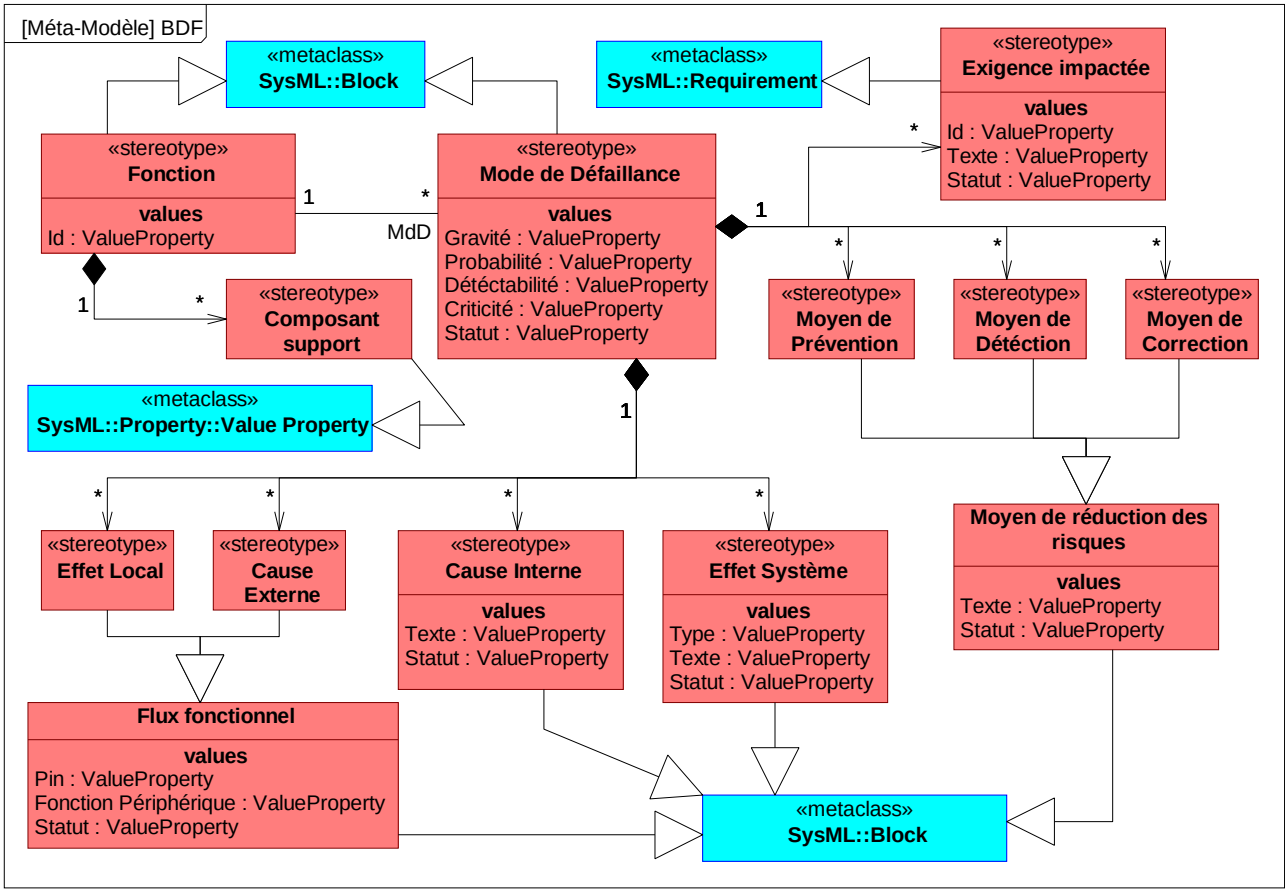


Figure III.10 : Méta-Modèle de la BDF

Nous pouvons voir que l'ensemble des données précédentes est présent dans le méta-modèle. Les informations concernant la fonction étudiée et devant être maintenue sont son nom et son Id. Ces informations sont donc réifiées dans la BDF à l'aide d'un *block* dont le nom est celui de l'*opaqueAction* (dans le modèle système) et possédant au moins une *value property* pour définir l'Id. Ce *block* possédera aussi une liste de *value properties* permettant de représenter le ou les composants support de la fonction. Le *block* représentant la fonction est associé à un nombre fini de modes de défaillance décrits eux aussi par un *block* dont le nom est le nom du mode de défaillance (« Pas de fonction »,…). Un mode de défaillance possédera au minimum trois *value properties* : « Probabilité », « Gravité » et « Criticité » et éventuellement une quatrième *value property* : « Détectabilité », si elle est renseignée. Les effets locaux et les causes externes sont des flux fonctionnels attribués au mode de défaillance. Un flux fonctionnel est un *block* contenant deux informations majeures : le *pin* de la fonction étudiée par lequel le flux passe et la fonction périphérique en relation avec la fonction étudiée par l'intermédiaire de ce flux. Les exigences impactées sont représentées en utilisant des *requirements* auxquels est ajoutée la *value property* « Statut » que nous détaillerons ensuite. Les effets système, sont représentés par un *block* contenant une *value property* « Type », pour décrire l'origine de l'effet système, à savoir :

- « use case » s'il s'agit d'un effet système déduit du modèle système par analyse de la hiérarchie fonctionnelle. Dans ce cas, la *value property* « Texte » peut être vide si l'expert SdF n'a pas détaillé cet effet.
- « spécifique » s'il s'agit d'un effet ajouté par l'expert. Dans ce cas la *value property* « Texte » sera utilisée pour décrire l'effet.

Les causes internes sont représentées par un *block* contenant une *value property* « Texte » permettant de décrire la cause. Enfin les moyens de prévention, détection et protection sont tous réifiés de façon similaire. Un moyen de réduction du risque quel qu'il soit est représenté sous forme d'un *block* disposant d'une *value property* « Texte » permettant la description fine de celui-ci. Enfin, les données susceptibles d'avoir été filtrées par l'expert possèdent une *value property* « Statut » prenant différentes valeurs :

- « maintenu », lorsque l'expert SdF a maintenu l'élément lors de la dernière phase de complétion de la pré-AMDEC,
- « supprimé », lorsque l'expert à filtré l'élément, considérant qu'il n'était pas utile à l'AMDEC.

Nous allons voir à présent comment la BDF va permettre l'amélioration de la génération de pré-AMDEC et optimiser le travail de l'expert.

4.2. Utilisation de la BDF

La BDF est utile pour compléter la pré-AMDEC fonctionnelle d'un système dont une version précédente avait déjà permis la génération d'AMDEC. En effet, la BDF est mise au point pour être initialisée lors de la première génération de pré-AMDEC après complétion par l'expert. Par la suite, à chaque génération successive, la BDF va permettre :

- De restituer le filtrage de l'expert pour les éléments de l'analyse fonctionnelle qui n'ont pas été modifiés.

- Mettre en avant les données qui ont été modifiées depuis la version précédente.

Pour atteindre ces objectifs, la BDF va dans un premier temps permettre d'identifier les variations de l'architecture fonctionnelle. Par comparaison des données de la pré-AMDEC nouvellement générée avec les données de la BDF, il possible d'identifier les nouvelles fonctions ainsi que les changements d'interface des fonctions existantes. En effet, la liste des fonctions du système est décrite dans la BDF. La connaissance des effets locaux et causes externes (contenant les informations de *pin* et de fonctions périphériques) permet de connaitre l'ensemble des *pins* de la fonction au moment de la constitution de la BDF. Par comparaison de ces données avec les informations du modèle SysML actuel, nous en déduisons les changements d'interfaces. Finalement, il est donc possible d'attirer l'attention de l'expert spécifiquement sur les fonctions et flux fonctionnels dont il n'a pas encore connaissance. Les moyens d'attirer l'attention de l'expert peuvent être multiples et dépendront principalement des choix d'implémentations. Pour toutes ces fonctions, le maximum d'informations et de données possible est fourni (comme lors de la première génération de pré-AMDEC).

Pour l'ensemble des fonctions qui ne sont ni nouvelles ni modifiées, nous pouvons alors compléter la pré-AMDEC par lecture de la BDF. Les Modes de défaillances spécifiques sont ajoutés et les modes de défaillances marqués « supprimé » dans la BDF sont retirés. L'ensemble des informations marquées « supprimé » est retiré et les données apportées par l'expert sont ajoutées (Causes internes, Effets système spécifiques, Facteurs de cotation du risque, Moyens de réduction du risque).

4.3. Apport au travail de l'expert

Dans un processus traditionnel d'ingénierie système, les analyses de SdF sont menées pour différentes versions de la description du système. Dès les premières phases de spécifications, l'expert doit effectuer la tâche fastidieuse de lecture et compréhension du modèle système, d'identification des éventuelles modifications et ensuite seulement il est en mesure d'analyser les modes de défaillances et leurs impacts sur la sûreté de fonctionnement. Pourtant c'est cette seconde partie qui constitue son cœur de métier.

Notre processus de rédaction de pré-AMDEC à partir du modèle système permet dans un premier lieu de réaliser l'extraction des données utiles du modèle afin que l'expert puisse rapidement réaliser son analyse. La possibilité d'automatiser ce processus permet d'envisager d'effectuer des générations successives à chaque évolution du système. Cependant, nous avons vu que dans ce cas, le travail de l'expert n'est pas nécessairement réduit si le retour d'expérience des AMDEC des précédentes versions du système n'est pas pris en compte. L'intégration de la BDF à notre processus de génération de pré-AMDEC vise à adresser ce problème spécifiquement. Ainsi, en tirant parti des avantages propres à l'approche ISBM supportée par SysML, l'utilisation du processus de génération de pré-AMDEC combiné à la BDF permet de proposer tout au long de l'évolution du système des informations cohérentes mises en forme spécifiquement pour faciliter le travail de l'expert SdF et le guider dans la réalisation de son analyse des modes de défaillances.

III. Implémentation de l'algorithme de génération de pré-AMDEC fonctionnelle

La théorie de la rédaction de pré-AMDEC fonctionnelle a été mise au point dans le but d'optimiser le temps passé par l'expert SdF à manipuler le modèle système pour obtenir les informations utiles à ses études. De plus, l'analyse automatique de certaines parties du modèle permet une diminution du risque d'erreurs humaines améliorant ainsi la qualité de l'AMDEC. Cependant, nous l'avons vu très clairement, le travail de l'expert SdF reste primordial et central pour certifier de la qualité de l'AMDEC fonctionnelle finale.

Dans cette partie, nous allons nous pencher sur la problématique de l'implémentation logicielle de la méthode de rédaction de pré-AMDEC fonctionnelle. Elle vise à montrer l'applicabilité de la méthode à un contexte industriel réel dans le cadre de l'utilisation des outils largement répandus dans la communauté des ingénieurs systèmes. Actuellement les outils de modélisation SysML majoritairement utilisés sont : Artisan Studio d'ATEGO [ATEGO], Rational Rhapsody d'IBM [Rhapsody], Magic Draw de NO MAGIC [MagicDraw] et Papyrus/Topcased [Topcased], un projet open source basée sur Eclipse. Le point dur de l'implémentation de la méthode précédente est de réussir à s'interfacer avec ces outils de modélisation afin de pouvoir effectivement accéder aux données du modèle SysML. Nous verrons comment ces problèmes techniques impactent la méthode d'implémentation afin de maximiser l'adaptabilité du code. Nous verrons aussi quelles sont les techniques retenues dans le cas d'un interfaçage avec Artisan Studio plus particulièrement.

1. La technique d'implémentation

SysML est un langage dont l'utilisation industrielle progresse de façon régulière et par conséquent attire plusieurs éditeurs logiciels. L'existence de plusieurs plateformes incite à poser la problématique de l'accès aux données du modèle SysML. Cette problématique n'est pas nouvelle puisqu'elle s'est déjà posée pour le langage UML. C'est pourquoi l'OMG a travaillé à la standardisation d'un format d'échange de modèle UML. Le format XMI [OMG 2011b] vise à permettre la représentation de l'ensemble des données d'un modèle de façon organisée pour permettre aux plateformes logicielles de l'utiliser comme format de base, ou bien comme moyen d'import et d'export de modèle. Il s'agit d'un format basé sur XML et donc un format à balise organisant la représentation des données UML sous une forme textuelle. Ce format a été standardisé par l'OMG et est maintenu en parallèle des évolutions d'UML. Le format XMI permet aussi de représenter les informations des modèles basés sur les profiles d'UML, et est donc utilisé aussi comme format standard d'échange de modèle SysML. Cependant, le format XMI n'est pas parfait et la majorité des logiciels possèdent leur propre format propriétaire de stockage du modèle SysML. Ces formats propriétaires sont souvent plus complets et mieux structurés que le format XMI puisqu'ils sont dédiés à SysML en tant que langage à part entière, mais il n'est pas toujours possible d'y accéder, ou alors par utilisation des outils de développement de plugin proposés par la plateforme logicielle. Enfin, certains outils logiciels proposent des outils de génération de documents ou de code à partir de modèle SysML. Ces outils sont souvent paramétrables et permettent la création de fichier texte comportant les informations du modèle qui sont utiles. Nous pouvons résumer les avantages et les inconvénients de chaque solution :

Format XMI :

- Avantages : format standard et multi-plateforme. Il est a priori conçu pour permettre la transcription de l'intégralité des informations du modèle SysML.
- Inconvénients : traite SysML comme un profile UML ce qui crée une structuration du fichier de résultat peu ergonomique.

Format propriétaire :

- Avantages : format dédié à SysML, et bien structuré. L'intégralité des informations du modèle y est contenue.
- Inconvénients : format fermé, difficile d'utilisation et spécifique à chaque plateforme. Les outils de développement de la plateforme logicielle permettant d'y accéder ne sont pas systématiquement adaptés à l'implémentation de notre méthode.

Outil de génération de documents / code :

- Avantages : permet un accès à l'ensemble des données du modèle. Il met en forme les informations utiles pour la méthode selon une structuration paramétrable.
- Inconvénients : solution spécifique à chaque plateforme et parfois non disponible. La configuration de l'outil demande des compétences spécifiques et peut être source d'erreurs.

Au cours de la thèse, dans le but de s'accorder avec les pratiques de nos partenaires industriels, il a été décidé d'utiliser la plateforme Artisan Studio d'ATEGO. Les solutions précédentes sont toutes disponibles au sein d'Artisan Studio. En effet, la plateforme propose un outil d'exportation des modèles au format XMI, une interface de programmation de plugin permettant l'accès aux données au format Artisan et des outils de génération de document et de code. Finalement, la problématique de l'accès aux données systèmes n'ayant pas de solution plus logique que les autres, nous avons opté pour une implémentation visant à ségréger l'extraction des données et l'application des algorithmes de rédaction de l'AMDEC. Ainsi, il est possible d'envisager plusieurs modules d'extraction de données SysML permettant tous l'application des algorithmes.

2. Extraction des données

Dans un premier temps, nous avons fait le choix d'utiliser le format XMI puisqu'il s'agit de la seule solution multi-plateforme. Le but de l'extraction des données consiste à obtenir assez d'informations permettant l'exécution des algorithmes présentés précédemment. Ainsi il est nécessaire d'obtenir l'ensemble des éléments suivant :

$$f = \begin{Bmatrix} nom \\ AD \end{Bmatrix},\ AD = \begin{Bmatrix} nom \\ niveau \end{Bmatrix},\ flux = \begin{Bmatrix} P_1 \\ P_2 \end{Bmatrix},\ P = \begin{Bmatrix} nom \\ direction \\ possesseur \end{Bmatrix},$$

$$uc = \begin{Bmatrix} nom \\ UCD \end{Bmatrix},\ req = \begin{Bmatrix} nom \\ id \\ text \end{Bmatrix},\ c = \begin{Bmatrix} nom \\ type \end{Bmatrix}$$

Il est aussi nécessaire d'obtenir les informations permettant d'exécuter les opérations :

listFcn(AD) = {f / f ∈ AD}
AD = refinedBy(f)
f = refines(AD)
listPin(f) = {P / P ∈ f}
listPin(AD) = {P / P ∈ AD}
uc = refines(AD)

f = satisfiedBy(req)

req = satisfy(f)

c = allocatedTo(f)

f = allocatedFrom(c)

L'ensemble des données liées à une relation de possession : les opérations de listage des fonctions d'un *activity diagram* ou de *pin* et la donnée « possesseur » des *pins* sont déterminées au sein du XMI par analyse de la hiérarchie des balises. Les données sont quant à elles représentées le plus souvent par un couple : nom de balise et attribut xmi:type spécifique. L'exemple ci-dessous présente un extrait de la transcription XMI de l'*Activity Diagram* raffinant le *use case* : « Se déplacer » (utilisé dans notre modèle d'exemple de la Bicyclette : cf. Chapitre II) qui permet d'illustrer nos propos.

```
<packagedElement xmi:type = "uml:Activity" xmi:id = "_ee177fe4" name = "Se déplacer">
        <node xmi:type = "uml:OpaqueAction" xmi:id = "_31337da1"
        name = "Diriger la bicyclette">
                <argument xmi:type = "uml:InputPin" xmi:id = "_8180099e"
                name = "ConsignesUtilisateur" incoming = "_c76539eb"> </argument>
        </node>
        <node xmi:type = "uml:OpaqueAction" xmi:id = "_cb3b9dfb" name = "Pédaler">
                <result xmi:type = "uml:OutputPin" xmi:id = "_84028357"
                name = "EnergieMécanique" outgoing = "_a070f921"> </result>
                <result xmi:type = "uml:OutputPin" xmi:id = "_1a9a9f49
                name = "AbsenceEnergie" outgoing = "_2a99844d"> </result>
                <argument xmi:type = "uml:InputPin" xmi:id = "_afe3ff23
                name = "EnergieUtilisateur" incoming = "_cc0827a0"> </argument>
        </node>
        <node xmi:type = "uml:OpaqueAction" xmi:id = "_5b9507eb"
        name = "Recevoir consigne de freinage">
                <result xmi:type = "uml:OutputPin" xmi:id = "_26148661"
                name = "ActionMécanique" outgoing = "_2a61f757"> </result>
                <argument xmi:type = "uml:InputPin" xmi:id = "_e645ede0"
                name = "ConsigneFreinage" incoming = "_895d8f84"> </argument>
        </node>
        <node xmi:type = "uml:OpaqueAction" xmi:id = "_8591c2f0" name = "Ralentir">
                <argument xmi:type = "uml:InputPin" xmi:id = "_2513c5ff"
                name = "ActionMécanique" incoming = "_2a61f757"> </argument>
                <argument xmi:type = "uml:InputPin" xmi:id = "_629b4de2"
                name = "AbsenceEnergie" incoming = "_2a99844d"> </argument>
```

```
        </node>
        <node xmi:type = "uml:OpaqueAction" xmi:id = "_f4123e02" name = "Accélérer">
                <argument xmi:type = "uml:InputPin" xmi:id = "_c272e749"
                name = "EnergieMécanique" incoming = "_a070f921"> </argument>
        </node>
        <node xmi:type = "uml:ActivityParameterNode" xmi:id = "_66997335"
        name = "Utilisateur" outgoing = "_895d8f84 _cc0827a0 _c76539eb">
        </node>
        <edge xmi:type = "uml:ObjectFlow" xmi:id = "_2a99844d" source = "_1a9a9f49"
        target = "_629b4de2"> </edge>
        <edge xmi:type = "uml:ObjectFlow" xmi:id = "_2a61f757" source = "_26148661"
        target = "_2513c5ff"> </edge>
        <edge xmi:type = "uml:ObjectFlow" xmi:id = "_895d8f84" source = "_66997335"
        target = "_e645ede0"> </edge>
        <edge xmi:type = "uml:ObjectFlow" xmi:id = "_a070f921" source = "_84028357"
        target = "_c272e749"> </edge>
        <edge xmi:type = "uml:ObjectFlow" xmi:id = "_cc0827a0" source = "_66997335"
        target = "_afe3ff23"> </edge>
        <edge xmi:type = "uml:ObjectFlow" xmi:id = "_c76539eb" source = "_66997335"
        target = "_8180099e"> </edge>
</packagedElement>
```

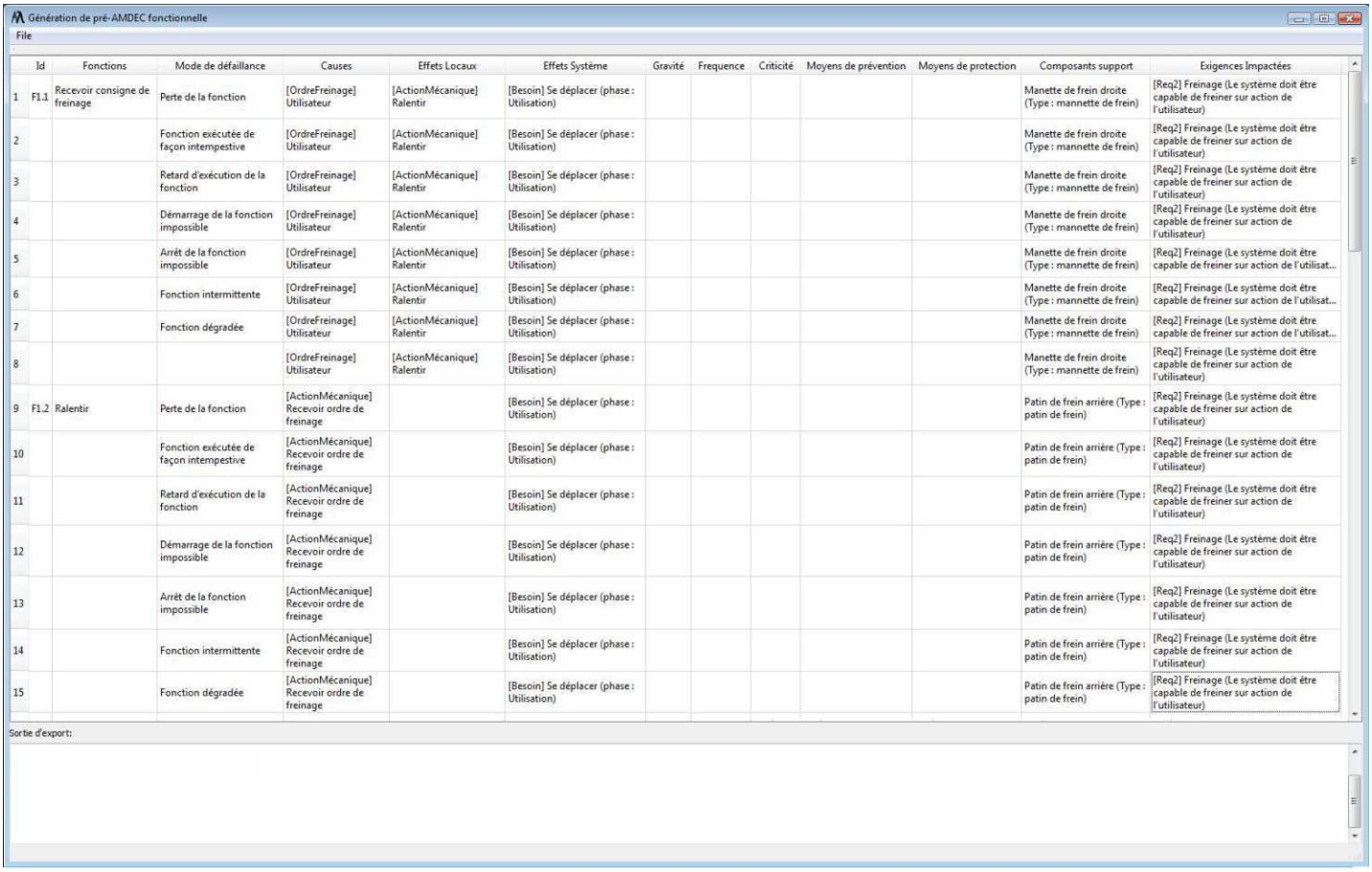
Nous pouvons y voir que les balises <node xmi:type = "uml:OpaqueAction" … > représentent les fonctions contenues dans l'AD « Se déplacer » représenté par la balise <packagedElement xmi:type = "uml:Activity" xmi:id = "_ee177fe4" name = "Se déplacer">. Ainsi lorsque une balise nodeoù l'attribut xmi:type est égal à "uml:OpaqueAction" alors l'attribut name de cette balise nous donne le nom de la fonction, et l'attribut name de la balise parente nous donne le nom de l'ADdans lequel elle se trouve. De façon similaire, chaque balise edge où l'attribut xmi:type = "uml:ObjectFlow" représente un flux dont les attributs source et target permettent d'identifier les *pins* reliés par ce flux. Nous pouvons noter que ces attributs source et target sont constitués d'un identifiant unique alphanumérique que chaque entité du XMI possède. Cet identifiant unique est constitué dans le XMI de 5 séries de chiffres séparées par des tirets (exemple : xmi:id = "_8180099e-fd49-4c74-81a0-83d9f0c442ac"), cependant par souci de lisibilité nous n'avons gardé que la première série dans l'exemple ci-dessus puisque cela ne crée pas de perte d'unicité des identifiants. À travers les xmi:id surlignés dans l'exemple, nous pouvons observer comment les flux permettent effectivement de relier les i des différentes fonctions. Nous observons aussi la balise node d'attribut xmi:type = "uml:ActivityParameterNode" qui correspond à un *pin* de bordure de l'*activity diagram* (de type input car il possède un attribut outgoing indiquant les flux qui sortent de lui). Les mêmes raisonnements (à quelques adaptations mineures près) peuvent être appliqués pour l'ensemble des entités : exigences, composants, besoins,…

Concernant les relations entre entités : *refine*, *allocate* et *satisfy*, nous avons vu précédemment qu'elles héritaient toutes du même type d'entités UML : *Relationship*. Ainsi dans le fichier XMI, ces relations seront représentées par une balise packageElement avec xmi:type = "uml:Abstraction" ou xmi:type = "uml:Dependency" (*Abstraction* et *Dependency* héritent toutes les deux de *Relationship*). Ces balises générales de relation permettent de connaitre les éléments reliés grâce aux attributs client et supplier (client et fournisseur). Enfin pour connaitre la nature exacte de la relation, deux cas de figure sont possibles :

- Si xmi:type = "uml:Dependency" et les identifiants de client et supplier correspondent à des *opaqueAction* et des *uses cases*, alors il s'agit d'une relation de *refine* utile à la pré-AMDEC telle que le supplier est raffiné par le client.
- Sinon, nous cherchons une balise de profile SysML. Dans le XMI, toutes les informations sont représentées comme si elles appartenaient à un modèle UML et les informations propres aux stéréotypes de SysML sont ajoutées à la fin du fichier. Ainsi pour *satisfy* et *allocate*, dans la partie traditionnelle UML, nous trouvons les balises packageElement où xmi:type = "uml:Abstraction" et il faut ensuite trouver les balises dont l'attribut base_Abstraction correspond à l'identifiant de l'abstraction. Si la balise est une balise <sysml:Satisfy … > alors il s'agit d'une relation *satisfy* et si c'est une balise <sysml:Allocate … > alors il s'agit d'une relation *allocate*. L'exemple ci-dessous illustre le cas d'une relation d'allocation.

```
<packagedElement xmi:type = "uml:Abstraction" xmi:id = "_a683c2e5" client = "_d7a84ec6" supplier =
"_78337ddb "></packagedElement>
...
<sysml:Allocate base_Abstraction = "_a683c2e5" xmi:id = "_b719c717_a683c2e5" />
```

Ainsi, nous sommes capables d'accéder à l'ensemble des données et relations utiles. Chaque entité et chaque relation vont être stockées dans une structure orientée objet qui permettra d'appliquer les algorithmes de rédaction ensuite. Le module d'extraction de données doit manipuler efficacement un format de fichier type XML pour traiter le fichier XMI comme nous l'avons présenté. C'est une des raisons pour laquelle il a été décidé d'avoir recours au langage C++ supporté par le framework Qt qui possède notamment une API logicielle efficiente pour la lecture et l'écriture de fichier de type XML. Le langage C++ est un langage orienté objet propice à une structuration des données telles qu'elles sont organisées dans le modèle SysML permettant après l'extraction des données, une application plus aisée des algorithmes. Qt est un framework qui propose de nombreuses autres fonctionnalités qui nous sont utiles comme des éléments d'interface graphique de type tableur permettant la représentation de la pré-AMDEC. Un aperçu de l'outil logiciel est présenté dans la Figure III.11.



	Id	Fonctions	Mode de défaillance	Causes	Effets Locaux	Effets Système	Gravité	Frequence	Criticité	Moyens de prévention	Moyens de protection	Composants support	Exigences Impactées
1	F1.1	Recevoir consigne de freinage	Perte de la fonction	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer (phase : Utilisation)						Manette de frein droite (Type : mannette de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)
2			Fonction exécutée de façon intempestive	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer (phase : Utilisation)						Manette de frein droite (Type : mannette de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)
3			Retard d'exécution de la fonction	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer (phase : Utilisation)						Manette de frein droite (Type : mannette de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)
4			Démarrage de la fonction impossible	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer (phase : Utilisation)						Manette de frein droite (Type : mannette de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)
5			Arrêt de la fonction impossible	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer (phase : Utilisation)						Manette de frein droite (Type : mannette de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisat...
6			Fonction intermittente	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer (phase : Utilisation)						Manette de frein droite (Type : mannette de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisat...
7			Fonction dégradée	[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer (phase : Utilisation)						Manette de frein droite (Type : mannette de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisat...
8				[OrdreFreinage] Utilisateur	[ActionMécanique] Ralentir	[Besoin] Se déplacer (phase : Utilisation)						Manette de frein droite (Type : mannette de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)
9	F1.2	Ralentir	Perte de la fonction	[ActionMécanique] Recevoir ordre de freinage		[Besoin] Se déplacer (phase : Utilisation)						Patin de frein arrière (Type : patin de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)
10			Fonction exécutée de façon intempestive	[ActionMécanique] Recevoir ordre de freinage		[Besoin] Se déplacer (phase : Utilisation)						Patin de frein arrière (Type : patin de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)
11			Retard d'exécution de la fonction	[ActionMécanique] Recevoir ordre de freinage		[Besoin] Se déplacer (phase : Utilisation)						Patin de frein arrière (Type : patin de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)
12			Démarrage de la fonction impossible	[ActionMécanique] Recevoir ordre de freinage		[Besoin] Se déplacer (phase : Utilisation)						Patin de frein arrière (Type : patin de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)
13			Arrêt de la fonction impossible	[ActionMécanique] Recevoir ordre de freinage		[Besoin] Se déplacer (phase : Utilisation)						Patin de frein arrière (Type : patin de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)
14			Fonction intermittente	[ActionMécanique] Recevoir ordre de freinage		[Besoin] Se déplacer (phase : Utilisation)						Patin de frein arrière (Type : patin de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)
15			Fonction dégradée	[ActionMécanique] Recevoir ordre de freinage		[Besoin] Se déplacer (phase : Utilisation)						Patin de frein arrière (Type : patin de frein)	[Req2] Freinage (Le système doit être capable de freiner sur action de l'utilisateur)

Figure III.11 : Capture d'écran de l'outil logiciel de rédaction de pré-AMDEC

3. Ouverture de l'outil à d'autres sémantiques

Au cours de la thèse, nous avons eu l'occasion de travailler avec un industriel à l'implémentation de cette méthode de génération de pré-AMDEC fonctionnelle, dans un cadre sémantique de modèle SysML différent. En effet, les pratiques de cet industriel reposent sur l'utilisation de diagramme interne de bloc pour la réalisation de l'analyse fonctionnelle. Ainsi les fonctions sont réifiées par des blocks et des parts à la place de nos *opaqueActions*.

Dans ce cas, c'est la sémantique du modèle Système qui varie. Il faut alors s'assurer que l'ensemble des éléments et relations précédemment formalisés existe toujours dans le nouveau cadre. Ici, l'AD est remplacé par l'IBD et la fonction par un part. Le concept de « type de fonction » réifié par des *blocks*, comme nous avions des « types de composants », est nouveau et peut-être intégré à nos algorithmes et à la pré-AMDEC. Dans le chapitre II, nous avions évoqué la possibilité d'utiliser nous aussi cette solution pour l'analyse fonctionnelle. Cependant nous avions préféré l'utilisation d'AD pour permettre une représentation claire et distincte des fonctions et des composants. Ainsi la question de la distinction des fonctions et des composants se pose puisqu'ils sont tous représentés par des *parts*. La solution consiste à étudier avec précision les relations d'allocation fonctionnelle réifiées dans le modèle système en SysML par des relations *allocate* entre *parts* (dans ce cadre sémantique). En effet, les *parts* qui sont alloués sont des fonctions, et ceux à qui nous allouons d'autres *parts* sont des composants, ce qui accroit fortement l'importance de cette allocation fonctionnelle. Nous pouvons noter que ce mécanisme rend impossible la génération de pré-AMDEC

fonctionnelle tant que la description organique et l'allocation fonctionnelle ne sont pas réalisées, car alors il est impossible de distinguer fonctions et composants. Cela minimise considérablement l'intérêt de la méthode de génération de pré-AMDEC mais ne l'empêche pas de s'appliquer malgré tout.

IV. Conclusion

Ce chapitre présente une première étape vers la valorisation de l'IS supporté par SysML pour l'intégration de COTS, tout en adhérant au cadre méthodologique MéDISIS visant à rapprocher les études de SdF du processus d'IS. Cette première étape est la rédaction de pré-AMDEC fonctionnelle à partir de l'analyse des données du modèle système dont la réalisation en SysML a été détaillée dans le chapitre précédent. Le processus de rédaction automatique de pré-AMDEC représente alors un outil permettant une initiation des analyses de SdF dès l'analyse fonctionnelle. Cet outil permet aussi de maintenir et de mettre à jour la pré-AMDEC à chaque évolution du système pour un coût en temps d'expert réduit.

Pour cela, nous avons d'abord étudié le méta-modèle sous-jacent à l'AMDEC fonctionnelle. Ceci nous a permis de mettre en exergue les éléments à extraire du modèle système en SysML utile à la rédaction de l'AMDEC fonctionnelle. Nous avons alors mis au point les algorithmes réalisant la recherche de ces éléments SysML. Ces éléments sont ensuite mis en forme pour rédiger la pré-AMDEC fonctionnelle. Le principe de cette pré-AMDEC fonctionnelle est de réunir sous une forme proche de l'AMDEC finale l'ensemble des informations provenant du modèle système et utiles à l'expert.

À travers l'utilisation de cette pré-AMDEC nous avons défini le besoin d'intégrer les itérations successives des rédactions de pré-AMDEC. Pour cela, la BDF a été définie pour répondre aux besoins spécifiques de notre processus de rédaction. Ceci était nécessaire, car ce processus est le premier processus à s'intégrer au cadre méthodologique MéDISIS qui repose sur l'architecture fonctionnelle plutôt que sur l'architecture organique.

Finalement, nous avons présenté l'implémentation du processus de rédaction d'AMDEC en un outil logiciel. Cet outil est actuellement utilisé dans le cadre du projet LEA que nous présentons au Chapitre V. De plus, nous avons vu que les principes de la génération de pré-AMDEC restent vrais dans un autre cadre sémantique de modélisation SysML puisque l'outil a été adapté au processus industriel dans le cadre d'un projet annexe de la thèse.

L'approche ISSM et la méthodologie MéDISIS augmentée de ce nouvel outil permettent donc de participer à la validation des systèmes à COTS. Cependant, seul l'aspect fonctionnel du COTS est alors validé. L'aspect organique n'est pour l'instant pas abordé par nos travaux. Nous allons donc maintenant étudier les moyens envisageables pour compléter notre méthode.

Chapitre IV. Intégration des études FIDES à la méthodologie MéDISIS

I. Introduction

Nous avons pu voir, dans le chapitre précédent, comment l'ISBM pouvait être outillée pour permettre la réalisation d'AMDEC fonctionnelle. Ceci consiste en la première étape de notre solution pour faciliter l'intégration de COTS dans un système critique grâce à une approche d'ingénierie système supportée par les modèles SysML. En effet, lors de l'AMDEC fonctionnelle, ce sont les fonctions du COTS qui sont étudiées. Ainsi l'AMDEC fonctionnelle n'est pas impactée par la connaissance limitée de l'architecture organique du COTS. Cependant, les études de sûreté de fonctionnement analysant la description fonctionnelle du système ne suffisent pas à la validation des exigences de fiabilité d'un système complexe. Les aspects matériels ne peuvent être ignorés. La méthodologie FIDES que nous allons aborder dans ce chapitre est née de ce besoin d'évaluer la fiabilité des systèmes complexes. FIDES, qui s'accorde particulièrement bien avec nos approches ISBM, permet l'estimation du taux de défaillance des composants électroniques présents dans le système. Par extension, FIDES permet l'estimation du taux de défaillance d'un système constitué par des composants électroniques (présent dans le guide FIDES). L'apport spécifique de FIDES vis-à-vis d'autres méthodes d'évaluation de taux de défaillance est la prise en compte du processus de développement d'un produit et de son environnement d'utilisation dans les paramètres d'évaluation de la fiabilité.

La méthodologie MéDISIS, présentée dans [David 2009], n'est pas dénuée de passerelles entre l'ISBM et l'évaluation de fiabilité des composants d'un système. En effet, le processus de génération de modèle Altarica DF, répond à cet objectif. Lors de l'établissement de ce processus de génération, la Base de données des Comportements Dysfonctionnels (BCD) a été conçue pour augmenter l'efficience de ce processus. Cette base de données vise à stocker et organiser les informations dysfonctionnelles utiles à la réalisation d'études de types Altarica DF. Cependant, Altarica DF ne peut pas nativement intégrer l'étude des COTS dont on ne connait pas les caractéristiques de fiabilité et c'est pour cela que FIDES est indispensable à notre réflexion.

Dans ce chapitre, la BCD sera définie dans sa forme la plus récente. En effet, la BCD actuelle résulte de l'ensemble des travaux menés dans le cadre de cette thèse s'appuyant sur la méthodologie MéDISIS. À partir de cette description de la BCD, nous montrerons les connexions possibles avec la méthode FIDES préalablement détaillée. Enfin, l'intérêt de l'approche ISBM pour l'application de la méthode FIDES sera illustré.

II. La BCD de MéDISIS

1. Principes

La méthodologie MéDISIS vise à intégrer efficacement les analyses de SdF au processus global d'IS. Pour cela, elle propose des processus de traitements de l'information et des connaissances, incluant leur création, expression, analyse, pérennisation et réutilisation répondant aux objectifs explicités ci-après :

1. Faciliter la transmission de connaissances entre équipes et entre les différentes activités d'ingénierie.

2. Accélérer la réalisation des études de SdF.
3. Organiser l'exploitation commune des connaissances sous forme de modèles.
4. Permettre la réutilisation des connaissances entre projets : par exemple favoriser le retour d'expérience.
5. Améliorer la cohérence et la qualité des analyses SdF.

MéDISIS intègre, de manière centrale, aux différents processus une couche de persistance de l'information : la BCD. Celle-ci contribue à répondre aux attentes du point 1 en permettant une agrégation structurée et maitrisée des connaissances, en proposant à chaque expert, détenteur d'un point de vue, une structure centrale permettant la pérennisation des informations issues de ses analyses. La BCD en répondant au point 3 apportera de plus son caractère multi-vue, et multi-langage et répondra aussi aux points 2 et 4. En effet, elle est le résultat même de l'expression de ces besoins à savoir permettre un accès rapide aux informations de sûreté de fonctionnement issues du REX, aussi bien pour être exploitées que pour être archivées. Enfin, la BCD permettra une cohérence des analyses de SdF grâce à l'architecture de son méta-modèle qui relie entre eux les informations fonctionnelles, métiers et les résultats des analyses ce qui répond en partie aux objectifs du point 5.

La pérennisation des informations dysfonctionnelles résultantes de l'application des processus MéDISIS dans la BCD permet d'inscrire l'utilisation de MéDISIS dans un cycle d'itérations successives d'amélioration des résultats. Cependant, de la bonne structuration de la BCD dépend l'efficacité de l'agrégation des connaissances métiers et de l'évolution de chaque processus de traduction qui compose MéDISIS. Pour remplir ces objectifs, le méta-modèle de la BCD doit être :

- Complet, et ainsi former un noyau commun d'information métier, organisé et accessible reprenant les concepts issus de l'analyse des méta-modèles des langages cible.
- Structuré, de façon à modéliser toutes les relations et échanges possibles entre tous ces concepts afin de pourvoir à tous les besoins des différents processus mis en œuvre.

Une première version de cette BCD et de son méta-modèle a été introduite dans [David 2009] et [David et al. 2010]. La BCD que nous décrivons dans ce chapitre est conçue pour améliorer encore le support des analyses de SdF au sein d'une approche ISBM. Elle permet de centraliser un vaste champ de données couvrant ainsi l'ensemble des informations utiles aux outils et langages de SdF.

2. Le méta-modèle de la BCD

Le méta-modèle de la BCD est représenté Figure IV.1 et Figure IV.2. La Figure IV.1 présente les concepts composant notre BCD ainsi que les entités SysML qui sont utilisées pour les réifier. La Figure IV.2, quant à elle, présente les allocations existantes entre les différents éléments de la BCD. Le méta-modèle est représenté à travers ces deux figures par souci de clarté de la représentation de la BCD, mais elle forme une unique structure d'organisation des concepts de la BCD.

La BCD est articulée autour du couple type de composant (« TypeDeComposant ») / mode de défaillance (« ModeDeDéfaillance »). La présence du *block* « TypeDeComposant » dans la structure de la BCD permet, lors de l'analyse d'un modèle système, la détermination de la présence d'un type de composant conjointement dans la BCD et dans le modèle système, par comparaison des *blocks* du modèle système avec les « TypeDeComposant » de la BCD.

Chaque MdD est décrit grâce au *block* « ModeDeDéfaillance » et à l'ensemble de ses propriétés (« DéfParamètre », « DéfOperation », …). « TypeDeComposant » et « ModeDeDéfaillance » sont reliés par deux relations distinctes : l'*association* et la *generalization*. L'*association* permet de réifier le fait qu'un type de composant puisse défaillir selon un certain nombre de MdD. La multiplicité « * » côté « ModeDeDéfaillance » indique la possibilité pour « 1 » type de composant d'avoir, entre 0 et n MdD ($n \in \mathbb{N}$). La *generalization* permet aux *blocks* « ModeDeDéfaillance » d'hériter de l'interface et de tous les paramètres et propriétés du « TypeDeComposant », ils seront donc utilisables pour décrire les dysfonctionnements induits par le MdD. De plus, puisqu'ils partagent les mêmes interfaces, le « TypeDeComposant » peut être remplacé par le « ModeDeDéfaillance » afin de réaliser une étude par injection de faute. Cette utilisation de la BCD a été illustrée dans [Cressent et al. 2011c], elle consiste à la génération d'un modèle Simulink de simulation système (cf. Chapitre V).

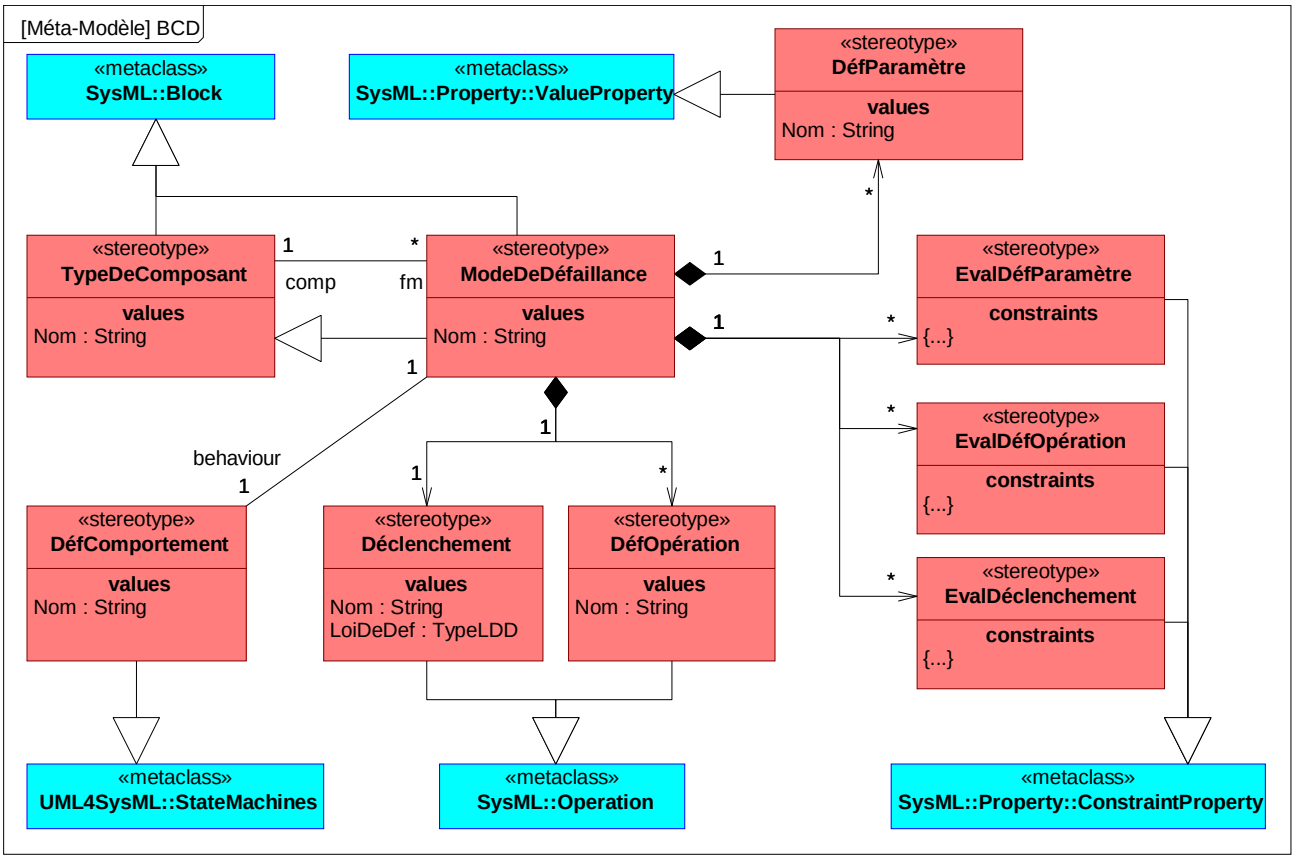


Figure IV.1 : Méta-modèle de la BCD

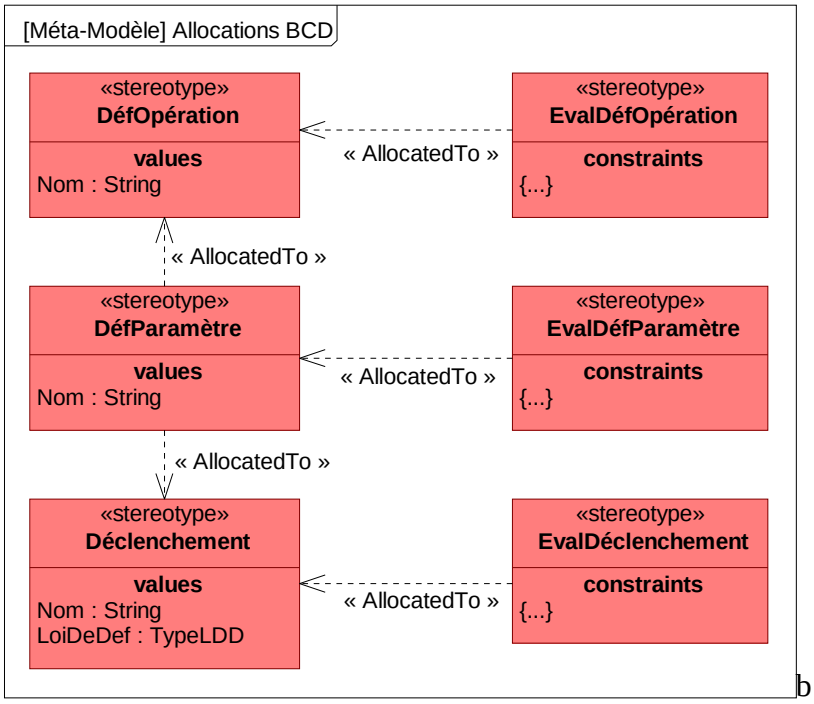


Figure IV.2 : Allocations entres les concepts de la BCD

Chaque MdD est décrit par un ensemble de paramètres : opérations (modélisant le dysfonctionnement), contraintes (permettant une description détaillée des mécanismes de défaillance) et un STM (synchronisant l'ensemble des comportements possible du MdD). Le dysfonctionnement est modélisé en deux parties : son déclenchement, et ses effets. Le déclenchement est modélisé avec une opération « Déclenchement » et les effets par des opérations : « DéfOpération ». « Déclenchement » possède un nom et une propriété « LoiDeDéf » de type « TypeLDD » permettant de décrire le type de loi de défaillance régissant le déclenchement du MdD : Exponentielle, Weibull, LogNormale ou Déterministe. Ainsi le déclenchement peut être décrit comme aléatoire ou déterministe comme le suggère la norme IEC 61508 [IEC 61508]. Cette *operation* « Déclenchement » peut être détaillée en utilisant des *constraint properties* stéréotypés « EvalDéclenchement » (qui pourront être représentées dans un ParD). Ces *constraint properties* modélisent les relations entre les *value properties* du composant (grâce à la *generalization* entre le type de composant et le MdD) et les *value properties* propres au MdD qui sont stéréotypés « DéfParamètre ». La Figure IV.3 présente un exemple de modélisation d'un MdD dans la BCD pour un composant quelconque. Nous pouvons observer que l'*opération* et les *value properties* de « ComposantA » sont bien héritées par son mode de défaillance : « MdD de A ». On distingue aussi que le MdD dispose d'*opérations* supplémentaires dans les sous-parties : « Déclenchement » et « DéfOpérations », de paramètres spécifiques dans la partie « DéfParamètres » et de *constraint properties* dans les parties « EvalDéclenchement » et « EvalDéfOpérations ».

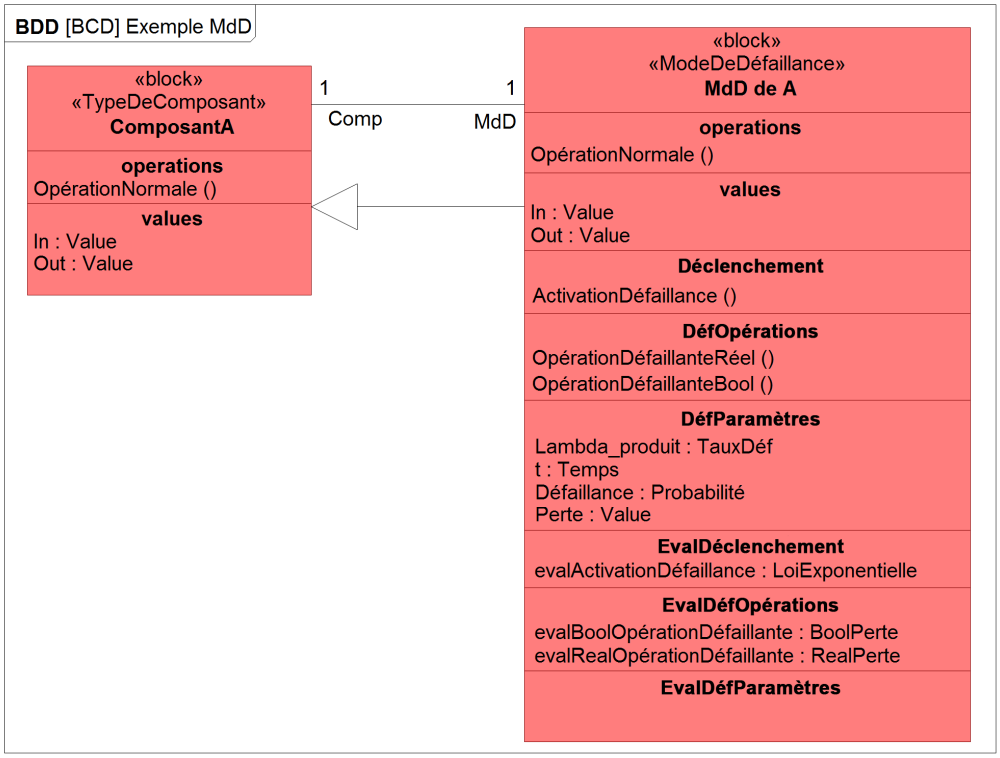


Figure IV.3 : Exemple d'un mode de défaillance représenté dans la BCD

Dans le cas d'une défaillance aléatoire, le déclenchement doit être quantifié à l'aide d'une loi de défaillance. Les paramètres spécifiques au MdD participants à la description de cette loi de défaillance sont stéréotypés « DéfParamètre ». Par soucis de cohérence de la modélisation, ces paramètres sont alloués à l'opération « Déclenchement » (comme cela est suggéré par la Figure IV.2 du méta-modèle). De même, la *constraint property* « EvalDéclenchement » est allouée à l'opération « Déclenchement » dont elle détaille la description. Occasionnellement, un « DéfParamètre » peut lui-même être issu de paramètres divers spécifiques au MdD, au composant ou à l'environnement. Dans ce cas, une *constraint property* est de nouveau utilisée pour modéliser cette description détaillée. Cette *constraint property* est alors stéréotypée « EvalDéfParamètre » et est allouée au « DéfParamètre » qu'elle décrit. Pour une défaillance déterministe, la *constraint property* « EvalDéclenchement » doit modéliser l'expression déterministe adéquate à la définition du déclenchement. Comme précédemment, les « DéfParamètre » et « EvalDéfParamètre » supportent cette modélisation. Les paramètres spécifiques au calcul du déclenchement peuvent être réduits à quelques *value properties*, comme dans l'exemple de la Figure IV.4 présentant un exemple d'« EvalDéclenchement ». Mais le déclenchement peut-être défini en prenant compte de nombreux autres paramètres comme nous le verrons, dans la suite de ce chapitre, avec la méthodologie FIDES.

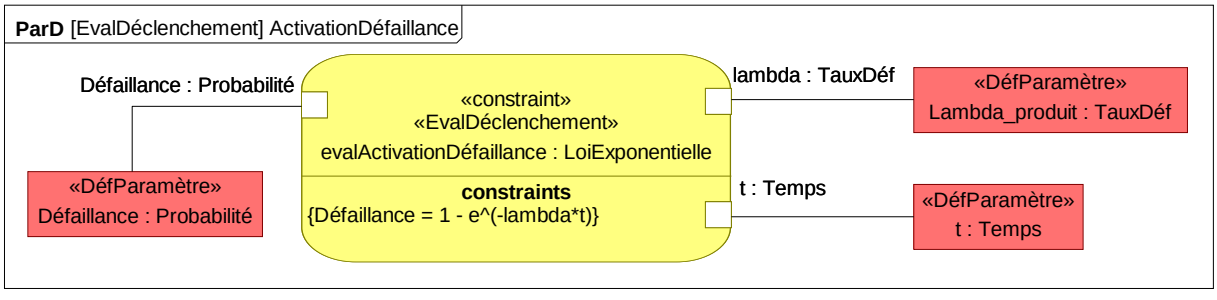


Figure IV.4 : Exemple de description détaillée du déclenchement

Le deuxième concept permettant la modélisation du dysfonctionnement est l'effet du dysfonctionnement modélisé par un ensemble d'*opérations* stéréotypées « DéfOpération ». Quand un composant défaille, les *opérations* qu'il réalise normalement ne sont alors plus exécutées ou bien différemment, voire même certaines nouvelles *opérations* propres au dysfonctionnement sont réalisées. Ces *opérations* correspondent aux entités devant être modélisées en utilisant le stéréotype « DéfOpération ». Comme avec le déclenchement, les « DéfOperation » peuvent être décrits en détail à l'aide d'une *constraint property* : « EvalDéfOpération » qui est allouée à l'*opération* qu'elle détaille. Cette *constraint property* peut avoir recourt aux paramètres du composant, ainsi qu'à des paramètres spécifiques qui sont représentés dans la BCD sous le stéréotype « DéfParamètres » (ils peuvent eux-mêmes être détaillés par une *constraint property* « EvalDéfParamètres » comme précédemment). Ces paramètres spécifiques sont par exemple, l'état de défaillance d'un composant, un débit permettant de qualifier une fuite, ou bien la puissance non fournie par une alimentation défaillante. La Figure IV.5 présente deux « EvalDéfOpération » décrivant à deux niveaux de granularité différents le dysfonctionnement d'un composant induit par un de ses modes de défaillance. Le premier (à gauche) repose sur une logique booléenne et le second (à droite) repose sur un calcul quantitatif de l'impact de la défaillance. Le paramètre de défaillance « Perte » est constant dans l'exemple, mais il pourrait être issu d'un calcul défini par une *constraint property* « EvalDéfParamètre » impliquant, par exemple, d'autres « DéfParamètres » et des paramètres fonctionnels hérités tel que la pression ou la température.

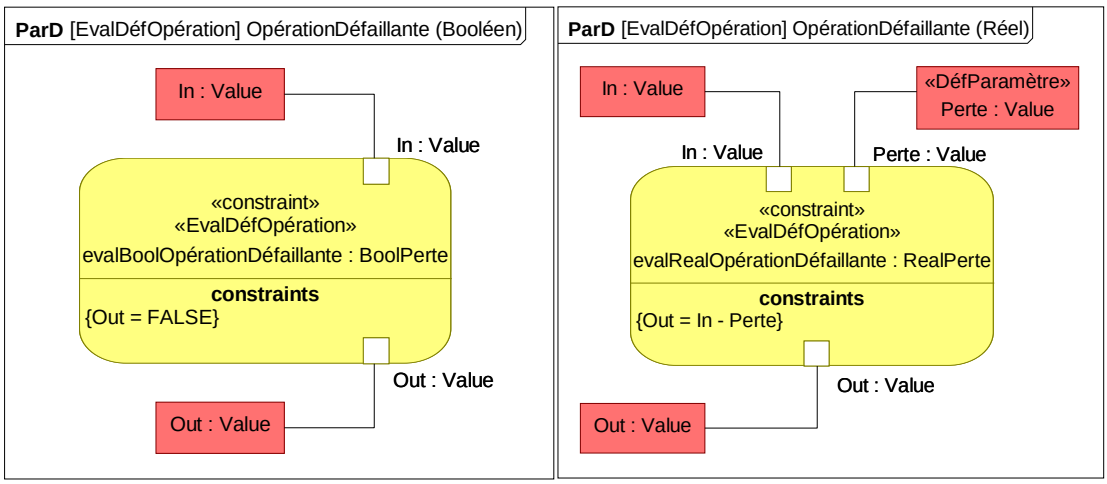


Figure IV.5 : Exemple de « DéfOpération » à deux niveaux de granularité

Le dernier concept présenté dans le méta-modèle de la BCD est le STM associé au MdD, stéréotypé : « DéfComportement ». Chaque MdD possède un STM « DefComportement » présentant l'ensemble des états pouvant se produire pendant la vie du projet, aussi bien un état de fonctionnement normal que des états défaillants. Le passage de l'état normal à un état défaillant est piloté par l'*opération* « Déclenchement ». Le STM permet aussi de piloter l'exécution des opérations normales (héritées de « ComponentType ») ou dysfonctionnelles (« DéfOpération ») dans chacun des états défaillants. Les transitions entre états défaillants peuvent être issues de la description de « Déclenchement » ou bien selon d'autres paramètres modélisés avec des « DéfParamètre ». Un exemple simple de « DéfComportement » est présenté dans la Figure IV.6. Sur cette figure le STM représente deux états : Normal et Défaillant. La transition de l'état Normal à l'état Défaillant pourra être coordonnée avec l'opération « ExempleDéclenchement » (Figure IV.3). On distingue aussi que dans chaque état représenté, des opérations différentes sont réalisées :

- Dans l'état Normal : l'opération OpérationNormale héritée du composant est réalisée.
- Dans l'état Défaillant : la « DéfOpération » OpérationDéfaillanteRéel propre au MdD est réalisée.

Ici le choix du niveau de granularité est fait par l'appel à OpérationDéfaillanteRéel plutôt qu'à OpérationDéfaillanteBool dans l'état Défaillant. Cependant, il serait possible de définir plusieurs états défaillants réalisant respectivement l'une ou l'autre des ces opérations.

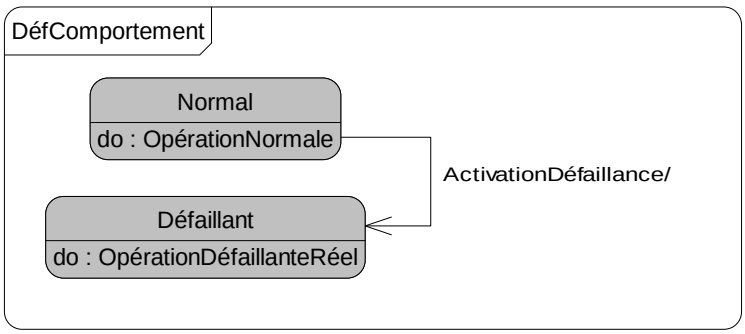


Figure IV.6 : Exemple de STM « DéfComportement »

Finalement, la structure de la BCD offre de nombreuses possibilités de modélisation des données dysfonctionnelles obtenues au cours des études de SdF de multiples projets. Dans la BCD, le déclenchement et les comportements d'un mode de défaillance sont décrits et modélisés. Ils sont synchronisés grâce à une spécification de l'évolution dynamique du composant (grâce au « DéfComportement »). De plus, l'ensemble des paramètres intervenants dans la modélisation de ces concepts peut être détaillé avec cohérence. Avoir recours à des *constraint properties* pour représenter les modèles logiques et mathématiques régissant l'ensemble des concepts de la BCD permet de s'adapter à différents niveaux de granularité en fonction du niveau de connaissances des mécanismes modélisés. De plus, l'intégration de modèle formel exécutable par des outils externes s'en trouve facilitée (cf. Chapitre II §V.2.b). Le support SysML de la BCD lui confère les moyens de modéliser tout type de composant et de mode de défaillances.

III. FIDES

1. Présentation

FIDES est une méthodologie d'évaluation de fiabilité relativement récente (première version en 2004 [FIDES 2004]) qui, fort de son succès en Europe dans le domaine aéronautique et de la défense, est régulièrement mise à jour (version la plus récente en 2011 [FIDES 2011]). La meilleure présentation de FIDES reste celle qui introduit le guide FIDES (norme UTE 80811) :

« Les objectifs du Guide FIDES sont d'une part de permettre une évaluation réaliste de la fiabilité des produits électroniques, y compris dans les systèmes qui rencontrent des environnements sévères ou très bénins (stockage), et d'autre part de fournir un outil concret pour la construction et la maîtrise de cette fiabilité.

Ses principales caractéristiques sont :

- L'existence de modélisations tant pour les composants Électriques, Électroniques et Électromécaniques que pour les cartes électroniques ou certains sous-ensembles.
- La mise en évidence et la prise en compte de tous les facteurs technologiques et physiques qui ont un rôle identifié dans la fiabilité.
- La prise en compte précise du profil de vie.
- La prise en compte des surcharges accidentelles électriques, mécaniques et thermiques (ou overstress).
- La prise en compte des défaillances liées aux processus de développement, de production, d'exploitation et de maintenance.
- La possibilité de distinguer plusieurs fournisseurs d'un même composant.

Au travers de l'identification des contributeurs à la fiabilité, qu'ils soient technologiques, physiques ou de processus, le Guide FIDES permet d'agir sur les définitions et dans tout le cycle de vie des produits pour améliorer et maîtriser la fiabilité. »

Comme cette présentation l'indique, FIDES est une méthode permettant d'évaluer le taux de défaillance d'un produit dont les composants sont dans le périmètre de FIDES. Ce périmètre correspond principalement aux composants « Électriques, Électroniques et Électromécaniques ». L'établissement des modèles d'évaluation de FIDES repose sur deux aspects : la physique de défaillance des composants et le retour d'expérience de leur utilisation. Enfin l'évaluation elle-même intègre la prise en compte de très nombreux paramètres système et environnementaux qui sont généralement définis lors des étapes d'ingénierie système, ce qui nous a poussé à étudier cette méthodologie.

2. Les principes de la méthodologie

La méthodologie FIDES repose sur l'évaluation du taux de défaillance (λ) d'un produit tout au long de son cycle de vie. Ce taux λ dépend de la phase de vie considérée et de paramètres reflétant l'impact du processus de conception, du processus de fabrication, du fabricant des composants, etc… Avant de détailler le modèle mathématique utilisant ces paramètres, nous allons définir deux concepts indispensables à l'utilisation de FIDES :

- Produit (Product) : dans le guide FIDES, correspond à l'entité pour laquelle la fiabilité est étudiée.
- Article (Item) : définit une entité élémentaire pour laquelle il est possible d'étudier la fiabilité. Dans le cadre FIDES, une résistance ou un condensateur sont des articles.

Ainsi, la méthodologie FIDES permet d'évaluer la fiabilité d'un produit composé d'articles dont la fiabilité est décrite dans le guide FIDES.

Le guide FIDES [FIDES 2011] se compose de trois parties principales :

- La première décrit les fondements mathématiques de l'évaluation de fiabilité d'un produit.
- La seconde partie contient les modèles mathématiques spécifiques à chaque article. Les résultats de ces modèles spécifiques sont utilisés par les modèles mathématiques présentés dans la première partie.

- La troisième et dernière partie présente la méthode de conduite d'un audit FIDES dans une entreprise souhaitant utiliser la méthodologie FIDES. Cet audit vise à évaluer la prise en compte de la SdF dans le processus industriel. Les résultats de cet audit permettent de valuer certains paramètres utilisés lors de l'évaluation de fiabilité du produit.

L'équation principale de la méthodologie FIDES définit que le taux de défaillance d'un produit est la somme des taux de défaillance de ses articles :

$$\lambda_{\text{product}} = \sum_{\text{item}} \lambda_{\text{item}} \tag{1}$$

Cette équation principale fait donc l'hypothèse de l'indépendance des défaillances des articles les unes par rapport aux autres. Le λ_{item} est ensuite composé de 3 facteurs :

$$\lambda_{\text{item}} = \lambda_{\text{physical}} \times \Pi_{\text{PM}} \times \Pi_{\text{Process}} \tag{2}$$

- $\lambda_{\text{physical}}$ décrit les contributions physiques à la défaillance.
- Π_{PM} (PM = Part Manufacturing) représente la qualité et la maîtrise technique de fabrication de l'article. L'évaluation de ce paramètre peut varier selon la nature de l'article considéré.
- Π_{Process} traduit la qualité et la maîtrise technique du processus de spécification, de développement, de fabrication et d'utilisation du produit contenant l'article.

Π_{PM} and Π_{Process} sont des termes multiplicatifs qui permettent de mitiger les contributions purement physiques représentées par $\lambda_{\text{physical}}$. Lors d'une étude FIDES complète, Π_{process} est calculé à partir des résultats de l'audit évoqué précédemment et Π_{PM} est calculé en évaluant le processus de fabrication du sous-traitant. Pour une étude préliminaire, ou lorsque les informations pour ces calculs sont insuffisantes, le guide FIDES prévoit la possibilité d'assigner des valeurs par défaut à ces termes. Par exemple, Π_{Process} vaut 4 par défaut et il est conseillé de mettre Π_{PM} à 1.7 pour les composants actifs et à 1.6 pour les composants passifs et pour les COTS. Nous détaillerons les calculs de Π_{PM} et Π_{Process} par la suite, mais nous allons d'abord étudier les équations régissant l'évaluation de $\lambda_{\text{Physical}}$. Pour cela, nous allons regarder deux équations proposées par le guide FIDES :

$$\lambda_{\text{physical}} = \sum_{\text{i}}^{\text{phases}} \left(\frac{\text{Annual_time}_{\text{phase i}}}{8760} \times \lambda_{\text{phase i}} \right) \tag{3}$$

$$\lambda_{\text{phase i}} = \left[\sum_{\text{contributions}} \left(\lambda_0 \times \Pi_{\text{acceleration}} \right) \right] \times \Pi_{\text{induced}} \tag{4}$$

L'équation (3) exprime l'impact du cycle de vie sur le taux de défaillance global. Le paramètre Annual_time correspond au temps passé (en heures) au cours d'une année dans une phase spécifique. Le paramètre $\lambda_{\text{phase i}}$ correspond quant à lui aux contributions physiques propres à la phase considérée. L'équation (4) exprime l'impact de différents stress sur le taux de défaillance intrinsèque de l'article : λ_0. Dans cette équation, $\Pi_{\text{acceleration}}$ traduit la sensibilité de l'article aux conditions d'utilisation du produit. Il représente l'impact de différentes familles de stress physique (thermique, électrique, mécanique, chimique, humidité, cycle de température). Le paramètre Π_{induced}

représente l'impact des surcharges accidentelles. Il est évalué selon la sensibilité naturelle aux surcharges de la technologie utilisée et la politique de durcissement adoptée lors du développement du produit, mais aussi de l'emplacement de l'article dans le produit et les conditions d'utilisation du produit.

Concernant λ_0 et $\Pi_{acceleration}$, le guide FIDES indique que :

« Ces facteurs sont déclinés pour chaque contrainte physique. Est appelée contrainte physique toute contrainte normalement appliquée au produit lors de son utilisation opérationnelle, y compris pour les aspects relevant de la conception. Les contraintes physiques sont regroupées en différentes familles :

- Thermique
- Électrique
- Cyclage thermique
- Mécanique
- Humidité
- Chimique ».

Ainsi, il existe en réalité un $\lambda_{0Thermique}$ et un $\Pi_{accelerationThermique}$, un $\lambda_{0Electrique}$ et un $\Pi_{accelerationElectrique}$, … Les λ_0 restent des paramètres intrinsèques du composant, mais les facteurs d'accélération sont quant à eux dépendant des conditions d'utilisation dans chaque phase de vie.

De même, l'évaluation du terme $\Pi_{induced}$ est dépendante de la phase de vie considérée. Pour chaque phase i, l'évaluation de $\Pi_{induced}$ est transcrite par cette équation :

$$\Pi_{induced\text{-}i} = \left(\Pi_{placement\text{-}i} \times \Pi_{application_i} \times \Pi_{ruggedising}\right)^{0.511 \times \ln(C_{sensitivity})} \quad (5)$$

- $C_{sensitivity}$ représente le coefficient de sensibilité aux surcharges de la technologie de l'article considéré. Ce paramètre dépend du type d'article étudié.
- $\Pi_{placement}$ exprime l'influence du placement de l'article dans le produit étudié sur la fiabilité. Ce paramètre est particulièrement influencé selon que l'article est en interface du produit (i.e. directement en contact avec l'environnement extérieur du produit).
- $\Pi_{application}$ exprime l'influence des conditions d'utilisation du produit. Ce paramètre est évalué par un questionnaire simple lié à une grille de cotation des conditions d'utilisation pour la phase considérée.
- $\Pi_{ruggedising}$ (durcissement) représente le facteur décrivant la politique de prise en compte des surcharges accidentelles lors du processus de développement du produit. Ce paramètre est évalué par un questionnaire simple lié à une grille de cotation de la politique de durcissement du produit.

Voyons maintenant les équations décrivant l'évaluation du paramètre Π_{PM} :

$$\Pi_{PM} = e^{(1-Part_Grade)} \tag{6}$$

$$\text{Part_Grade} = \left\lfloor \frac{\left(QA_{manufacturer} + QA_{item} + RA_{item}\right) \times \varepsilon}{36} \right\rfloor \tag{7}$$

- $QA_{manufacturer}$ correspond à l'assurance qualité associée au sous-traitant. Ce paramètre est une note entre 0 et 3 selon la certification du sous-traitant.
- QA_{item} correspond à l'assurance qualité de l'article. Ce paramètre est une note entre 0 et 3 selon la certification de l'article.
- RA_{item} correspond à l'assurance fiabilité de l'article. Le guide FIDES définit ce paramètre comme n'étant pertinent que pour certains types d'articles tels que les circuits imprimés, les DELs, les opto-coupleurs, ou les semi-conducteurs discrets. Ce paramètre est une note entre 0 et 3 selon les tests de fiabilité que l'article a passés.
- ε correspond au facteur d'expérience. Il représente le retour d'expérience de l'acheteur et utilisateur de l'article ainsi que sa confiance envers le sous-traitant. Ce paramètre est en général spécifique à un sous-traitant et s'applique donc à tous les composants qu'il fabrique, mais si cela est pertinent le paramètre peut être spécifique à un type de composant. ε est noté entre 1 et 4.

Remarque : Pour l'ensemble des critères intervenant dans l'évaluation de Π_{PM} , des notes élevées correspondent à de meilleures performances en terme de fiabilité.

Pour finir ce tour d'horizon du modèle mathématique proposé par FIDES, nous allons décrire l'équation d'évaluation du $\Pi_{process}$:

$$\Pi_{Process} = e^{(1-Process_Grade)} \tag{8}$$

Le paramètre : Process_Grade est la note obtenue lors de l'audit FIDES de l'entreprise fabriquant le produit étudié.

Dans le chapitre II de ce manuscrit (cf Chapitre II §V.2.b), nous avons détaillé un moyen de représenter les relations mathématiques et logiques en utilisant le langage SysML : l'utilisation de *constraint block* et *constraint property*, organisés dans un *Parametric Diagram* (ParD). La Figure IV.7 présente les équations, mises en œuvre par la méthodologie FIDES, modélisées en SysML en utilisant un ParD. Chaque équation est réifiée par une *constraint property* (en jaune) et chaque paramètre ne dépendant pas d'une autre équation est identifié par une *value property* (en rouge). Cette représentation permet de mettre en exergue les paramètres nécessaires à l'évaluation de la fiabilité d'un produit avec la méthodologie FIDES : les *value properties*. Notons que l'utilisation de stéréotype propre à la BCD dans la Figure IV.7 sera justifiée dans la suite de ce chapitre.

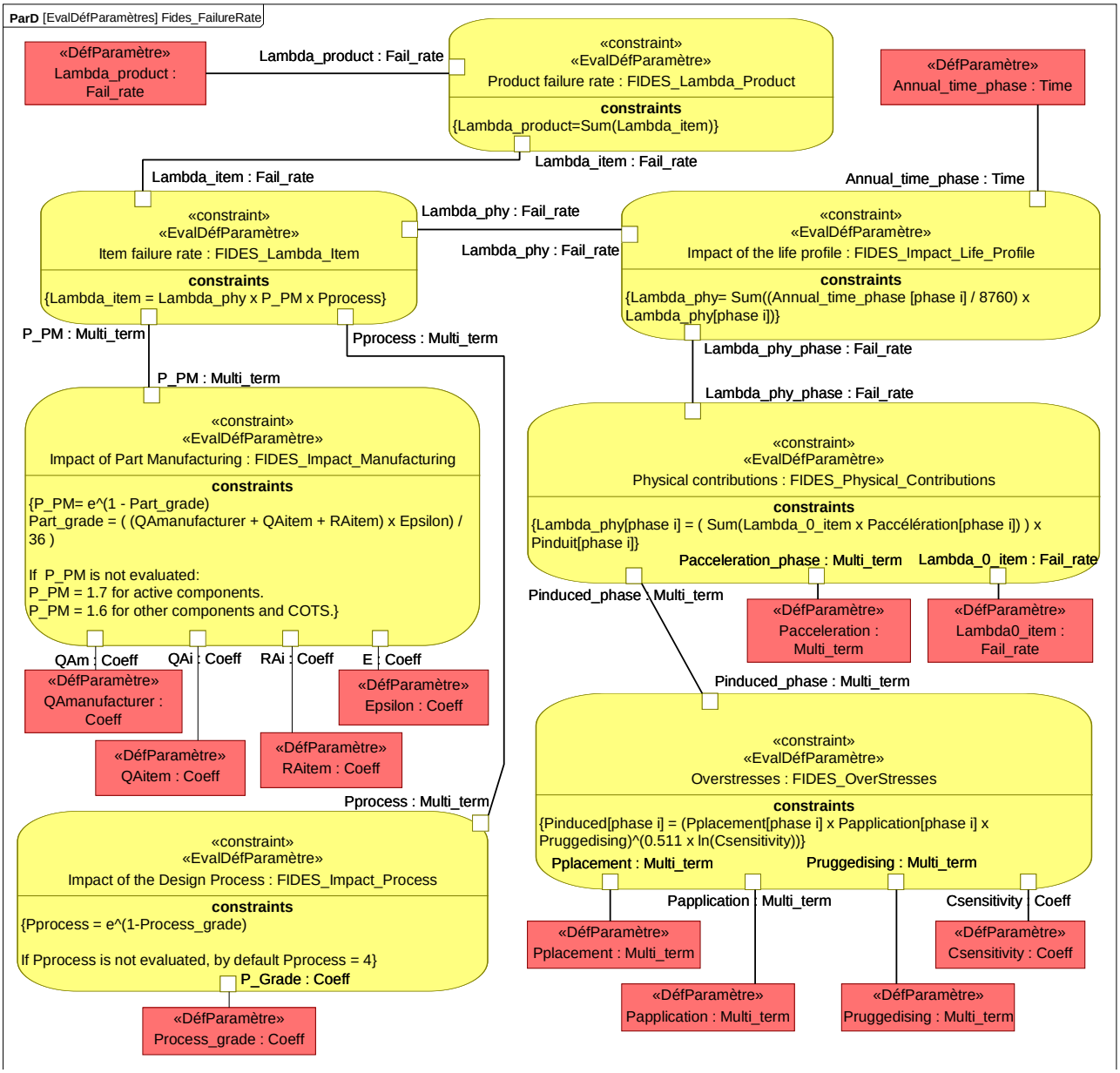


Figure IV.7 : Le modèle mathématique FIDES d'évaluation de fiabilité modélisé en SysML

Dans la suite de ce chapitre, nous allons étudier précisément comment la BCD peut profiter de la méthodologie FIDES et aussi comment FIDES peut profiter de l'approche ISBM qui accompagne le déploiement de la BCD.

IV. Intégration de l'évaluation de fiabilité FIDES à MéDISIS

1. Connexion de la méthodologie FIDES à la BCD

Au travers du §II de ce chapitre, la capacité de la BCD à pérenniser une large palette de données dysfonctionnelles a été mise en avant. La BCD, bien que conçue depuis [David 2009] pour être une base de données de MdD, répondait principalement à un besoin de permettre la représentation et le stockage des comportements dysfonctionnels propre aux MdD qu'elle contient, d'où son nom : Base de données des Comportements Dysfonctionnels. Ce besoin est né des études présentées dans

[David et al. 2009a] à propos de la rédaction de pré-AMDEC composant et de la génération de modèle Altarica DF à partir d'un modèle système en SysML. Cependant, les mécanismes de représentation proposés par SysML et réutilisés par la BCD sont tels que les comportements dysfonctionnels aussi bien que le déclenchement d'un MdD peuvent être décrits de façon détaillée. Diverses utilisations spécifiques de la BCD pour la modélisation des comportements dysfonctionnels ont été présentées dans différentes publications : [David 2009], [David et al. 2009a], [David et al. 2009b] et [David et al. 2010], mais les détails de l'utilisation de la BCD afin de modéliser des méthodes d'évaluation du déclenchement de mode de défaillances sont plus récents : [Cressent et al. 2012a] [Cressent et al. 2012b]. Différentes méthodes de prédiction de défaillance et d'évaluation de la fiabilité existent et pourraient être modélisées et connectées, voire intégrées à la BCD : statistiques de retour d'expérience, analyse de la physique des défaillances, … FIDES est à la croisée de plusieurs de ces méthodes et offre la possibilité d'évaluer la fiabilité des COTS, c'est pourquoi nous nous sommes intéressés à cette méthodologie.

En reprenant l'exemple de composant et de MdD présenté dans le §II de ce chapitre, nous voyons sur la Figure IV.4 que l'un des paramètres nécessaires à la modélisation du « Déclenchement » de notre mode de défaillance est le taux de défaillance du composant considéré, or c'est justement l'objectif principal de FIDES de pouvoir évaluer ce paramètre. Si le composant de notre exemple est un composant possédant un niveau de décomposition, alors il correspondra à un produit pour FIDES et son taux de défaillance pourra être évalué selon la méthodologie FIDES (équations (1) à (8)). Si le composant est lui-même un composant unitaire, alors il est considéré comme un article pour FIDES et l'évaluation de son taux de défaillance sera similaire (équations (2) à (8)).

Les entités de modélisation proposées par la BCD pour modéliser un raffinement de l'évaluation d'un paramètre du MdD sont les *constraint properties* « EvalDéfParamètre ». Ainsi pour connecter la méthodologie FIDES à la BCD nous pouvons transposer les *constraint properties* présentées dans la Figure IV.7 comme étant des « EvalDéfParamètre » dans la BCD pour l'ensemble des composants correspondant aux hypothèses d'utilisation de FIDES :

- Le mode de défaillance considéré pour la méthodologie FIDES suit une distribution exponentielle, comme c'est le cas de notre exemple du §II (cf Figure IV.4).
- Le composant étudié est un composant « Électrique, Électroniques et Électromécaniques » présent dans le guide FIDES ou bien il est composé de ce type de composants.

La logique d'évaluation de la méthodologie FIDES est à présent intégrée à notre BCD. Cependant pour être utile, l'ensemble des paramètres nécessaires à l'utilisation de la méthodologie FIDES doit être renseigné. Certains de ces paramètres sont propres au MdD considéré et certains propres au projet. Le Tableau IV.1 présente l'ensemble des concepts mis en avant dans la méthodologie FIDES (Figure IV.7) et leur correspondance dans la structure de la BCD.

Tableau IV.1 : Connexion des concepts de la méthodologie FIDES et de la BCD

Concepts de FIDES	Concepts de la BCD
Constraint properties	EvalDéfParamètres
Taux de défaillance produit **($\lambda_{product}$)**	DéfParamètre Paramètre du Déclenchement
Life Cycle **(Annual Time phase)**	DéfParamètre
Lambda_0_item **($\lambda_{0\ item}$)**	DéfParamètre

Pacceleration **($\Pi_{acceleration}$)**	DéfParamètre [1]
Csensitivity **($C_{sensitivity}$)**	DéfParamètre
Pplacement **($\Pi_{placement}$)**	DéfParamètre [1]
Pruggedising **($\Pi_{ruggedising}$)**	DéfParamètre [2]
Papplication **($\Pi_{application}$)**	DéfParamètre [1]
QAmanufacturer **($QA_{manufacturer}$)**	DéfParamètre [2]
QAitem **(QA_{item})**	DéfParamètre
RAitem **(RA_{item})**	DéfParamètre
Epsilon	DéfParamètre
Process_grade	DéfParamètre [2]

[1] : DéfParamètres pouvant être raffinés par d'autres constraint properties (« EvalDéfParamètres »). Certains des sous-paramètres peuvent alors être spécifiques au projet.
[2] : DéfParamètres commun a plusieurs composants et MdD, hérités du point de vue Entreprise.

Nous pouvons observer que la majorité des *value properties* identifiées dans la Figure IV.7 correspondent dans la BCD à des « DéfParamètres ». Pourtant, nous devons distinguer différents types de paramètres :

- Il y a des paramètres indépendants du projet et du contexte d'utilisation des composants. Parmi ces paramètres :
  - **Type n°1 :** Certains paramètres sont spécifiques à un composant et à l'évaluation de sa fiabilité. Les facteurs physiques de surcharges et l'assurance qualité d'un composant en sont des bons exemples. Ces paramètres peuvent être définis et valués dans la BCD selon le guide FIDES.
  - **Type n°2 :** Certains paramètres sont spécifiques à l'entreprise fabriquant le système ou les composants. La note : Process_Grade, qui permet de calculer le $\Pi_{Process}$ ou l'assurance qualité des sous-traitants en sont des exemples. Ces paramètres peuvent être définis et valués dans la BCD selon le guide FIDES, mais ils sont communs à plusieurs composants et MdD.
- Il y a des paramètres dépendant du projet et du contexte d'utilisation des composants. Parmi ces paramètres :
  - **Type n°3 :** Certains paramètres sont spécifiés lors des activités d'IS. C'est le cas des durées de phase, des environnements thermiques de chaque phase, … Ces paramètres pourront être définis dans la BCD mais leurs valeurs ne pourront être attribuées que lors de l'analyse du modèle système.
  - **Type n°4 :** Certains paramètres ne sont pas spécifiés lors des activités d'IS, par exemple : $\Pi_{placement}$ et $\Pi_{application}$. Ce sont des paramètres qui n'ont de sens que pour l'étude du MdD. Ils seront donc définis dans la BCD mais leurs valeurs ne pourront leur être attribuées que par l'analyse d'un expert de SdF. Ces valeurs pourront cependant être réutilisées d'une itération sur l'autre comme nous l'avons défini avec certains paramètres de la pré-AMDEC fonctionnelle dans le chapitre III.

L'ensemble de ces paramètres peut donc être défini dans la BCD. Lors de l'utilisation de processus MéDISIS nécessitant d'avoir accès aux données de déclenchement de défaillance (par exemple le processus de génération de modèle Altarica DF), l'expert SdF devra évaluer les paramètres qui ne le

sont pas dans la BCD, et le lien avec le modèle système en SysML permettra d'évaluer les données environnementales nécessaires à la méthodologie FIDES.

2. Valorisation du modèle système pour l'étude FIDES

L'intégration de la méthodologie FIDES dans la BCD présente quelques limites, comme nous venons de le voir. En effet, certains paramètres indispensables à l'évaluation de la fiabilité d'un produit ou d'un article avec FIDES ne peuvent pas être simplement stockés dans la BCD à cause de leur dépendance vis-à-vis du projet et de l'environnement du produit (Type n°3 et 4). Concernant les paramètres de type n°3, il est alors possible de décrire les algorithmes permettant d'extraire ces informations du modèle système comme nous l'avons fait pour la génération de pré-AMDEC fonctionnelle.

Les paramètres de type n°3 sont par définition des paramètres du modèle système dépendant d'un projet spécifique et nécessaire à la méthodologie FIDES. Dans la Figure IV.7, le seul paramètre correspondant à cette définition est le temps passé au cours d'une année dans une phase spécifique (Annual Time phase). Cependant dans le Tableau IV.1, nous indiquions que certains paramètres peuvent eux-mêmes être raffinés par des modèles mathématiques, c'est notamment le cas de $\Pi_{acceleration}$ qui est décliné en plusieurs facteurs d'accélération (pour chaque famille de contribution physique). Par exemple, le facteur d'accélération thermique pour une phase spécifique est défini par l'équation suivante :

$$\Pi_{\text{Thermique}} = e^{\frac{E_a}{K_B}\left(\frac{1}{273+T_0}-\frac{1}{273+T_{ambient}}\right)} \quad (8)$$

Avec :

- E_a : énergie d'activation considérée constante (en eV/mol)
- K_B : constante de Boltzmann (8,617.10-5 eV/K).
- T_0 : températures de référence (20 °C).
- $T_{ambient}$: température ambiante (en °C).

La température ambiante est donc un paramètre de la description du cycle de vie de notre système indispensable à l'évaluation de la fiabilité de celui-ci par la méthodologie FIDES, c'est-à-dire un paramètre de type n°3. De plus, le guide FIDES indique aussi que pour les phases de vie où le système est éteint (« Non-powered phases ») la contrainte thermique est la plupart du temps annulée. Ainsi l'indication, la description de l'état d'alimentation dans chaque phase (« Powered » ou « Non-powered ») est aussi un paramètre de type n°3.

Sans détailler l'ensemble des équations correspondantes aux différentes familles de contributions physiques, nous retirons de celles-ci un ensemble non-exhaustif mais représentatif de paramètres de type n°3 :

- Powered : l'alimentation du système (Vrai ou Faux)
- $T_{ambient}$: la température ambiante du milieu (en °C)
- RH : l'humidité relative du milieu (pourcentage d'humidité)
- G_{RMS} : vibrations du milieu (en G^2_{RMS}/Hz)

À cela, nous pouvons ajouter le temps passé au cours d'une année dans une phase spécifique, et nous retrouvons finalement que les paramètres de type n°3 correspondent à la description détaillée des conditions d'utilisation dans chaque phase et de la durée de la phase. Or la description détaillée du cycle de vie et de l'environnement d'utilisation du système sont des activités réalisées lors de l'IS et notamment lors de l'élicitation des besoins (cf. Chapitre II §II).

En SysML, nous avons vu qu'un diagramme de contexte pouvait être réalisé pour spécifier exactement les phases de vie et les contraintes associées au système. Ainsi lors des activités d'IS, si la méthodologie FIDES doit être utilisée par la suite, un diagramme de contexte tel que celui présenté en Figure IV.8 peut alors fournir l'ensemble des paramètres de type n°3.

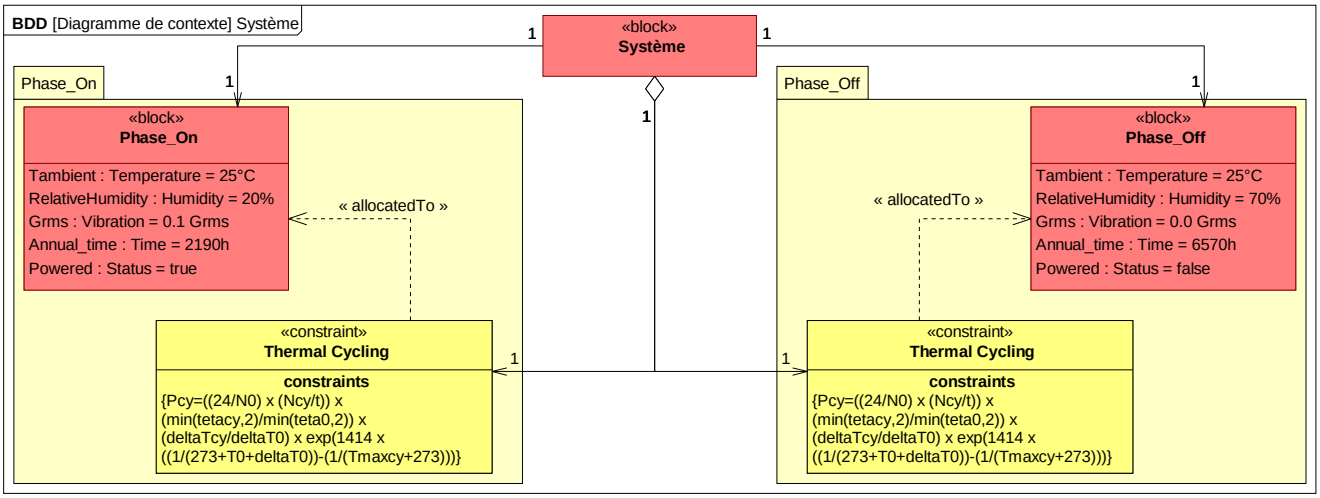


Figure IV.8 : Exemple de diagramme de contexte

Afin d'illustrer l'intérêt de l'interaction entre FIDES, la BCD et le modèle système pour l'analyse de fiabilité d'un produit électronique nous allons maintenant détailler les successions d'actions supportées par MéDISIS et l'ISBM qui mènent à l'évaluation de la fiabilité de ce composant avec la méthode FIDES.

3. Utilisation de la BCD et FIDES pour évaluer la fiabilité d'un composant

Nos hypothèses de travail pour cette partie sont :

- La méthodologie FIDES et ses modèles mathématiques ont effectivement été intégrés dans la BCD
- Le composant étudié est un composant qui entre dans la définition du produit au sens du guide FIDES (il est composé d'articles présents dans le guide FIDES). Le schéma électrique de notre composant d'exemple est présenté sur la Figure IV.9
- Les activités de description organique du composant ont été réalisées et leurs résultats réifiés en SysML.

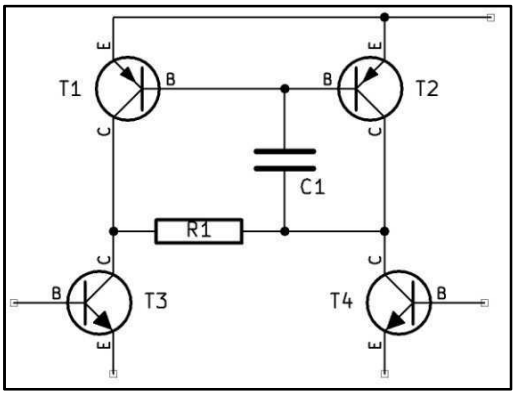


Figure IV.9 : Schéma électrique du produit d'exemple

Les diagrammes SysML de description organique indispensable à l'utilisation de la méthode FIDES sont le BDD et l'IBD. Comme vu précédemment, pour une étude rapide, l'IBD est facultatif, mais les multiplicités du BDD doivent alors être fixées (au lieu d'utiliser des plages de valeur). Le BDD représente ainsi une liste des composants du produit étudié et le $\Pi_{placement}$ sera évalué de façon globale au produit (résultant cependant en une fiabilité moins précise). La Figure IV.10 présente le BDD et l'IBD de notre produit électronique d'exemple. Il se compose de 4 transistors, 1 résistance et 1 capacité.

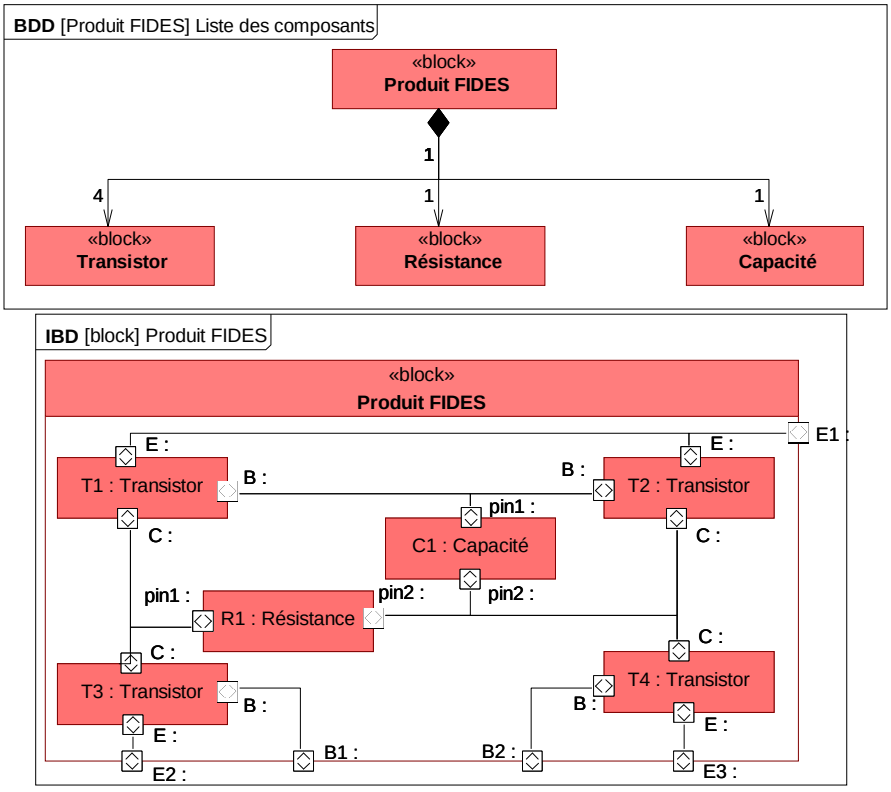


Figure IV.10 BDD et IBD du produit d'exemple

Ce BDD définit les types de composants utilisés et permet donc de parcourir la BCD à la recherche d'un block « TypeDeComposant » du même nom. Dans la Figure IV.11 , un extrait de la BCD pour une résistance est représenté. Seuls les « DéfParamètre » utiles à FIDES sont affichés par souci de lisibilité. On y retrouve l'ensemble des λ_0 pour chaque contribution physique utile (Thermique,

Humidité et Mécanique) et par type de phase (« Powered » et « Non-Powered »), la sensibilité aux surcharges accidentelles et les facteurs permettant de calculer Π_{PM} . Notons que RA_{item} est à « NA » (Non Applicable), car la résistance n'est pas un composant actif[1].

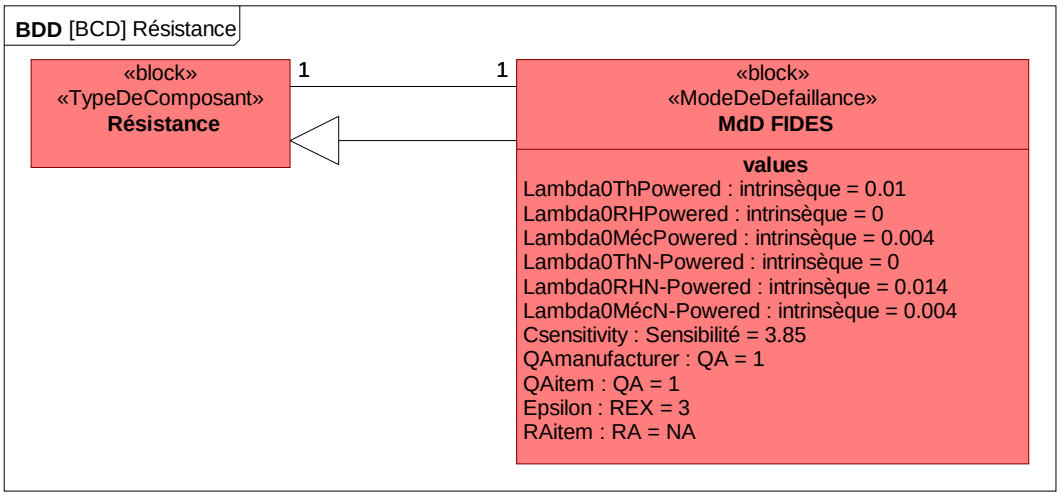


Figure IV.11 : Extrait de la BCD: Résistance

Tous ces paramètres sont récupérés dans la BCD pour chaque type de composant présent dans le produit. Le Tableau IV.2 récapitule l'ensemble de ces données.

Tableau IV.2 : Paramètres d'évaluation de la fiabilité extraits de la BCD

		transistor	capacitor	resistor
Powered	$\lambda_{0\ Th}$	0,014	0,048	0,01
	$\lambda_{0\ RH}$	0	0	0
	$\lambda_{0\ M}$	0,00011	0,0014	0,004
Non-powered	$\lambda_{0\ Th}$	0	0	0
	$\lambda_{0\ RH}$	0,031	0	0,014
	$\lambda_{0\ M}$	0,00011	0,0014	0,004
	$\mathbf{C_{sensitivity}}$	5,2	6,05	3,85
	$\mathbf{QA_{manufacturer}}$	1	1	1
	$\mathbf{QA_{item}}$	1	1	1
	$\mathbf{RA_{item}}$	1	NA	NA
	$\mathcal{E}$	3	3	3

Si le produit est intégré au système dont le cycle de vie est présenté dans le diagramme de contexte de la Figure IV.8, alors les paramètres de type n°3 sont renseignés par analyse du modèle SysML. Le Tableau IV.3 résume ces données.

Tableau IV.3 : Description du cycle de vie, déduit du modèle système

Nom phase	Powered	$t_{annual\text{-}phase}$	$T_{ambient}$	RH	G_{RMS}
Phase_Off	False	6 570 h	25 °C	70 %	0,0 Grms
Phase_ On	True	2 190 h	25 °C	20 %	0,1 Grms

[1] Les composants considérés actifs dans FIDES sont les circuits intégrés, les transistors et les diodes

Le paramètre $\Pi_{process}$ est choisi à sa valeur par défaut (= 4) si l'entreprise développant le système n'a pas réalisé d'audit FIDES, sinon $\Pi_{process}$ est calculé grâce aux résultats de l'audit grâce à l'équation (8). De même, $\Pi_{ruggedising}$ est mis à sa valeur par défaut (= 7) si aucun test de durcissement n'a été réalisé. Dans le cas contraire, l'expert SdF, en possession de la description des tests et de leurs résultats pourra établir la valeur de $\Pi_{ruggedising}$ selon le questionnaire présent dans le guide FIDES [FIDES 2011]. Les derniers paramètres $\Pi_{application}$ et $\Pi_{placement}$ sont des paramètres qui sont évalués par l'expert SdF (l'un selon un questionnaire, l'autre selon une échelle simple de cotation). Ils peuvent être évalués par l'expert au niveau de l'utilisation du produit étudié (ici le produit), mais pour une étude plus précise, le guide FIDES recommande d'évaluer ces deux paramètres article par article. La cotation du $\Pi_{application}$ ne peut pas être extrapolée du modèle système, car il s'agit en réalité de la cotation de certains aspects de la phase de vie considérée. Cependant la description des phases de vie telle qu'elle est réalisée en SysML permettra de fournir à l'expert devant juger les critères de cotation du $\Pi_{application}$, un ensemble de données structurées pouvant l'aider dans son jugement.

La cotation du $\Pi_{placement}$ quant à elle se résume à répondre deux questions à propos de l'article étudié :

- Numérique, analogique bas niveau ou analogique puissance ?
- Interface ou non interface ?

Concernant la nature numérique ou analogique du composant, le modèle ne peut a priori rien apporter de plus à l'expert que la description du composant et de son type afin qu'il juge par lui-même. Nous pouvons cependant observer que le typage des flux entrant et sortant du composant se révélera particulièrement utile pour répondre à cette question. Concernant la question de savoir si l'article étudié est un composant d'interface ou non, le modèle système peut cette fois être utile. En effet, à partir de l'IBD du produit (Figure IV.10), nous pouvons exécuter un algorithme simple sur les *flowports* d'entrée et de sortie du composant. Si au moins un des *flowports* du composant est relié par un flux à un *flowport* de bord, alors le composant étudié est un composant d'interface.

Finalement, dans cet exemple simple, avec les données indiquées dans ce paragraphe, le taux de défaillance de notre produit est de 8.9 FIT[2]. Outre le résultat numérique, il faut noter que la majeure partie des paramètres permettant l'évaluation de fiabilité est issue du modèle système ou de la BCD. Ainsi, l'expert de SdF peut se concentrer sur les paramètres où son jugement d'expert est indispensable.

V. Conclusion

Dans ce chapitre, deux concepts de structuration de retour d'expérience ont été développés : la Base de données des Comportements Dysfonctionnels de MéDISIS et la méthodologie FIDES d'évaluation de la fiabilité des produits électroniques. Au cours de la description des principes de chacun de ces concepts, nous avons mis en évidence les connexions possibles entre eux. Nous avons ensuite décrit comment cette connexion peut être utilisée afin de définir un nouvel axe de valorisation de l'ISBM pour les études de sûreté de fonctionnement.

[2] FIT: Failure In Time, 1 FIT = 1 défaillance par 10^9 heures

La BCD a été créée pour répondre aux besoins de la méthodologie MéDISIS : permettre les interactions entre le modèle système en SysML et les modèles de SdF. La BCD apporte d'autres possibilités, notamment la pérennisation des données dysfonctionnelles. Cela permet ainsi d'optimiser les processus MéDISIS au fur et à mesure des itérations et des projets successifs. Afin d'optimiser l'intégration de ces données dysfonctionnelles aux processus de traduction, le méta-modèle de la BCD s'appuie sur les concepts de SysML. La structure de la BCD permet de modéliser trois aspects des modes de défaillances des composants pour lesquels elle contient des données :

- Les mécanismes de déclenchement de la défaillance (aléatoire ou déterministe)
- Les effets de la défaillance sur le fonctionnement normal du composant
- La logique régissant les comportements opérationnels et défaillants en fonction des paramètres du composant et de son mode de défaillance.

En détaillant les principes mathématiques du guide FIDES, il apparait que l'évaluation de fiabilité proposée par FIDES peut s'intégrer à la BCD à travers les mécanismes décrivant le déclenchement de la défaillance. Nous avons donc présenté les moyens offerts par le langage SysML permettant l'intégration des équations de FIDES, aux composants électroniques présents dans la BCD. Lors de la définition des paramètres mathématiques utilisés par FIDES, il a été mis en avant la prise en compte systématique du contexte d'utilisation des composants. Cette dépendance vis-à-vis du cycle de vie du composant et *a fortiori* vis-à-vis du modèle système nous a amené à décrire l'utilisation de la BCD et de FIDES pour l'étude d'un composant appartenant à un système qui aurait profité d'une approche ISBM telle que nous la conseillons dans la thèse. Cet exemple a alors illustré les bénéfices apportés (rapidité et cohérence) aussi bien par la BCD que par l'ISBM pour l'expert SdF devant réaliser une étude de fiabilité avec FIDES.

Avec ce chapitre, nous ajoutons à la méthodologie MéDISIS un autre atout pour son utilisation dans un contexte industriel : la méthodologie FIDES. Afin d'effectivement montrer l'applicabilité des travaux théoriques présentés dans ces deux derniers chapitres, le chapitre V va s'efforcer de présenter l'ensemble des travaux industriels réalisés dans le cadre du projet LEA et l'impact des processus de MéDISIS et de la BCD pour ces travaux.

Chapitre V. Application au projet LEA

I. Introduction

Au cours des chapitres précédents, nous avons présenté et décrit l'approche ISBM, son apport pour les spécifications d'un système complexe, la modélisation des résultats d'activité d'IS avec le langage SysML. L'intérêt de la méthodologie MéDISIS a été montré ainsi que les moyens de valoriser un modèle système pour des études de SdF, notamment pour la réalisation d'AMDEC fonctionnelle et l'évaluation de la fiabilité des composants. Ces résultats académiques illustrés à travers des exemples pédagogiques simple ont été présentés dans diverses conférences [Cressent et al. 2010][Cressent et al. 2011a][Cressent et al. 2011c][Cressent et al. 2012a] et revues [Cressent et al. 2011b] [Cressent et al. 2012b] souvent instanciés par le projet industriel support de nos travaux : le projet LEA.

Après un bref rappel sur le projet LEA et ses objectifs, les activités industrielles spécifiques menées par notre équipe sur le véhicule LEA et son système embarqué seront présentées et organisées. La planification de ces activités met d'ailleurs en avant, une fois de plus, l'importance du modèle système pour un grand nombre de tâches ou travaux d'autres domaines que ceux de l'IS. Parmi ces tâches, certaines profitent des principes décrits dans les chapitres II à IV. L'application de ces principes sera alors détaillée, aussi bien lorsque celle-ci est directe, que lorsque la dynamique du projet impacte quelque peu la théorie. Enfin, nous présenterons des travaux initiés pour répondre à des besoins du projet LEA et qui ne s'inscrivent pas directement dans le sujet de la thèse, mais qui profite de la réflexion méthodologique déployée jusqu'à maintenant : notamment le processus de génération de modèle AADL pour l'analyse d'ordonnancement et le processus de génération de modèle Simulink pour la simulation et l'injection de faute.

Finalement nous décrivons succinctement la méthode d'ingénierie utilisée pour le développement et la validation du logiciel de vol de LEA. Cette méthode s'inscrit dans la continuité de l'approche ISBM et permet d'optimiser le temps de développement d'une application et sa validation en limitant les risques d'erreurs humaines grâce à la génération de code automatique et l'utilisation d'un unique modèle de test tout au long des travaux de conception et validation.

II. Projet LEA

1. Présentation

Le projet LEA, initié par MBDA et l'ONERA, a pour objectif de définir un véhicule expérimental à statomixte dont le prototype sera testé en conditions réelles de vol. Le moteur à statomixte est un concept de statoréacteur fonctionnant d'abord en combustion subsonique, puis en combustion supersonique et hypersonique, pour pouvoir évoluer dans un domaine de Mach très large (de 2 à 8).

Le véhicule LEA évoluera, lors des essais en vol, à des vitesses comprises entre Mach 4 et Mach 8, dans la stratosphère (de 12 à 50 km d'altitude). Plusieurs essais sont prévus à différentes vitesses afin de multiplier les données sur les performances du moteur. Les essais auront lieu sur le territoire russe en collaboration avec l'entreprise RADUGA qui assurera la mise en condition de tests du véhicule LEA. Cette mise en condition de tests est séquencée en deux étapes :

- Un avion emporte le véhicule LEA et le largue à une vitesse de Mach 1.8.

- Le véhicule LEA est ensuite propulsé par un booster à propulsion thermique jusqu'à l'altitude et la vitesse de l'essai. Lorsque les conditions de tests sont atteintes, le booster se sépare du véhicule LEA pour lui permettre de réaliser son vol autonome.

Au terme de l'essai du vol autonome, le véhicule s'écrase et sera vraisemblablement détruit. Ainsi pour permettre la récupération des données de test, le véhicule LEA transmettra l'ensemble des données capteurs au sol pendant le vol. Cette chronologie d'essai est résumée dans la Figure V.1.

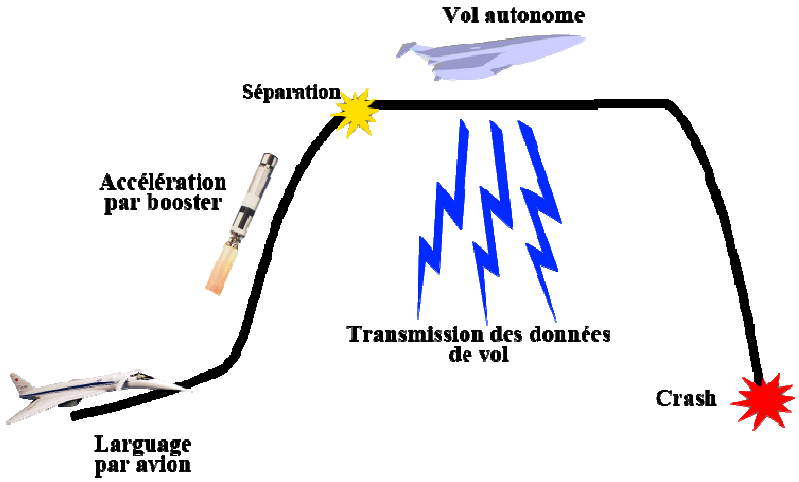


Figure V.1 : Chronologie de l'essai du véhicule LEA

À l'issue d'études préliminaires, la première définition des fonctions principales du système embarqué de LEA a été réalisée par MBDA et le découpage suivant en sous-système a été proposé :

- Pour le pilotage de la séquence d'essai et des équipements : **un calculateur de vol et un logiciel de vol**
- Pour l'acquisition des données : **un ensemble d'instrumentation**
- Pour le conditionnement, le multiplexage et le formatage des données mesurées : **un codeur**
- Pour la transmission des données : **un ensemble télémesure (émetteur, antenne, …)**
- Pour l'alimentation en électricité : **une ou des alimentations électriques**
- Pour l'alimentation en carburant : **un sous-ensemble d'alimentation gaz**

Remarque : le moteur et le fuselage du véhicule ne font pas partie du système embarqué.

Afin d'illustrer l'aspect multipartenaire du projet LEA, notons que le codeur est un équipement fourni et paramétré par la société ZODIAC DATASYSTEMS, l'ensemble de télémesure est fourni par la société RADUGA, le moteur et le fuselage de LEA sont mis au point par MBDA et l'ONERA et enfin, notre équipe développe principalement le logiciel de vol et met en place les moyens de validation sur le calculateur de vol.

Ainsi, afin de réduire les coûts de développement d'un tel système, il est nécessaire de planifier l'ensemble des activités des différents domaines d'expertise dès les premières phases de spécification et anticiper les éventuels points durs.

2. Planification

Dans les précédents chapitres de ce manuscrit, nous avons décrit le cadre méthodologique MéDISIS ainsi que l'intérêt du modèle système en SysML indispensable à l'application des raisonnements de MéDISIS. Au cours de la thèse, le cadre MéDISIS original proposé par [David 2009] a été modifié et étendu afin de valoriser l'ensemble des activités d'IS (de l'élicitation des besoins à la conception détaillée des composants et de leurs comportements) pour la réalisation d'études de sûreté de fonctionnement. Le processus d'IS considéré, présenté sur la Figure I.2, définis 5 types de données résultant de l'IS : la « Spécification des besoins », l'« Analyse fonctionnelle », la « Spécification des Exigences et Contraintes », le « Dossier de conception du système » et le « Dossier de conception détaillée du système ». Les moyens de réifier ces résultats avec le langage SysML ont été présentés dans le chapitre II, et deux méthodes de valorisation de ce modèle système pour les études de SdF de systèmes intégrant des COTS ont été présentées aux chapitres III et IV.

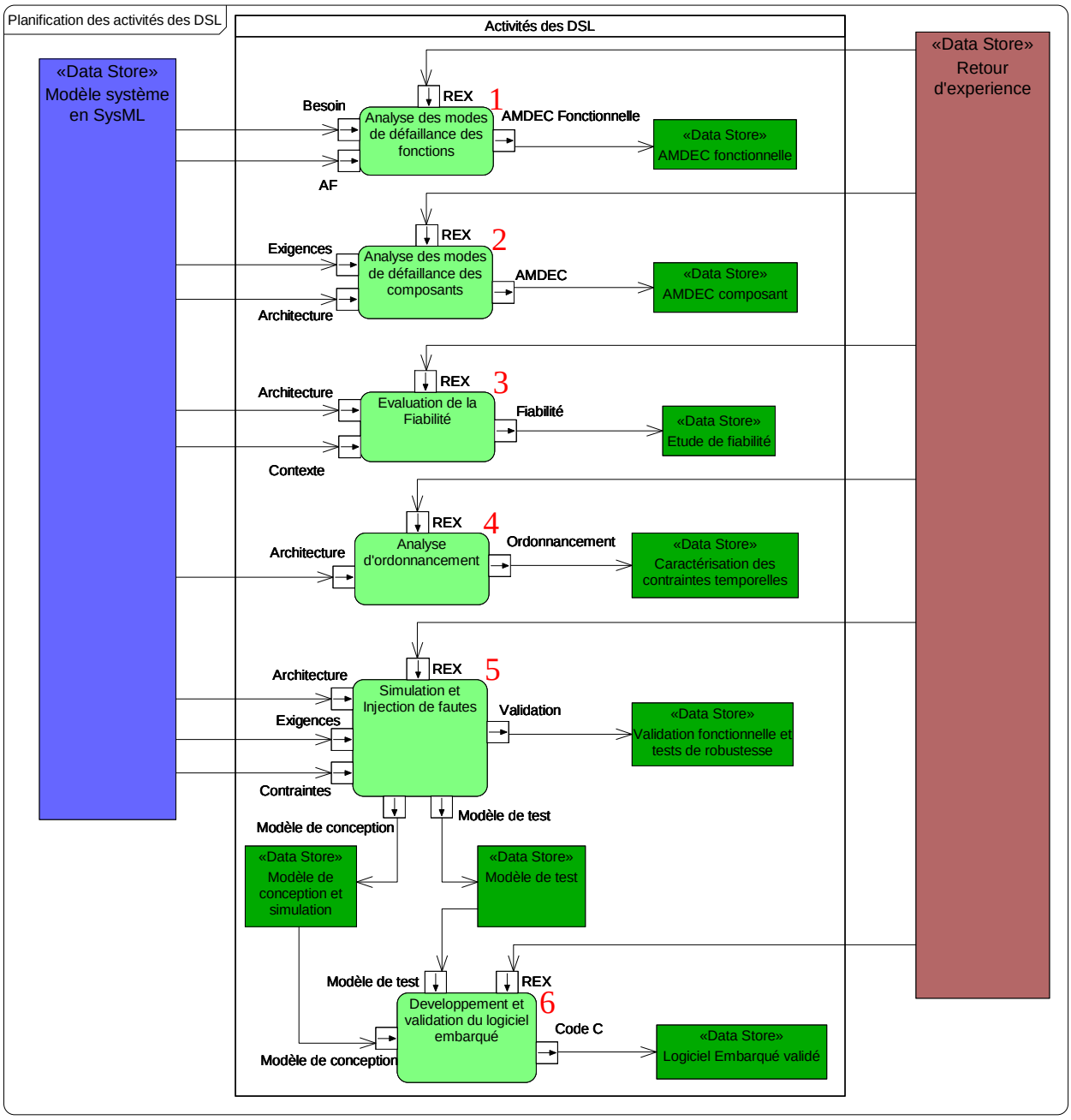


Figure V.2 : Planification des activités des DSL pour le projet LEA

Comme nous l'avions vu dans le chapitre I, les études de sûreté de fonctionnement ne sont pas les seules activités spécifiques pouvant profiter de l'approche ISBM et du modèle système. Ce principe était présenté dans la Figure I.3 où le modèle système est le point de rencontre entre les connaissances, les études utilisant des DSL et la production documentaire. Ainsi dans un projet tel que LEA, même si les études de SdF sont une considération majeure, d'autres activités propres à de multiples DSL pourront s'interfacer avec le modèle système. La Figure V.2 présente la planification des activités spécifiques dans le cadre du projet LEA et les liens de celles-ci avec les résultats de l'IS.

En analysant la Figure V.2, il apparait que la méthodologie MéDISIS peut être utilisée pour faciliter certaines activités (notamment les activités 1 à 3). Les processus de traduction de MéDISIS associés aux activités 1 et 3 ont été décrits précédemment. Le processus associé à l'activité 2 a été décrit dans [David 2009].

Le système embarqué du véhicule LEA comporte des aspects logiciels importants ainsi que des contraintes de temps d'exécution (notamment pour garantir la qualité des données télémesurées). Dans ces conditions, une analyse d'ordonnancement des traitements matériels et tâches logicielles est recommandée (activité 4). Enfin, pour garantir les performances et le respect des exigences fonctionnelles du système, la simulation du logiciel de vol a été décidée (activité 5). Cette simulation est réalisée à l'aide d'un modèle du logiciel dans son contexte en Simulink, permettant aussi de réaliser des tests de robustesse de type injections de fautes. À terme ce modèle Simulink sert aussi à générer le code C du logiciel de vol en utilisant des outils MathWorks (activité 6).

Avant de commencer les activités avec des DSL, les activités d'IS ont été réalisées et leurs résultats modélisés en SysML selon la sémantique présentée dans le chapitre II. Dans la partie suivante, nous allons présenter quelques résultats de ces activités ainsi que l'utilisation du processus MéDISIS de rédaction de pré-AMDEC qui a été appliqué afin de fournir une base de travail au partenaire du projet.

III. Génération de la pré-AMDEC fonctionnelle de LEA

1. Elicitation des besoins

La phase d'élicitation des besoins du projet LEA commence avec une définition du contexte de vie de notre véhicule. Le résultat de cette étape est réifié par un diagramme de contexte (Figure V.3). Suivant nos règles de modélisation SysML, les phases de vie sont associées au système (ici « LEA »). Elles sont raffinées selon les besoins en utilisant un lien de composition. Toutes ces phases de vie héritent d'un *block* générique « Phase de vie ». Les liens de *generalization* correspondants à ces héritages ne sont pas représentés sur la Figure V.3 mais ont été exprimés par une couleur verte pour les *blocks* représentant des phases de vie. Les acteurs amenés à interagir avec le système sont eux aussi représentés et associés au *block* « LEA ».

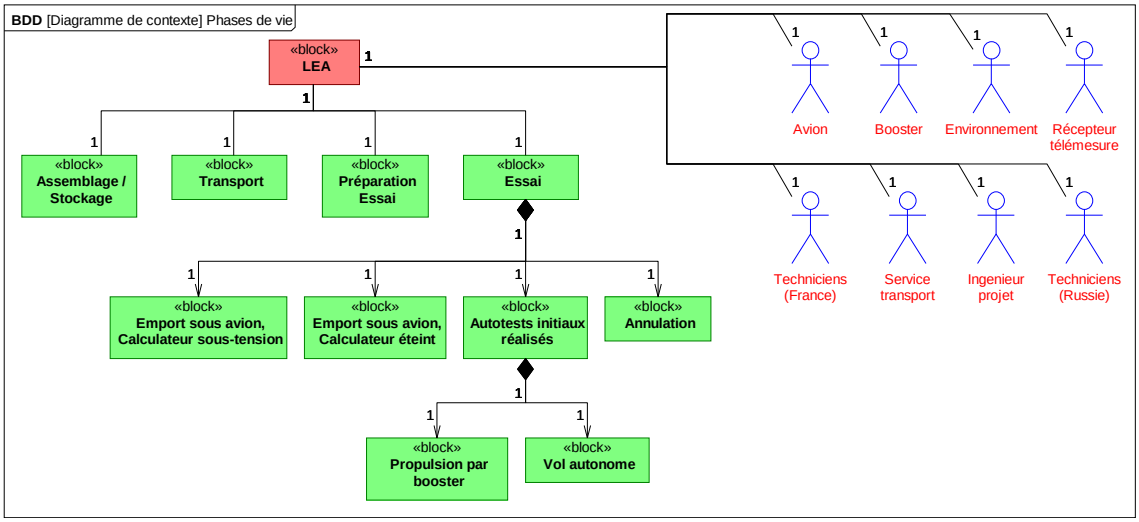


Figure V.3 : Diagramme de contexte du véhicule LEA

Lorsque la définition des phases de vie est terminée, l'élicitation des besoins fonctionnels peut être réalisée. Elle résulte en un UCD. La Figure V.4 illustre le résultat de cette étape pour la phase de vie « Essai » et ses sous-phases : « Emport sous avion, calculateur éteint », « Emport sous avion, calculateur sous-tension », « Annulation » et « Autotests initiaux réalisés » ainsi que les sous phases de cette dernière : « Propulsion par booster » et « Vol autonome ». Le cadre nommé LEA représente le domaine du système (*block* « LEA ») et les cadres en pointillé représentent les phases de vie décrites dans le diagramme de contexte ci-dessus. Les acteurs impliqués dans le cycle de vie de LEA sont aussi représentés ainsi que leurs liens avec certains besoins fonctionnels spécifiques du système (par une relation d'interaction simple). Nous pouvons observer que certains besoins sont propres à une phase de vie, par exemple : « Préparer le vol autonome » et d'autres sont communs à plusieurs sous-phases, par exemple : « Chauffer la batterie ».

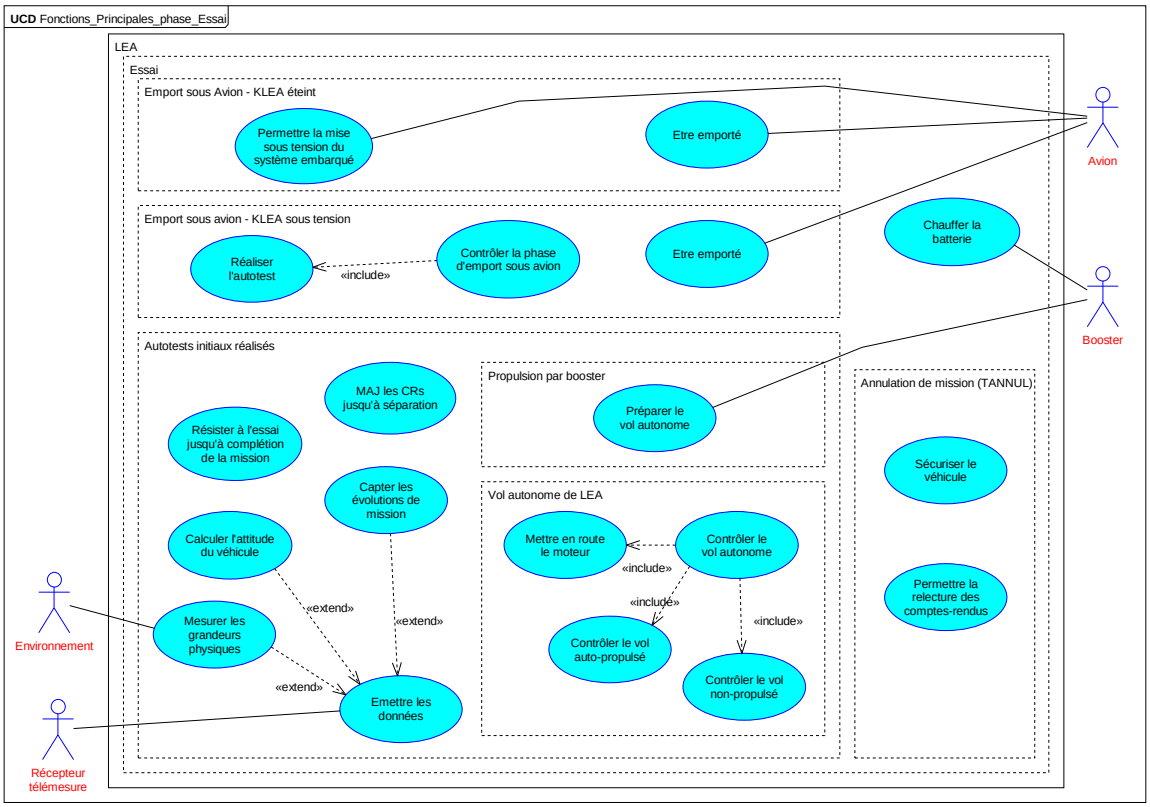


Figure V.4 : UCD des besoins du véhicule LEA, phase d'essai en vol

Lorsque les besoins sont spécifiés, il est alors possible de réaliser l'analyse fonctionnelle qui permet de détailler ces besoins de haut niveau en fonctions du système. Nous allons illustrer cette activité d'analyse fonctionnelle en présentant les AD réifiant le raffinement du besoin : « Contrôler le vol autonome ».

2. Analyse fonctionnelle

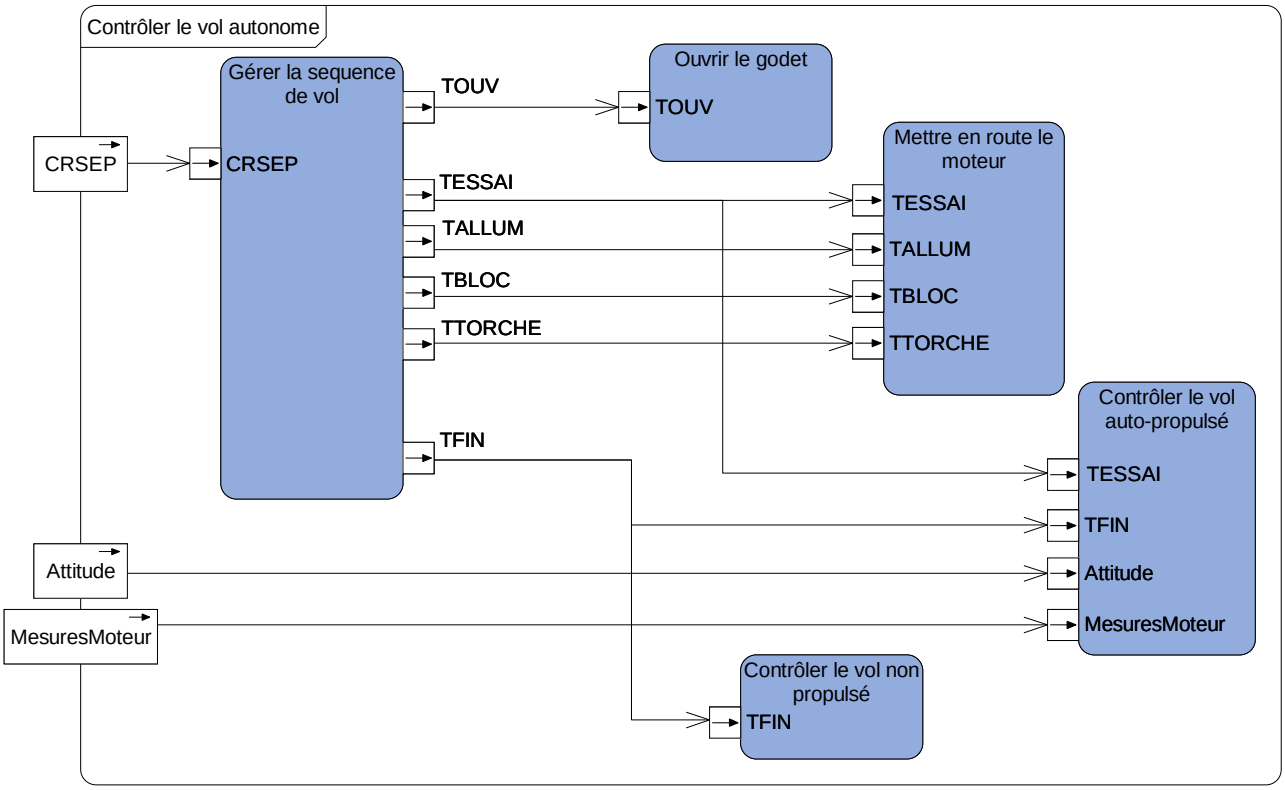


Figure V.5 : AD de la fonction : « Contrôler le vol autonome »

Le besoin : « Contrôler le vol autonome » est d'abord raffiné grâce à l'AD de la Figure V.5. Sur cet AD, il y a trois *pins* de bordure correspondant à des flux fonctionnels provenant d'autres besoins fonctionnels (*use case*). « CRSEP » provient de « Capter les évolutions de missions », « Attitude » provient de « Calculer l'attitude du véhicule » et « MesuresMoteur » provient de « Mesurer les grandeurs physiques ». La première fonction : « Gérer la séquence de vol » va engendrer plusieurs flux fonctionnels (TOUV, TESSAI, …) qui sont en entrée des autres fonctions : « Ouvrir le godet », « Mettre en route le moteur », « Contrôler le vol auto-propulsé » et « Contrôler le vol non propulsé ». Ces fonctions sont éventuellement elles-mêmes raffinées au fur et à mesure que l'analyse fonctionnelle du système se précise. Nous pouvons prendre l'exemple du raffinement de la fonction de mise en route du moteur. Au niveau SysML, l'*opaqueAction* « Mettre en route le moteur » est relié par un lien *refine* à l'AD présenté Figure V.6.

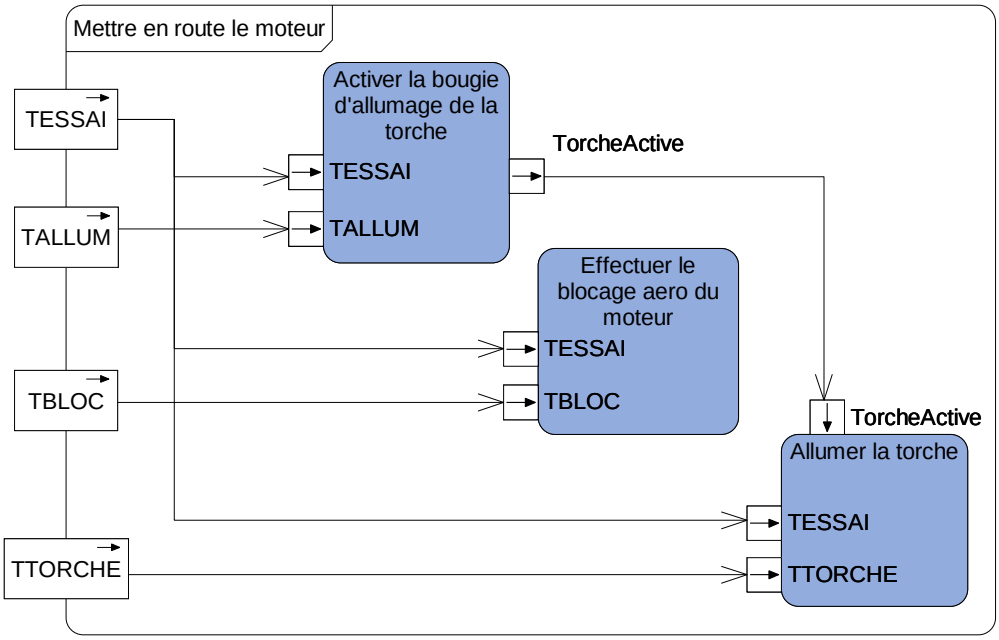


Figure V.6 : AD de la fonction : « Mettre en route le moteur »

En SysML, il est possible d'annoter un élément avec des informations utiles. Cette note est un élément SysML appelé *rationale* (il existe un élément SysML similaire appelé *problem* permettant d'indiquer des informations renseignant un éventuel problème du modèle ou du système modélisé : non-cohérence, non-complétude,…). Sur la Figure V.7, un *rationale* est spécifiquement représenté pour la fonction « Allumer la Torche ». Nous pouvons observer qu'il est lié à l'élément SysML par une relation propre à l'utilisation de *rationale* et *problem*. Les informations référencées dans ce *rationale* correspondent à l'identifiant de la fonction et aux critères de description d'une fonction lors d'une étude traditionnelle d'analyse fonctionnelle destinée à être présentée dans un document contractuel (e.g. livrable). Le critère « Facteur de réalisation » indique les documents de réalisation matérielle de la fonction. Le critère « Nombre de réalisation » précise le nombre de fois que la fonction sera réalisée lors de la vie du système. De même, les autres critères renseignent différents points tels que la durée d'exécution de la fonction, les moyens de réalisation de la fonction (composants supports), … À chaque critère est ensuite associée une classe de flexibilité de F0 à F2 (F0 étant le niveau le plus rigide) pour indiquer la flexibilité de modification du critère, lors des évolutions futures du système.

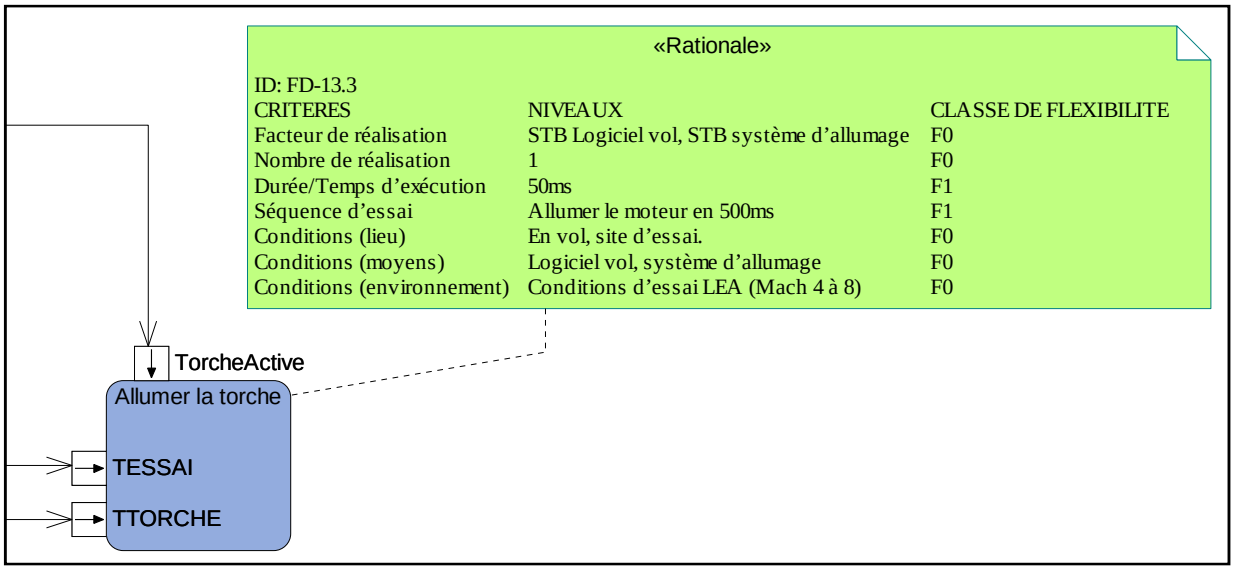


Figure V.7 : Exemple d'utilisation de l'élément SysML : *rationale*

Cette possibilité de stocker des informations pour chaque élément SysML est particulièrement utile afin de permettre la génération de livrables. En effet, l'approche ISBM que nous utilisons et que nous recommandons n'est actuellement pas totalement reconnue ou appliquée dans un processus industriel traditionnel. Le passage de l'ingénierie système à base de document à l'ingénierie système à base de modèle nécessite une période hybride où les deux approches doivent cohabiter. C'est pourquoi le *rationale* est utile, puisqu'il permet d'identifier lors de la rédaction automatique de dossier de spécification les informations spécifiques à chaque élément du modèle. Plusieurs outils logiciels de modélisation SysML proposent déjà des moyens de génération automatique de documents plus ou moins paramétrables qui permettent d'intégrer SysML plus aisément dans le processus industriel de développement d'un système sans bouleverser la livraison des documents contractuels.

3. Pré-AMDEC

Dès que l'analyse fonctionnelle est achevée, l'outil de génération de pré-AMDEC fonctionnelle (cf Chapitre III) peut être utilisé. Notons que nos algorithmes de rédaction de pré-AMDEC fonctionnelle peuvent être appliqués à une analyse fonctionnelle incomplète. Ainsi l'AMDEC peut être réalisée partie par partie au fur et à mesure que l'analyse fonctionnelle se complète. Cependant, pour assurer des résultats optimaux de notre outil, un modèle complet d'analyse fonctionnelle (sans pin libre, sans fonction non reliée au reste de l'architecture, …) est indispensable. Le Tableau V.1 ci-dessous est un extrait de cette pré-AMDEC fonctionnelle correspondant aux extraits d'analyse fonctionnelle du projet LEA présenté dans le §III.2 de ce chapitre.

Tableau V.1 : Extrait de la pré-AMDEC fonctionnelle du véhicule LEA

Id	Fonction	Modes de Défaillance	Causes	Effets Locaux	Effets Système	Gravité	Fréquence	Détectabilité	Criticité	Moyens de Prévention	Moyens de Protection	Moyens de Détection	Composants support	Exigences impactées
FD.12.1	Gérer la séquence de vol	Perte de la fonction ; Fonction exécutée de façon intempestive ; Retard d'exécution de la fonction ; Démarrage de la fonction impossible ; Arrêt de la fonction impossible ; Fonction intermittente ; Fonction dégradée	[CRSEP] Capter les évolutions de missions	[TOUV] Ouvrir le godet ; [TESSAI] Mettre en route le moteur ; [TALLUM] Mettre en route le moteur ; [TBLOC] Mettre en route le moteur ; [TTORCHE] Mettre en route le moteur ; [TESSAI] Contrôler le vol auto-propulsé ; [TFIN] Contrôler le vol auto-propulsé ; [TFIN] Contrôler le vol non propulsé ;	[Besoin] Contrôler le vol autonome *(phase: Vol autonome de LEA)*								Logiciel de vol (*Type : Logiciel)*; Calculateur de vol *(Type : Carte Electronique)*	[REQ_LEA_STB_370_a] Séquence d'allumage *(Le calculateur doit respecter la séquence décrite à la section 7.4 pour déclencher les ordres suivants : TOUV, TESSAI, TALLUM, TBLOC, TTORCHE, TFIN)*
FD.12.2	Ouvrir le godet	Perte de la fonction ; Fonction exécutée de façon intempestive ; Retard d'exécution de la fonction ; Démarrage de la fonction impossible ; Arrêt de la fonction impossible ; Fonction intermittente ; Fonction dégradée	[TOUV] Gérer la séquence de vol		[Besoin] Contrôler le vol autonome *(phase: Vol autonome de LEA)*								Système d'alimentation gaz *(Type : Alimentation gaz)*	[REQ_GAZ_STB_20] Réception TOUV *(Le godet du véhicule doit s'ouvrir sur réception du signal TOUV)*
FD.14	Contrôler le vol auto-propulsé	Perte de la fonction ; Fonction exécutée de façon intempestive ; Retard d'exécution de la fonction ; Démarrage de la fonction impossible ; Arrêt de la fonction impossible ; Fonction intermittente ; Fonction dégradée	[TESSAI] Gérer la séquence de vol ; [TFIN] Gérer la séquence de vol ; [Attitude] Calculer l'attitude du véhicule ; [MesuresMoteur] Mesurer les grandeurs physiques		[Besoin] Contrôler le vol autonome *(phase: Vol autonome de LEA)*								Logiciel de vol *(Type : Logiciel)* ; Calculateur de vol *(Type : Carte Electronique)* ; Système d'alimentation gaz *(Type : Alimentation gaz)*	[REQ_LEA_STB_375] Executer l'algorithme de contrôle du moteur *(L'algorithme de régulation des carburants et comburant du moteur est exécuté et les consignes transmises au système d'alimentation gaz)*

FD.15	Contrôler le vol non propulsé	Perte de la fonction ; Fonction exécutée de façon intempestive ; Retard d'exécution de la fonction ; Démarrage de la fonction impossible ; Arrêt de la fonction impossible ; Fonction intermittente ; Fonction dégradée	[TFIN] Gérer la séquence de vol		[Besoin] Contrôler le vol autonome *(phase: Vol autonome de LEA)*								Logiciel de vol *(Type : Logiciel)* ; Calculateur de vol *(Type : Carte Electronique)* ; Système d'alimentation gaz *(Type : Alimentation gaz)*	[REQ_LEA_STB_380] Stopper l'alimentation gaz *(À la réception du TFIN, les consignes d'alimentation gaz sont maintenues à zéro)*
FD.13.1	Activer la bougie d'allumage de la torche	Perte de la fonction ; Fonction exécutée de façon intempestive ; Retard d'exécution de la fonction ; Démarrage de la fonction impossible ; Arrêt de la fonction impossible ; Fonction intermittente ; Fonction dégradée	[TESSAI] Gérer la séquence de vol ; [TALLUM] Gérer la séquence de vol ;	[TorcheActive] Allumer la torche	[Besoin] Contrôler le vol autonome *(phase: Vol autonome de LEA)*								Système d'alimentation gaz *(Type : Alimentation gaz)*	[REQ_GAZ_STB_30] Activer la torche *(Lorsque TESSAI et TALLUM ont été reçus, la bougie d'allumage de la torche est activée)*
FD.13.2	Effectuer le blocage aero du moteur	Perte de la fonction ; Fonction exécutée de façon intempestive ; Retard d'exécution de la fonction ; Démarrage de la fonction impossible ; Arrêt de la fonction impossible ; Fonction intermittente ; Fonction dégradée	[TESSAI] Gérer la séquence de vol ; [TBLOC] Gérer la séquence de vol ;		[Besoin] Contrôler le vol autonome *(phase: Vol autonome de LEA)*								Système d'alimentation gaz *(Type : Alimentation gaz)*	[REQ_GAZ_STB_35] Blocage aero du moteur *(Lorsque TESSAI et TBLOC ont été reçus, le blocage aero du moteur est réalisé)*
FD.13.3	Allumer la torche	Perte de la fonction ; Fonction exécutée de façon intempestive ; Retard d'exécution de la fonction ; Démarrage de la fonction impossible ; Arrêt de la fonction impossible ; Fonction intermittente ; Fonction dégradée	[TESSAI] Gérer la séquence de vol ; [TTORCHE] Gérer la séquence de vol ; [TorcheActive] Activer la bougie d'allumage de la torche		[Besoin] Contrôler le vol autonome *(phase: Vol autonome de LEA)*								Système d'alimentation gaz *(Type : Alimentation gaz)*	[REQ_GAZ_STB_40] Allumage de la torche *(Lorsque TESSAI et TTORCHE ont été reçus, l'allumage de la torche est effectué)*

Remarque : La pré-AMDEC fonctionnelle générée automatiquement comporte une ligne par mode de défaillance générique plus une ligne vierge pour que l'expert puisse décrire des modes de défaillance spécifique. Par souci de lisibilité de cet extrait de pré-AMDEC fonctionnelle, une seule ligne par fonction à été gardée.

La génération de la pré-AMDEC fonctionnelle du véhicule LEA a permis de tester sur un exemple concret notre outil logiciel (cf. Chapitre III). Les résultats sont satisfaisants tant en terme de qualité de la pré-AMDEC générée, qui est conforme au format défini dans le chapitre III, qu'en terme de temps d'exécution, puisque la génération de la pré-AMDEC fonctionnelle ne prend que quelques secondes. Concernant cette pré-AMDEC du véhicule LEA, notre partenaire industriel (MBDA) a validé l'intérêt de l'outil ainsi que l'approche générale consistant à limiter l'interaction de l'expert SdF avec le modèle SysML qu'il ne maîtrise pas.

Il est cependant intéressant de noter que les limites de notre outil correspondent aux limites de l'outil logiciel utilisé pour la modélisation SysML et la génération du fichier XMI utilisé en entrée de la génération de pré-AMDEC. En effet, dans le cadre d'un projet annexe à la thèse, nous avons pu expérimenter des limites dans la taille du fichier XMI que l'outil utilisé par notre partenaire industriel est capable de fournir. Ce bug de l'outil de modélisation SysML nous a forcé à adapter notre couche d'extraction des informations d'analyse fonctionnelle pour nous adapter à un autre format de fichier que XMI. Après cette modification, les algorithmes présentés dans le chapitre II ont permis la rédaction de la pré-AMDEC fonctionnelle. La taille conséquente du modèle SysML de notre partenaire n'est donc pas un frein à la génération de la pré-AMDEC fonctionnelle qui comporte, dans ce cas, plusieurs milliers de lignes.

IV. Evaluation de fiabilité d'un COTS avec FIDES

Suite à l'analyse fonctionnelle et à l'AMDEC fonctionnelle réalisée sur la base de la pré-AMDEC générée automatiquement par l'outil logiciel présenté dans le chapitre III, les activités d'IS ont finalement défini l'architecture organique de LEA. Cette architecture illustrée par un BDD (Figure V.8), présente les équipements principaux qui composent l'architecture du véhicule LEA. La plupart de ces équipements ont été modélisés à des niveaux de détails plus fins, cependant certains équipements ou composants ne sont pas détaillés dans le modèle système, faute de connaissances précises de leur architecture et/ou de leur comportement. Par exemple, le système de télémesure n'est pas détaillé puisqu'il s'agit d'un équipement russe fourni par RADUGA et pour lequel nous de connaissons que les interfaces avec nos équipements. La télémesure fournie par RADUGA correspond donc à un COTS qu'il faut intégrer au système LEA.

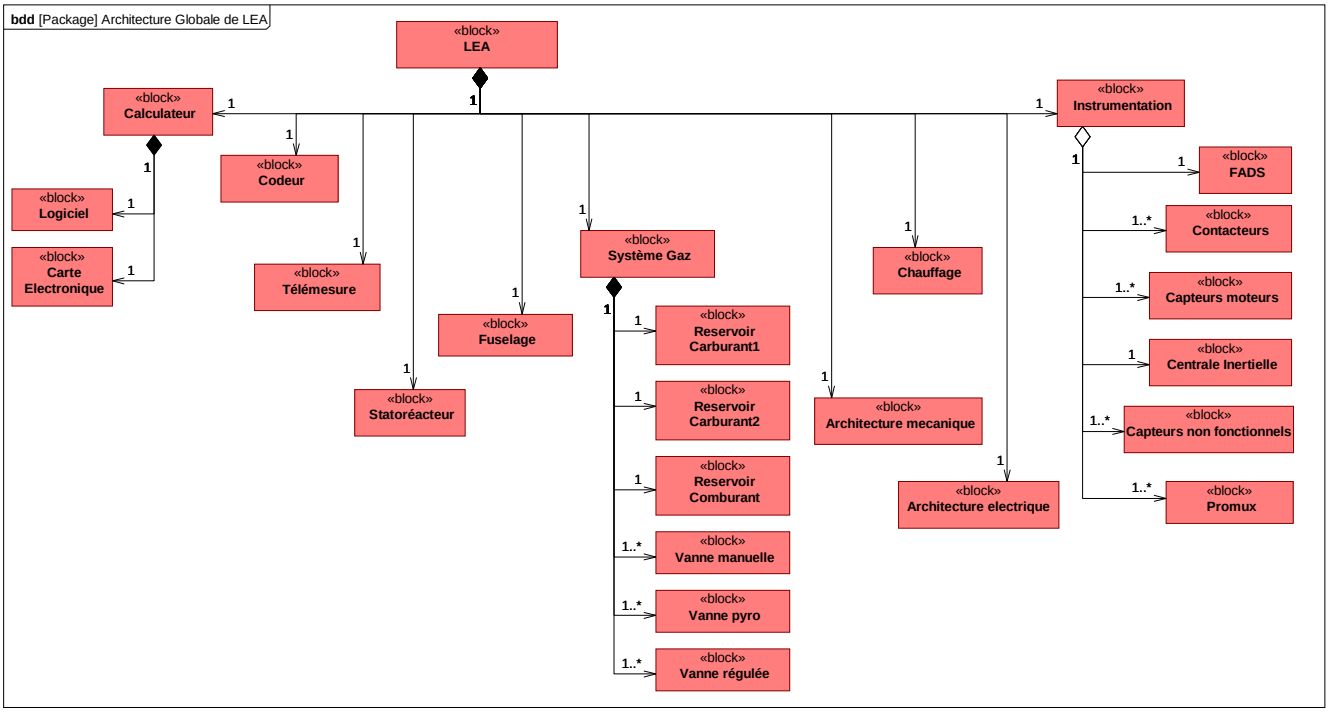


Figure V.8 : BDD présentant l'architecture globale du véhicule LEA

Le système LEA contient plusieurs équipements, sous-équipements, cartes électroniques et composants qui sont fournis par des partenaires du projet ou par des fournisseurs tiers et qui correspondent à des COTS. Dans cette section, nous allons illustrer notre propos concernant l'évaluation de fiabilité d'un COTS avec un composant utilisé dans le projet LEA : Le Promux PM8TC. Le Promux est un système électronique permettant de lire plusieurs entrées capteurs et retransmettre l'ensemble de ces données sur demande sur un réseau RS485 avec un protocole de type MODBUS.

Pour l'évaluation des COTS tel que le Promux, la méthodologie FIDES s'applique de la même facon que ce que nous avons décrit dans le chapitre IV, à un détail près : la décomposition du produit ne s'effectue plus en détaillant l'ensemble des articles qui le compose, mais en détaillant :

- L'ensemble des fonctionnalités techniques réalisées par le COTS.
- Les interfaces physiques du COTS avec le reste du système.

Cette manière de décrire les fonctionnalités techniques et les interfaces est spécifique à l'évaluation de la fiabilité d'un COTS selon FIDES. La liste des fonctionnalités techniques à considérer est finie :

- **Fonctions communes toutes cartes**
  - Fonctions communes
- **Fonctions numériques centrales**
  - Fonction CPU
  - Fonction Mémoire FLASH Boot (NOR)
  - Fonction Mémoire FLASH Stockage (NAND)
  - Fonction Mémoire DRAM (DDR-SDRAM, SGRAM)
  - Fonction L2, L3 cache ou SRAM
  - Fonction contrôleur SCSI

- Fonction Chipset (Northbridge, Southbridge)

- **Fonctions numériques périphériques**
  - Fonction contrôle Ethernet (LAN)
  - Fonction contrôle Graphique (VGA)
  - Fonction contrôle Fieldbus (CAN, ARINC, 1553)
  - Fonction contrôle Wireless (Bluetooth, WIFI)
  - Fonction conversion analogique/digital ou digital/analogique

Les interfaces du COTS sont elles aussi associées à une catégorie proposée par FIDES :

- **Interfaces entrées/sorties**
  - Ligne numérique bus parallèle
  - Ligne analogique ou discrète
  - Ligne périphérique série (RS232, RS485, RS422, USB, souris, clavier, ethernet)
  - Ligne bus série (CAN, ARINC, 1553)
  - Isolation d'entrée sortie par optocouplage
  - Isolation ou aiguillage d'entrée/sortie par relais électromécanique

Les équations d'évaluation présentées dans le chapitre IV et leurs paramètres (par exemple : $\Pi_{placement}$, $C_{sensitivity}$, …) sont communes aux deux méthodologies FIDES (Classique et Spécifique COTS), seule la décomposition des éléments unitaires étudiés diffère. Ainsi, pour les cartes COTS, les interfaces et les fonctionnalités sont équivalentes aux articles de la méthodologie classique et chacun de ces éléments (interface ou fonctionnalité) possède les mêmes paramètres FIDES d'évaluation de fiabilité qu'un article dans la méthode classique.

Dans le cadre de notre approche ISBM supportée par SysML (cf. Chapitre II), la description des interfaces d'un COTS reste identique à ce qui est réalisé pour n'importe quel autre composant. Cette description se fait à l'aide de *flowports* visibles au niveau BDD ou IBD (Figure V.9).

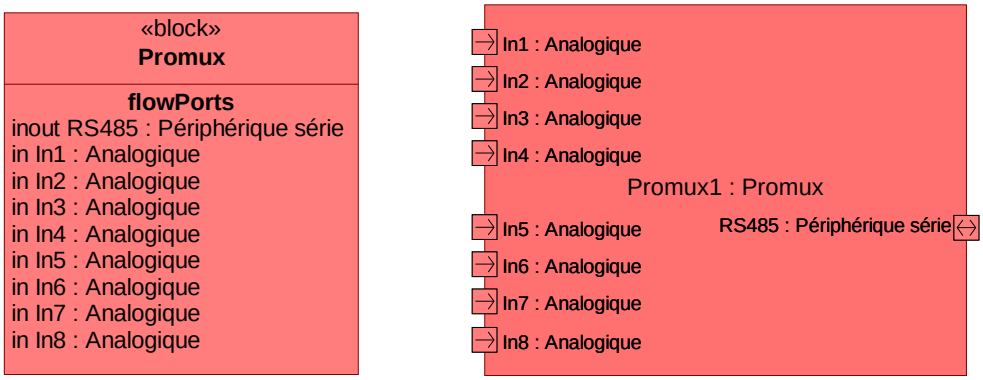


Figure V.9 : Représentation des interfaces d'un composant aux niveaux BDD et IBD

Cependant, les fonctionnalités génériques utilisées pour décrire un COTS dans le cadre FIDES ne sont pas traditionnellement utilisées, ni lors de l'analyse fonctionnelle, ni lors de la description organique des composants. Ainsi, deux solutions sont envisageables :

- Nous considérons qu'il s'agit d'informations propres à l'évaluation de la fiabilité et elles ne sont pas réifiées dans le modèle système. Les règles de modélisation du chapitre II sont respectées. L'expert SdF devra réaliser lui-même l'évaluation des fonctionnalités du COTS.

- Nous considérons qu'il s'agit de données de description du COTS devant être réifiées dans le modèle système et nous proposons alors une méthode de réification de ces fonctionnalités en SysML afin de préparer le travail de l'expert.

Dans le cas de la deuxième solution, afin de réifier ces informations tout en respectant les règles décrites dans le chapitre II, nous avons modélisé les fonctionnalités du COTS sous la forme d'une décomposition organique du COTS avec des *blocks* portant les noms des fonctionnalités décrites dans FIDES. En ne décrivant que le BDD comme sur la Figure V.10, nous ne décrivons pas de flux (qui serait modélisé dans un IBD) risquant d'être mal interprété par le processus de génération d'AMDEC composant décrit par [David 2009]. La Figure V.10 présente cette description pour le Promux.

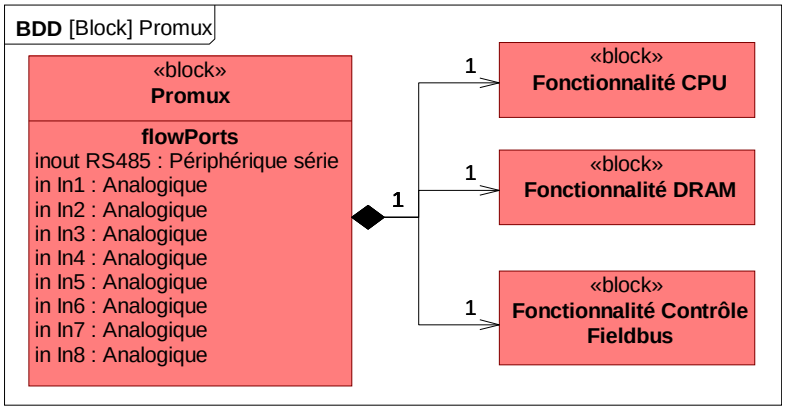


Figure V.10 : Décomposition du Promux en fonctionnalités techniques

Dans le guide FIDES, l'ensemble des valeurs des paramètres d'évaluation de la fiabilité est donné, fonction par fonction et interface par interface, de la même façon que pour les articles lors de l'application de la méthodologie FIDES classique. Il est ainsi possible de créer des entrées dans la BCD : « Fonctionnalité CPU », « Fonctionnalité DRAM », … afin de stocker ces paramètres comme nous le suggérions dans le chapitre IV dans le cadre de la méthodologie FIDES classique. De même, les paramètres dépendant du cycle de vie du système sont extraits du diagramme de contexte de LEA comme cela avait été défini dans le chapitre IV.

La décomposition du COTS en fonctionnalités techniques pourra aussi être pérennisée dans la BCD. En effet, la décomposition organique telle que présentée sur la Figure V.10 se traduit par la création de propriétés *parts*. Les *parts* sont le plus souvent représentés uniquement dans un diagramme IBD, mais il s'agit malgré tout de propriétés intrinsèques au composant qui sont donc stockées dans la BCD sous le stéréotype « TypeDeComposant ». La Figure V.11 présente l'affichage de ces propriétés sur le *block* « Promux » après modélisation de la décomposition présentée dans la Figure V.10. Les *parts* ne sont pas instanciés dans un IBD, ils ne sont donc pas nommés ce qui explique l'absence de nom pour les *parts* sur la Figure V.11. Cette possibilité du langage SysML et de la BCD se révèle particulièrement utile dans le cas où la décomposition du COTS en fonctionnalités techniques n'est pas réifiée dans le modèle système.

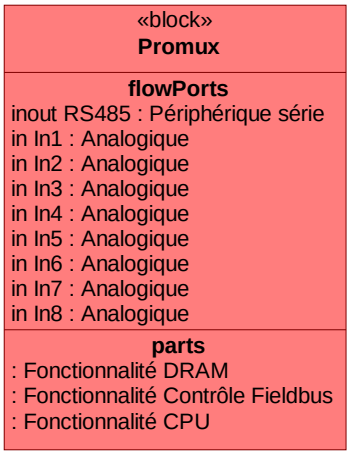


Figure V.11 : Représentation de la décomposition organique au niveau BDD

Dans le cadre du projet LEA, nous avons maximisé l'usage du modèle système et de la connexion de la BCD avec la méthodologie FIDES. En effet l'association du modèle SysML, du cadre MéDISIS (comprenant la BCD) et de la méthodologie FIDES (qui permet l'étude de composants électroniques et de COTS) nous a permis de réaliser des études de fiabilité des composants critiques y compris de COTS tel que le Promux.

V. Valorisation de l'ISBM pour la conception et la validation de LEA

1. Analyse d'ordonnancement

De par la nature et ses fonctions au sein du véhicule, le système embarqué de LEA est assujetti à de nombreuses contraintes temporelles. Pour analyser précisément ces points, nous avons étudié différentes solutions méthodologiques et techniques, tout en prenant en considération la valorisation de l'approche ISBM et le modèle système en SysML. Pour cela plusieurs alternatives étaient envisageables :

- Le langage MARTE est une extension d'UML spécifique pour la modélisation de systèmes embarqués temps réels.
- Le langage AADL, que nous avons présenté dans le chapitre I, est un langage objet textuel et graphique permettant lui aussi la modélisation de systèmes embarqués temps réels.

Ces deux langages permettent une description plus formelle et plus fine des contraintes temporelles et des contraintes liées aux systèmes embarqués (dualité matériel et logiciel) qu'avec un modèle SysML. En 2009, au début du projet LEA, la décision a été prise d'utiliser AADL pour réaliser les études temporelles du système LEA. En effet, l'utilisation d'AADL offrait plusieurs atouts par rapport à MARTE :

- La possibilité de modéliser des modes de défaillances avec les entités standards du langage AADL [Feiler & Rugina 2007][SAE 2006].
- Un ensemble d'outils associé au langage permettant différents types d'études. Notamment un outil d'ordonnancement de tâche : Cheddar [Singhoff 2007].

Depuis cette décision, l'écart d'intérêt entre ces deux solutions s'est réduit puisque l'outil Cheddar accepte maintenant des spécifications de système modélisé en MARTE, et la proximité de MARTE avec SysML permet d'envisager l'extension de la BCD à MARTE pour la réification des comportements dysfonctionnels.

Dans le cadre de notre travail avec l'approche ISBM et le cadre MéDISIS, nous avons mis au point une transformation du modèle système SysML vers un modèle AADL permettant d'effectuer les études qui nous intéressent. Comme avec tous les processus MéDISIS décrits par [David 2009] et dans les chapitres précédents, le but de cette traduction est de permettre la réutilisation des connaissances contenues dans le modèle SysML, pour faciliter l'analyse spécifique propre à un DSL (ici une analyse du comportement temps réel du système).

L'approche orientée objet de SysML comme d'AADL permet une traduction efficace des concepts architecturaux de l'un vers l'autre. Cependant, AADL est un langage de modélisation de plus bas niveau que SysML et son domaine de représentation est plus restreint. En effet, SysML permet, a priori, de modéliser n'importe quel type de composants, ce qui n'est pas le cas en AADL puisqu'il n'accepte que 10 catégories de composants (Memory, Processor, Bus, Device, Process, Thread, Data, Thread Group, Subprogram, System). Lors de la traduction de SysML vers AADL, les composants de notre système embarqué devront être attribués à ces catégories. Selon les règles de modélisation présentées au chapitre II, le modèle système en SysML ne permet pas de déduire ce classement. Plusieurs techniques complémentaires sont alors envisageables pour combler ce manque :

- Faire appel à l'expert pour classer chaque composant.
- Utiliser la BCD qui stocke l'appartenance d'un type de composant en SysML à une catégorie en AADL. Ces informations peuvent alors être basées sur le retour d'expérience de précédentes analyses.

Un problème similaire se présente pour définir toutes les propriétés qui dimensionnent le système, notamment les propriétés de temps d'exécution des tâches logicielles, les priorités de préemption, la taille des données échangées sur les bus,… Ces propriétés quantifiées sont indispensables à l'utilisation d'outils tels que Cheddar ou des études de type Rate Monotonic Analysis [Klein et al. 1993]. Pour résoudre ces problèmes, nous avons opté pour une méthode calquée sur les travaux de génération de modèle Altarica DF présentée dans [David 2009] : faire appel à un expert pour compléter notre modèle et maintenir la BCD pour limiter le travail de l'expert pour les prochaines évolutions du système ou les futurs projets.

Du fait que la BCD se base sur le « TypeDeComposant », seuls les paramètres des composants physiques peuvent être maintenus dans la BCD. Les modifications inévitables du méta-modèle de la BCD pour accueillir ces paramètres ne seront pas décrites ici, mais ne diffèrent que peu par rapport à la définition de paramètres de défaillance. Notons que la BCD ainsi « augmentée » dévierait de son attribution initiale en maintenant des informations non dysfonctionnelles (taille de bus, caractéristique d'exécution de tâches des processeurs, …). Un équivalent de la BCD pour les tâches logicielles pourrait aussi être envisagé (de la même façon que nous avons décrit la BDF pour les fonctions).

Finalement les différentes étapes suivies pour construire un modèle AADL se résument ainsi :

Etape n°1. Identifier tous les *blocks* et *parts* SysML et établir la hiérarchie entre toutes ces entités, en prenant en compte les différents niveaux de conception.

Etape n°2. Cartographier les relations entre composants en utilisant les *flowports* et les flux entre eux.

Etape n°3. Classifier par les catégories adaptées chaque composant du système (par exemple : un *block* « Mémoire_Partagée » appartient à la catégorie « Memory »).

Etape n°4. Créer le modèle structurel en AADL (l'architecture du système peut être construite textuellement et graphiquement).

Etape n°5. Renseigner l'ensemble des propriétés qui ne peuvent être déduites du modèle SysML.

Etape n°6. Créer le modèle AADL final complet, analysable par les outils tels que Cheddar

Les étapes 1, 2, 4, 6 peuvent être automatisées, les étapes 3 et 5 nécessitent de faire appel à un expert métier aidé pour cette activité de renseignement par la BCD.

Le Tableau V.2présente les correspondances entre les concepts principaux des deux langages : SysML et AADL. Ce tableau est le support des premières étapes de traduction : 1, 2 et 4.

Tableau V.2 : Table de correspondance des concepts entre AADL et SysML

Concepts	**AADL**	**SysML**
Composant logiciel /Implémentation	Composant logiciel /Implémentation	*Block* *Part*
Composant matériel /Implémentation	Composant matériel /Implémentation	*Block* *Part*
Relation	Binding	*Relation d'agrégation et de composition*
Connecteurs Flux	Port Connections Event, Data, Data-Event In, Out, Inout	*Flowports* *Value type / Block* *Direction*
États	Modes	*State Diagram/state*
Propriétés	Properties	*Requirement Diagram,* *Parametric Diagram*

Ce processus de traduction permet d'obtenir un modèle AADL du fonctionnement du système qui pourra être analysé par Cheddar pour obtenir des évaluations de conformité et de performance. Cependant, comme nous l'avons déjà évoqué, le langage AADL dispose de moyens de représentation des modes de défaillance ainsi, en appliquant notre processus de traduction aux données dysfonctionnelles contenues dans la BCD, il nous est possible d'obtenir un modèle AADL de fonctionnement et de dysfonctionnement de notre système.

L’exemple le plus parlant de l’utilité de ce processus est l’étude de l’âge des données issues des capteurs. Cet âge correspond au temps entre l’observation effective d’une grandeur physique par un capteur et la prise en compte de cette valeur par le logiciel de vol. Ce paramètre a piloté au début du projet le choix de l’architecture générale du système embarqué afin d’optimiser la réactivité du logiciel de vol. Dans l’architecture choisie, ce paramètre influence encore différentes contraintes temporelles que nous avons dû étudier. Pour réaliser cette étude, nous nous sommes basés sur la description organique des équipements impliqués dans le traitement de ces données capteurs (Figure V.12).

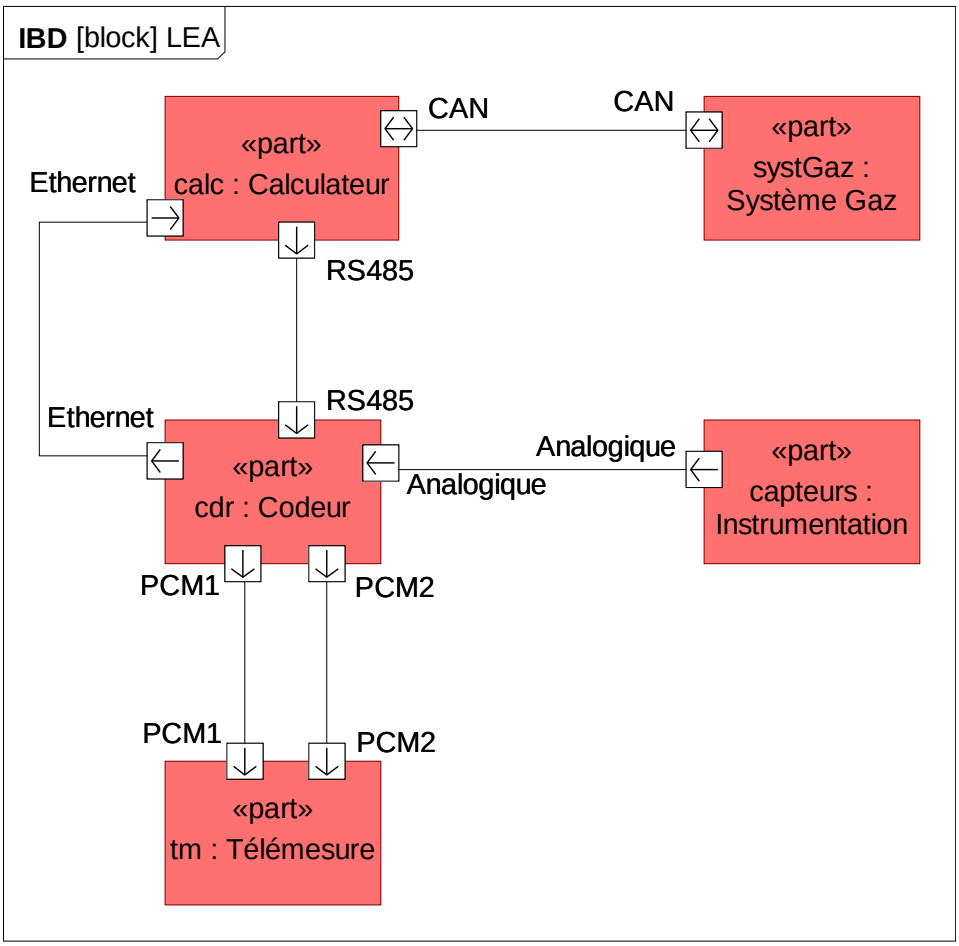


Figure V.12 : IBD du véhicule LEA

En SysML, afin de détailler les échanges entre *parts*, nous utilisons un *sequence diagram* (SD, cf. Chapitre II). Le SD de la Figure V.13 présente la chaine d’acquisition des données capteurs jusqu'à leur stockage dans le calculateur. Sur la droite de ce SD, des propriétés temporelles appartenant aux différents *parts* sont représentées (par exemple le temps de mise à disposition des mesures par le codeur : T_Disp_co, …)

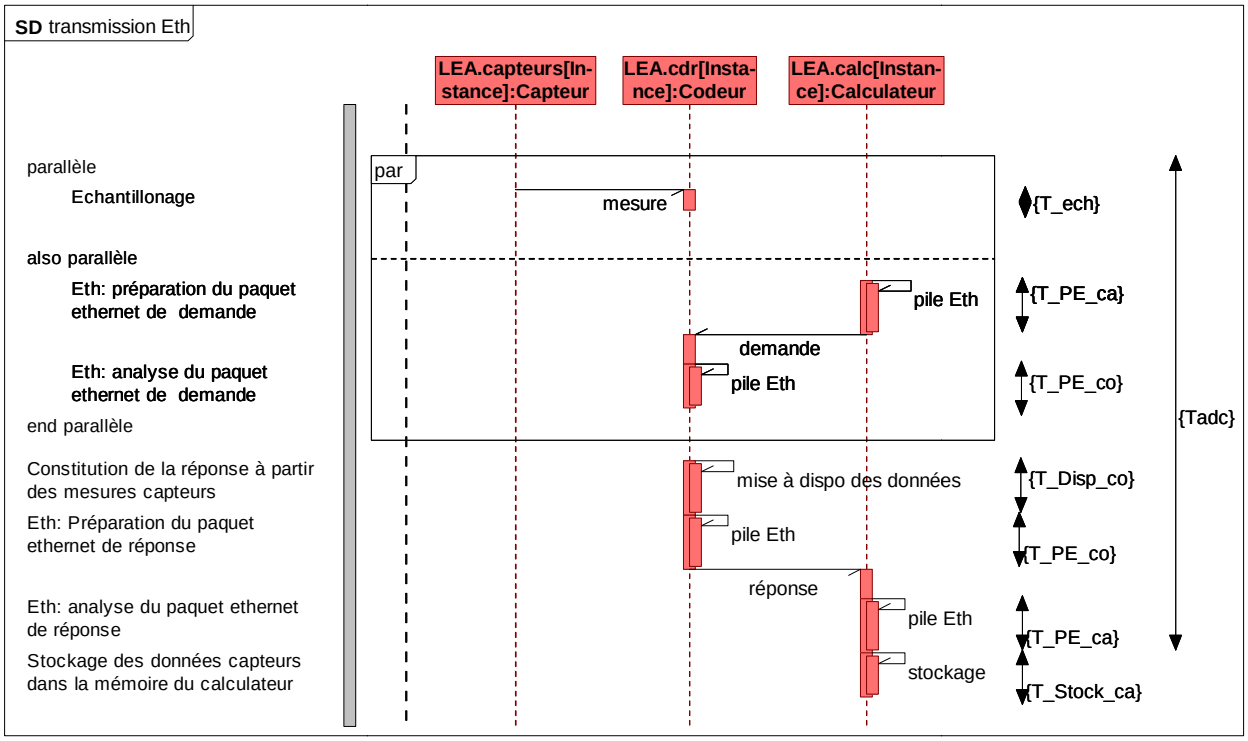


Figure V.13 : SD de la transmission Ethernet des données capteurs

Après application du processus de traduction défini précédemment, complété par les connaissances de l'expert, nous obtenons le modèle AADL présenté Figure V.14. Ce modèle est ensuite analysé par Cheddar afin d'étudier les contraintes d'ordonnancement du système.

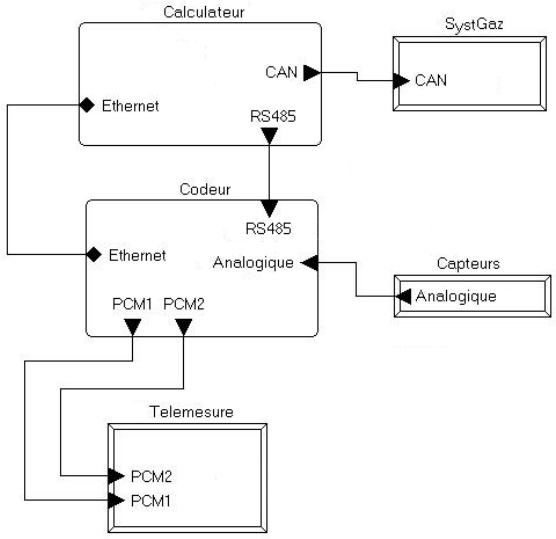


Figure V.14 : Modèle AADL du véhicule LEA

Les détails de l'étude d'ordonnancement réalisée par l'outil Cheddar sortent du périmètre de la thèse. Il est cependant important de noter que cette étude a permis de déterminer un âge maximum que les données ne doivent pas dépassées avant leur prise en compte par le calculateur. Ce résultat a permis de compléter le modèle système en SysML en ajoutant une contrainte définissant l'évaluation de l'âge des données : Tadc, à partir des propriétés temporelles des différents équipements, ainsi que la limite que Tadc ne doit pas dépasser. Ces contraintes sont représentées dans le ParD de la Figure V.15.

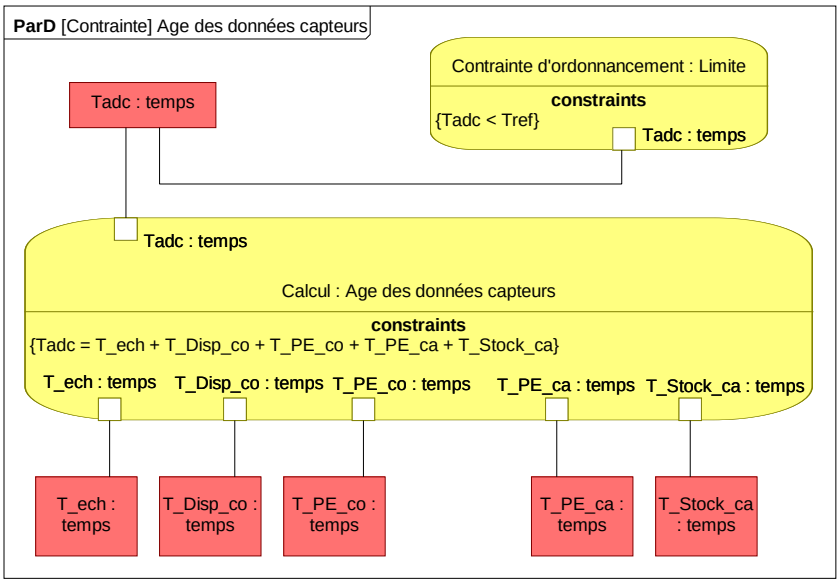


Figure V.15 : Modélisation de la contrainte sur l'âge des données capteurs avec un ParD

Finalement, nous voyons que l'extension de MéDISIS à des processus de traduction pour des DSL n'appartenant pas spécifiquement au domaine de la SdF est possible et que cela peut s'intégrer à la méthodologie MéDISIS et à l'approche ISBM sous-jacente.

L'illustration des bénéfices d'un tel processus a été présentée à travers l'analyse de l'âge des données capteurs dans le cadre du projet LEA. Bien que, les risques liés aux contraintes temporelles aient été assez vite identifiés et maitrisés, rendant le processus de traduction vers AADL peu utile pour la suite du projet, les spécifications temporelles définies lors de ces études ont été maintenues dans le modèle système tout au long des différentes évolutions du système LEA.

2. Simulation et injection de fautes

La simulation d'un système dynamique est très souvent indispensable dans le cadre de projet innovant où la réalisation de prototype est très onéreuse. Pour de nombreuses raisons tant techniques que méthodologiques ou scientifiques, le monde industriel a très souvent recours au logiciel de conception et simulation de système dynamique : Simulink. Comme précédemment, dans le cadre de notre travail avec l'approche ISBM et le cadre MéDISIS, nous avons mis au point un processus de traduction des éléments SysML vers des entités Simulink, afin de permettre, de nouveau, de valoriser le modèle système pour la réalisation d'études spécifiques.

Dans un premier temps, on peut aisément remarquer des correspondances entre les éléments de modélisation SysML et ceux de Simulink : les **Blocks** et les **Line** sont les entités de base d'un modèle Simulink. Un **block** représente un système qui peut contenir d'autres systèmes (**subsystem** qui est aussi un block). Les systèmes utilisent des ports d'entrée (**Inport**) et des ports de sorties (**Outport**). Une **line** relie deux **blocks** ensemble. On retrouve une organisation similaire au sein de multiples diagrammes de SysML.

Un **block** Simulink correspond à un *block* SysML et un **subsystem** correspondra à un des éléments d'un *internal block diagram*. Les **lines** entre les **blocks** Simulink correspondent aux flux de données

qui transitent par un **port** Simulink. Ils seront directement associés aux flux qui transitent entre deux *flowports* de SysML. L'ensemble des *flowports* nous permet de définir la structure des **inport**/**outport**. Au niveau de la représentation du comportement des composants, les éléments **Stateflow** en Simulink sont un sur-ensemble des *Statemachines* de SysML. Les contraintes imposées au système, modélisées à l'aide de *parametric diagram* en SysML seront représentées à l'aide de **blocks** et **lines** elles aussi. Enfin, les *requirements* réifiés en SysML pourront être associés à leurs composants en Simulink. Cependant, en accord avec ces observations, nous pouvons définir la table des équivalences entre les concepts manipulés par chacun des langages.

Tableau V.3 : Table de correspondance des concepts entre Simulink et SysML

Concept	**Simulink**	**SysML**
Composants	Block	*Block / Part*
Relation	Line	*Block Association*
Sous-composants	Subsystems	*Part*
Connecteurs Flux	Inport / Outport Line	*Flowports* *flux*
Machine à Etats	Stateflow Diagram	*Statemachines*
Contraintes	Block	*Parametric diagram /Constraint block*
Relation entre Contraintes	Line	*Parametric diagram /Connections*
Exigences	Associated Exigences	*Requirement Diagram*

Comme nous pouvons le voir dans le Tableau V.3, différents types d'éléments de modélisation SysML peuvent êtres transformés en un même type d'éléments en Simulink. Par exemple : les **lines** en Simulink représenteront aussi bien les flux entre *flowports* que les connexions dans un ParD. En effet, la traduction de SysML vers Simulink est surjective, ce qui signifie qu'il y aura une perte d'information lors du processus de traduction, ou tout du moins une perte de précision dans la représentation du système. Ce fait induit aussi que les décisions de conception impactant les spécifications du système ne peuvent pas être réintroduit en SysML en inversant le processus de traduction. L'action d'un expert système est alors nécessaire pour mettre à jour le modèle SysML.

L'ajout de ce processus de traduction à MéDISIS permet d'obtenir un modèle fonctionnel en Simulink comme nous l'avons fait avec AADL. De même que pour le processus AADL, c'est l'application du processus de traduction sur les concepts modélisés dans la BCD qui permettra d'ajouter le comportement dysfonctionnel au modèle fonctionnel que nous venons d'obtenir. Les mécanismes et techniques de modélisation nécessaire à l'obtention d'un modèle permettant la simulation du système aussi bien en état nominal que subissant différents modes de défaillance ont été détaillés dans plusieurs publications [Cressent et al. 2011a] [Cressent et al. 2011c]. La technique la plus complète consiste à appliquer le même raisonnement que celui qui régit la BCD : le mode de défaillance d'un composant hérite du composant et y ajoute les comportements défaillants. Ainsi de façon similaire, le mode de défaillance en Simulink possédera au moins les mêmes interfaces que le

composant fonctionnel, afin de pouvoir se substituer à celui-ci lors du déclenchement de la défaillance. La Figure V.16 présente ce principe.

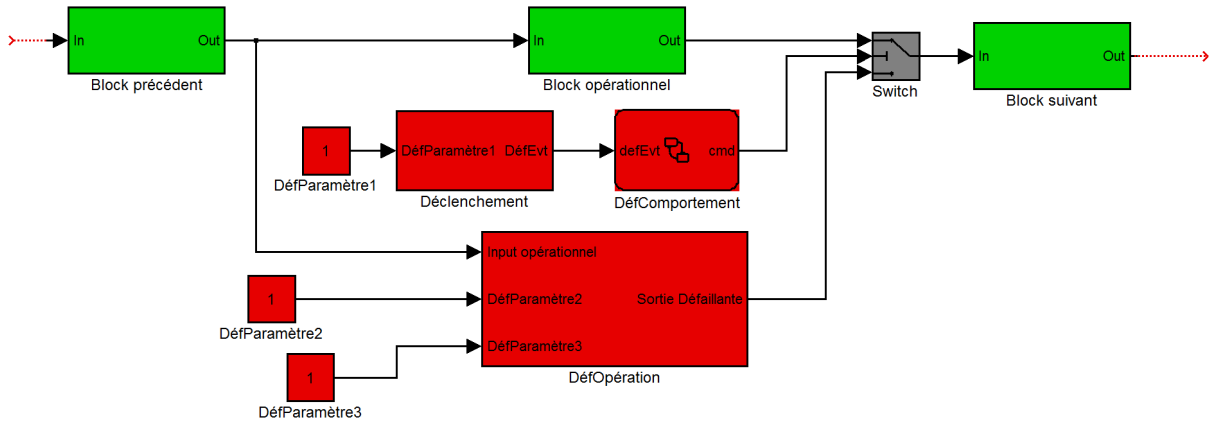


Figure V.16 : Modèle de simulation opérationnelle et dysfonctionnelle

On retrouve en vert, sur la Figure V.16, les éléments opérationnels issus de la traduction du modèle SysML. En rouge, ce sont les éléments issus de la traduction de la DBD. On retrouve les « DéfParamètres » utilisés par « DéfOpération » et « Déclenchement ». Le stateflow (diagramme d'état Simulink) intitulé « DéfComportement » réalise le même objectif que dans la BCD, il régit le comportement du duo : composant opérationnel / mode de défaillance, en permettant soit à la sortie opérationnelle d'être transmise au reste du système, soit à la sortie défaillante. Notons que dans le cas où le mode de défaillance décrit dans la BCD comporte plus d'une « DéfOpération », alors nous retrouverions plusieurs **blocks** « DéfOpérations » dans le modèle simulink, dont les différentes sorties défaillantes seraient reliées à un « Multiport Switch » (Figure V.17), toujours commandé par le stateflow « DéfComportement ». Ce « DéfComportement » prend ici en considération le résultat du **block** « Déclenchement » pour permettre la simulation du déclenchement de la défaillance. Cependant, pour réaliser un test de robustesse par injection de faute, les **blocks** « Déclenchement », « Défcomportement » et « Switch » peuvent être remplacés par un « Manual Switch » (Figure V.17) permettant à l'utilisateur de forcer la prise en compte de la sortie défaillante et ainsi étudier le fonctionnement du système avec une ou plusieurs défaillances.

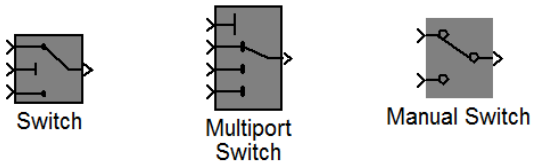


Figure V.17 : Différents types de « switch » en Simulink

Le processus de traduction de SysML vers Simulink permet donc de réaliser la simulation du système ainsi que des tests de robustesse. Une autre application de ce processus complète ces analyses.

En appliquant le processus de traduction aux contraintes du système (tel que celle illustrée dans la Figure V.15), il est possible de compléter le modèle Simulink du système par des **assertions**. Les **assertions** sont des **blocks** reliés aux sorties ou aux paramètres du système permettant de vérifier que les valeurs de ces derniers correspondent à une certaine contrainte. Tous comme en SysML, la contrainte peut dépendre de multiples paramètres et/ou sorties. Dans l'exemple de la Figure V.18, la

contrainte de calcul de l'âge des données capteurs est transcrite en Simulink. Les paramètres du système : T_ech, T_Disp_co, T_PE_co, T_PE_ca et T_Stock_ca proviennent directement du modèle opérationnel du système afin de calculer l'âge des données : Tadc. Celui-ci est ensuite soumis à un test par rapport à Tref. En fonction du résultat de ce test, l'**assertion** (qui correspond au **block** le plus à droite) pourra lever une erreur lors de la simulation du système lorsque la contrainte ne sera plus respectée.

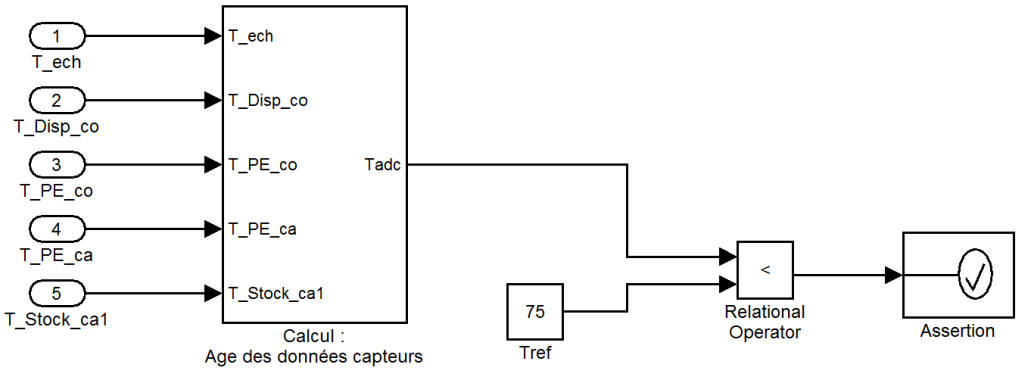


Figure V.18 : Assertion de la contrainte sur l'âge des données capteurs en Simulink

Par configuration de Simulink, l'état de l'assertion peut piloter plusieurs mécanismes :

- Lorsque l'assertion devient fausse, l'état du système est mémorisé et la simulation continue. À la fin de la simulation, ou à l'arrêt par l'utilisateur, un rapport de test est généré contenant le nombre de fois où l'assertion a été à l'état faux et quel était l'état du système à ce moment.
- Lorsque l'assertion devient fausse, la simulation s'arrête et la durée de simulation ainsi que l'état d'exécution du système sont mémorisés. Ainsi, lors d'un test de stabilité du système, de robustesse face à des déclenchements de défaillance, il est possible d'effectuer plusieurs simulations pour réaliser des statistiques.

Ces deux solutions sont également utiles pour le projet LEA. Notamment dans le cadre technique mis au point pour la conception détaillée du logiciel de vol de LEA et sa validation à différents niveaux de conception. Ils seront détaillés dans la partie suivante.

3. Développement du logiciel embarqué de LEA

Depuis le début de ce chapitre, le modèle système en SysML de LEA a été utilisé de multiples façons afin de générer et faciliter l'exploitation de modèles et documents spécifiques à des domaines d'expertises divers : SdF, analyse d'ordonnancement logiciel et évaluation de performances. Le dernier point que nous avons vu est la génération d'un modèle Simulink du système et d'**assertions** permettant de tester l'exécution de celui-ci. Dans le cadre du développement du logiciel de vol de LEA, ce modèle Simulink obtenu par traduction de SysML constitue un nouveau pivot pour les phases de conception détaillée et de validation. En effet, en ayant recours aux méthodes de générations de code C (optimisé pour des applications embarquées telles que c'est le cas pour le projet LEA), le modèle Simulink du logiciel de vol fait office de spécification exécutable, de prototype fonctionnel et de code source du livrable. De plus les

assertions issues des contraintes SysML jouent elles aussi un rôle majeur puisqu'elles interviennent dans la politique de validation fonctionnelle du modèle et pourront être réutilisées pour valider le logiciel final sur le calculateur de vol grâce aux boitiers de prototypage rapide que nous utilisons dans le cadre du projet LEA.

La méthode de conception et de validation d'un logiciel sur lequel nous nous sommes appuyées repose sur la définition de 3 niveaux de conception : MIL, SIL et HIL.

MIL (Model In the Loop) : Il s'agit d'un premier niveau de conception et de validation. À cette étape seul le modèle du système est disponible. Des jeux de tests sont mis au point (ou traduit depuis les contraintes SysML) pour former un modèle de test qui permet de valider le fonctionnement du système en simulation.

SIL (Software In the Loop) : Le code source du logiciel est généré à partir du modèle validé au niveau MIL. Ce code C est ensuite interfacé avec le même modèle de test que précédemment. Il s'agit d'un test de non-régression entre le niveau modèle et le niveau code source. Des outils Simulink permettent de réaliser le niveau SIL très simplement, par exemple les SFunction qui embarquent et exécutent du code C.

HIL (Hardware In the Loop) : Le code généré et validé au niveau SIL est compilé et embarqué sur sa plateforme cible. Le modèle de test est quant à lui transposé dans le monde réel soit par définition d'un banc de tests spécifique soit en ayant recours à un simulateur temps réel qui exécutera le modèle de test.

Dans le cadre du projet LEA, le modèle système en SysML donne lieu à une première version du modèle Simulink du logiciel de vol. Dans le cadre d'un développement MIL/SIL/HIL, un modèle de tests doit aussi être constitué. Les assertions provenant de la traduction des contraintes SysML forment une première source de données pour ce modèle de test que nous avons ensuite complété afin de mettre en place une politique de validation fonctionnelle du modèle du logiciel.

Au niveau MIL, la validation du modèle du logiciel est réalisée en plusieurs étapes :

- Validation de la modélisation : L'outil : Model Advisor de Mathworks permet la vérification de règles de modélisation. Notamment dans le cadre d'un projet où la génération de code est prévue, certaines règles spécifiques doivent être respectées. Dans le cadre du projet LEA, des règles issues de la norme de SdF IEC 61508 [IEC 61508] sont utilisées.
- Validation structurelle : Simulink propose aussi un outil de test de couverture. La vérification de la logique du modèle est alors réalisée (i.e. toutes les branches logiques sont-elles exécutées ?).
- Validation fonctionnelle : Il s'agit de définir des assertions sur les sorties et les paramètres du modèle qui permettent de valider la cohérence du fonctionnement avec les exigences et les contraintes du système.

Au niveau SIL, il existe des outils d'analyse de code C qui permettent de réaliser la vérification des règles de programmation (par exemple respect de la norme MISRA). De même, des tests de couverture similaire à ceux du niveau MIL (atteignabilité de toutes les branches logiques) sont réalisables par des outils spécifiques. L'ensemble des tests de validation fonctionnelle peut quant à

lui être rejoué au sein de Simulink en intégrant le code C dans un **block** SFunction qui se substituera au modèle du logiciel.

Au niveau HIL, nous avons opté pour l'utilisation d'un simulateur temps réel de la société dSPACE. Un boitier dSPACE de simulation temps réel est un boitier permettant d'exécuter du code embarqué et de communiquer à travers diverses interfaces numériques et analogiques couramment utilisées dans l'industrie (Figure V.19). Une des particularités d'un tel boitier par rapport aux calculateurs industriels de production est qu'il offre la capacité d'être utilisé grâce à un modèle graphique tant pour leur programmation que pour les observations du programme exécuté. L'utilisation d'un simulateur temps réel réduit ainsi la plupart des difficultés liées à la mise en œuvre d'un algorithme sur un calculateur. Dans le cas de dSPACE, la programmation du boitier se réalise par la modélisation de l'application en Simulink. Le boitier dSPACE est donc livré avec une librairie de **blocks** Simulink permettant de configurer le boitier et de préparer la communication à travers les entrées et sorties physiques du boitier. La logique de fonctionnement du boitier est quant à elle totalement réalisée avec le langage de modélisation Simulink.



Figure V.19 : Exemple de simulateur temps réel dSPACE

L'utilisation de ce type de matériel permet donc de réaliser la validation de niveau HIL sans avoir à développer un banc de test spécifique cohérent avec le modèle de test des niveaux MIL et SIL puisque le modèle initial pourra directement être exécuté par le boitier dSPACE. Le modèle de test du niveau MIL (et SIL) sera cependant agrémenté des blocks d'entrées sorties dSPACE avant de pouvoir être opérationnel sur le boitier. Cette étape d'adaptation est le seul traitement nécessaire à la réalisation des tests de validation fonctionnelle au niveau HIL.

D'un point de vue technique, l'utilisation de la technologie dSPACE permet aussi d'accéder au logiciel de pilotage ControlDesk de dSPACE qui permet d'instrumenter l'application exécutée sur le boitier avec une interface graphique dont la Figure V.20 est un exemple issu du projet LEA. De même, le logiciel AutomationDesk de dSPACE, permet d'effectuer l'ensemble des tests de validation de façon autonome (à travers une modélisation graphique de la logique d'exécution des tests) et de rédiger les rapports de tests.

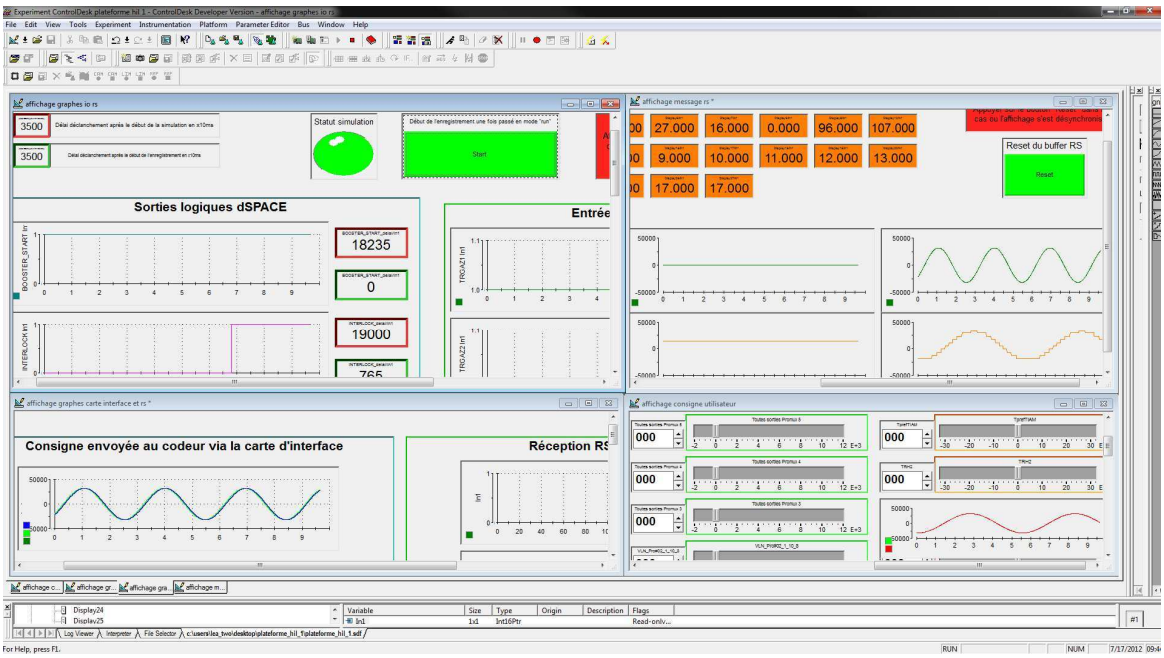


Figure V.20 : Interface de pilotage et d'observation du boitier dSPACE

Dans le cadre du projet LEA, deux principales configurations HIL ont été utilisées :

- Une configuration de validation fonctionnelle du logiciel de vol sur le calculateur de vol (KLEA). Cette configuration permet au boitier dSPACE d'exécuter l'ensemble des fonctions du modèle de test et de communiquer directement avec le calculateur de vol (Figure V.21).

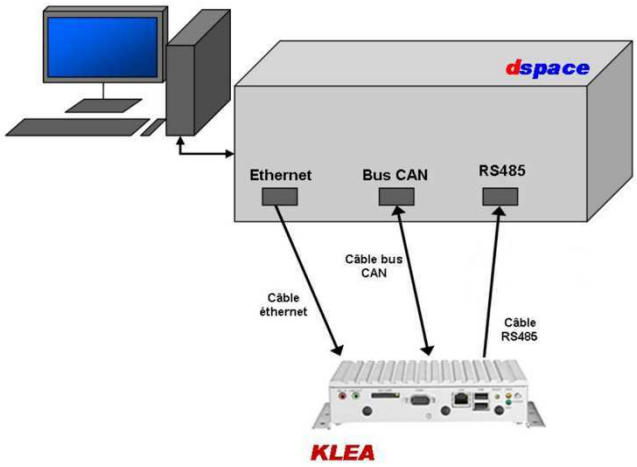


Figure V.21 : Configuration HIL de validation fonctionnelle du logiciel de vol

- Une configuration de validation du calculateur de vol, du codeur et de différents systèmes d'acquisition (tels que les Promux, des scrutateurs de tensions, …) (Figure V.22). Pour rappel, le codeur est le boitier d'acquisition de la majorité des capteurs du véhicule LEA, il est fourni par Zodiac DATASYSTEM sous le nom UMA 2000. Dans cette configuration le modèle de test a dû être adapté pour permettre au boitier dSPACE de piloter un ensemble électronique permettant la génération des signaux capteurs (près de 300) qui sont ensuite lus par le codeur qui les transmet ensuite par Ethernet au calculateur de vol. Un exemple de cette configuration telle que nous l'utilisons en laboratoire est représenté sur la Figure V.23.

La configuration photographiée sur la Figure V.23 diffère du schéma de la Figure V.22, du fait de la dynamique du projet et de l'indisponibilité de certains composants à l'heure actuelle (par exemple : 1 seul Promux au lieu des 12 prévus).

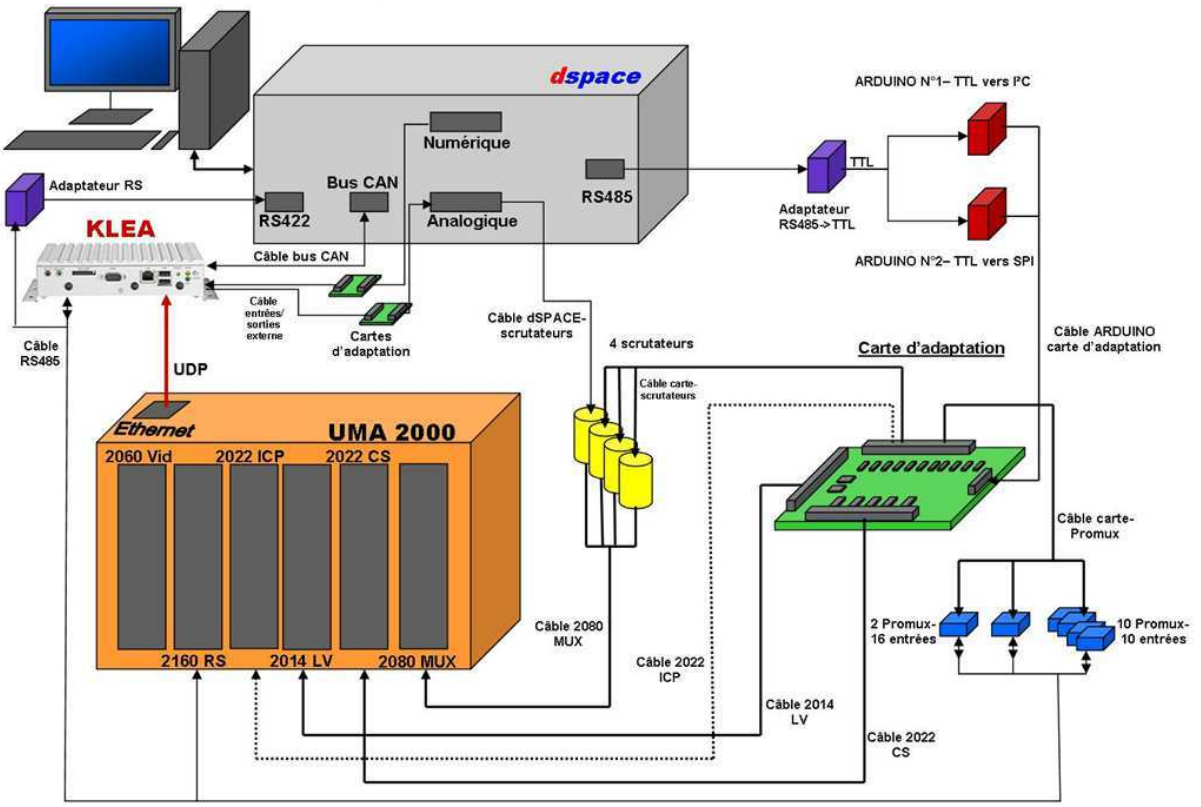


Figure V.22 : Configuration HIL de validation de la maquette fonctionnelle

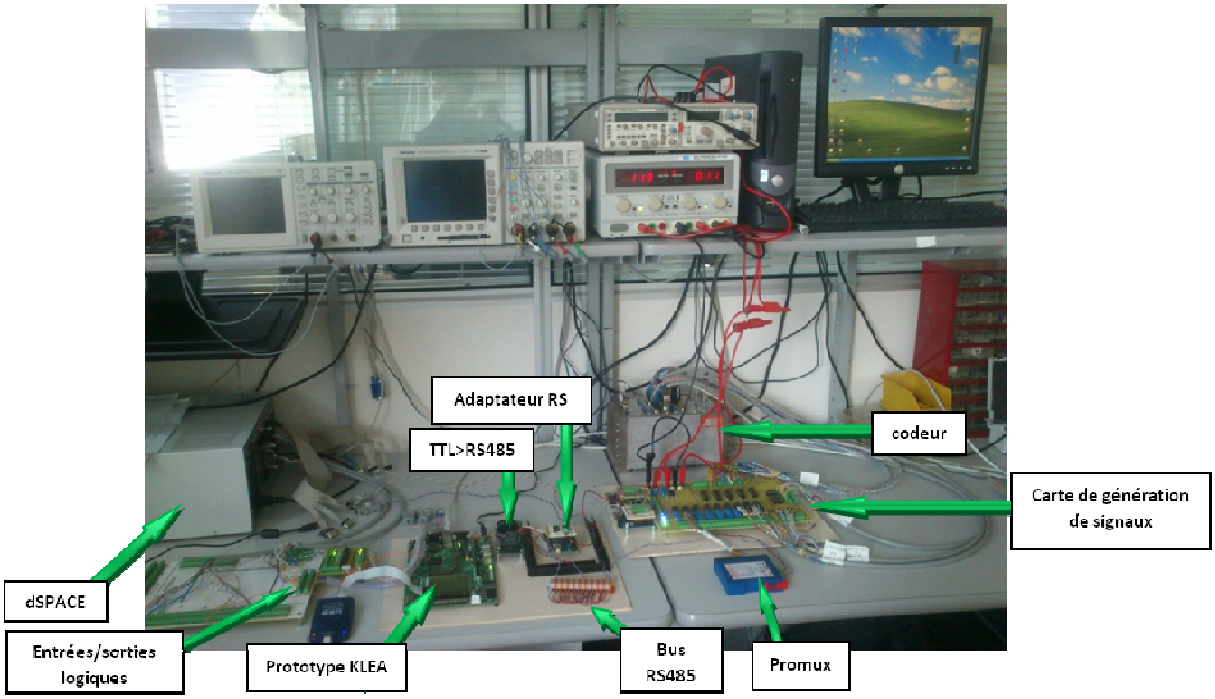


Figure V.23 : Photographie de la maquette fonctionnelle dans notre laboratoire

Finalement, avec le support de la technologie Mathworks et dSPACE, l'ensemble des phases de conception détaillée et de validation du logiciel de vol de LEA est réalisé selon le principe MIL/SIL/HIL qui réduit considérablement les redéveloppements de tests et de code source à la main, réduisant considérablement les risques d'erreur. Notons qu'il est alors possible de générer la défaillance d'un signal (i.e. d'un capteur) en entrée de l'ensemble de la maquette fonctionnelle (calculateur + codeur + système d'acquisition + …) et ainsi étudier la réponse du système embarqué à des fautes ainsi que l'efficacité des mécanismes de mitigation ou de compensation.

VI. Conclusion

Ce chapitre présente l'application de notre approche ISBM et de MéDISIS au projet LEA. La planification des activités de conception, de validation, d'études SdF et d'études spécifiques pilote l'organisation de ce chapitre où chaque activité principale est détaillée ainsi que l'apport de l'ISBM, de MéDISIS ou de la thèse pour sa réalisation. Nous y détaillons notamment la modélisation de l'analyse fonctionnelle en SysML et présentons un extrait de pré-AMDEC fonctionnelle. Nous voyons comment la méthode présentée au chapitre IV qui permet de réaliser une étude de fiabilité d'un composant grâce à l'association de la BCD et de FIDES reste vraie pour les COTS du projet LEA. Deux processus MéDISIS additionnels nés des besoins spécifiques du système embarqué LEA sont ensuite définis. Enfin, nous détaillons la méthode de conception et validation MIL/SIL/HIL et son application au projet LEA.

Les résultats des activités d'élicitation des besoins et d'analyse fonctionnelle présentés dans ce chapitre correspondent aux fonctions du véhicule LEA impliquées dans les phases d'essai en vol. Les règles de modélisation décrite dans le chapitre II sont respectées et illustrées pour la réification de ces fonctions, de leurs interactions et des phases de vie dans lesquelles elles interviennent. La pré-AMDEC fonctionnelle générée par notre outil est présentée. Elle permet de retrouver l'analyse des fonctions impliquées dans les phases d'essai en vol.

Nous illustrons ensuite un exemple de modélisation organique du système embarqué LEA et nous nous attardons sur l'étude d'un composant particulier : le Promux. Il s'agit d'un COTS électronique dont nous avons réalisé l'évaluation de fiabilité. En nous appuyant sur les travaux présentés dans le chapitre IV, nous avons détaillé les particularités de l'évaluation de fiabilité de cartes COTS dans la méthodologie FIDES. Cependant ces particularités n'empêchent pas l'utilisation de la BCD et de FIDES conjointement pour obtenir des résultats.

Parmi les activités planifiées au début de ce chapitre, deux d'entre elles n'étaient pas a priori facilitées par la méthodologie MéDISIS. Cependant au cours de la thèse, pour répondre aux besoins du projet LEA en termes d'analyse d'ordonnancement et d'injection de fautes, nous avons décrit deux processus de traduction que nous présentons ici : SysML vers AADL et SysML vers Simulink. L'intérêt de chacun de ces processus est illustré avec des cas pratiques issus du projet LEA.

Autour du modèle Simulink généré depuis le modèle système en SysML, nous organisons ensuite les activités de conception détaillée du logiciel de vol. La méthode d'ingénierie alors mise en place repose en grande partie sur les possibilités offertes par les outils Mathworks (Simulink, …) en termes de simulation et de génération de code et les possibilités offertes par la technologie dSPACE qui étend la simulation Simulink au domaine physique grâce à un simulateur temps réel.

Finalement, ce chapitre donne un large aperçu des travaux d'ingénierie réalisés par notre équipe au cours du projet LEA ainsi que les bénéfices apportés par nos travaux de recherches. Ce chapitre conclut la thèse et nous nous efforcerons dans la conclusion générale de montrer l'impact de nos travaux tant sur la méthodologie MéDISIS à court terme que sur les pratiques d'ingénierie en IS et en SdF à moyen et long terme.

Conclusion Générale

De la spécification à l'utilisation d'un système complexe, les contraintes s'appliquant au système sont multiples : fonctionnelles, technologiques, financières, normatives, … Il est alors précieux que l'ensemble des parties prenantes et des intervenants techniques puisse organiser l'ensemble des activités du projet et s'accorder sur les spécifications du système. L'approche d'ingénierie système à base de modèles adresse particulièrement ces problématiques. L'ISBM offre l'expressivité et la cohérence des informations ainsi que la possibilité de présenter un sous-ensemble de ces données sous la forme d'une vue adaptée à chaque interlocuteur. Ainsi un processus d'IS supporté par un langage de modélisation tel que SysML va permettre de centraliser l'ensemble des informations du projet qui doivent être régulièrement échangées et mises à jour. Les activités spécifiques à des domaines autres que l'IS doivent aussi pouvoir s'interfacer avec ce processus sans le ralentir, et sans l'être eux-mêmes. Assurer la cohésion de l'ISBM avec les études spécifiques et même envisager la valorisation de l'ISBM pour ces études constitue un des objectifs de notre équipe.

Une première réponse à cette problématique est la méthodologie MéDISIS qui a été présentée lors de précédents travaux de notre équipe [David 2009] (cf. Figure I.7). La méthodologie MéDISIS a pour objectif de valoriser l'ISBM pour deux types d'études de sûreté de fonctionnement : l'AMDEC composant et l'évaluation de scénarios de défaillance avec Altarica DF. MéDISIS vise à permettre, à partir d'un modèle SysML du système étudié, la mise à disposition de l'expert SdF, les données système qui lui sont utiles sous une forme proche de l'étude qu'il doit réaliser. MéDISIS construit un pont méthodologique entre deux domaines et entre leurs propres outils sans les altérer. En effet, MéDISIS ne propose pas une fusion de l'ISBM et de la SdF mais bien de valoriser les informations résultantes de l'ISBM pour la réalisation d'AMDEC et de modèle Altarica DF. Ces raccourcis offerts par la méthodologie MéDISIS permettent de fluidifier les échanges et ainsi la planification des activités d'IS et de SdF.

Dans cette thèse, nous nous sommes placés dans le contexte de la méthodologie MéDISIS : l'approche ISBM est adoptée, le langage de modélisation système est SysML et les activités d'IS et de SdF ne doivent pas être altérées, mais connectées. Le contexte industriel de la thèse : le projet LEA, apporte quant à lui l'objectif précis de nos travaux : appliquer les principes de MéDISIS aux études de sûreté de fonctionnement d'un système intégrant des COTS. Pour cela nous avons établi une base sémantique de SysML utilisée lors de notre processus d'IS sur laquelle nous appuyons nos travaux afin de permettre l'extension de la méthodologie MéDISIS à l'étude de système à COTS grâce à un processus de génération de pré-AMDEC fonctionnelle et à l'intégration de la méthodologie d'évaluation de fiabilité FIDES. Lors de l'application de nos travaux à la réalité industrielle du projet LEA, nous avons de plus défini deux autres processus MéDISIS, afin de répondre aux besoins d'ingénierie spécifiques du projet.

Après avoir présenté l'ensemble des concepts de l'ISBM, de MéDISIS, de SysML et les contraintes propres à l'intégration des COTS dans le chapitre I, nous avons dédié le chapitre II à la mise au point d'une sémantique de modélisation des résultats d'IS avec SysML pour la réification des résultats de toutes les activités d'IS. En effet, SysML est présenté comme le langage de l'ISBM, cependant certaines étapes de l'IS comme l'élicitation des besoins et l'analyse fonctionnelle ne sont pas toujours modélisées de la même façon en SysML. C'est pourquoi il était nécessaire de statuer

sur les règles de modélisation qui structurent les informations, leur nature et leur sens en SysML, afin d'y accéder correctement par l'intermédiaire de nos processus MéDISIS.

Dès le chapitre III, nous faisons appel au modèle système résultant des activités d'IS afin de mettre au point un processus d'extraction des données issues de l'analyse fonctionnelle. Ces informations et les structures d'interconnexion des entités modélisées sont alors organisées dans un tableau de pré-AMDEC fonctionnelle suivant une organisation classique. Dès lors, la pré-AMDEC nécessitera d'être complétée par un expert en SdF qui transformera alors la pré-AMDEC fonctionnelle en AMDEC fonctionnelle, principalement en ajoutant les données dysfonctionnelles qui ne sont pas modélisées dans le modèle système. Dans le but d'optimiser l'utilisation de la pré-AMDEC dans un projet où les évolutions du système sont régulières, nous avons aussi décrit la Base de données des Dysfonctionnements des Fonctions qui maintient itération après itération les apports de l'expert SdF.

Le principe de la BDF provient de la Base de données des Comportements Dysfonctionnels qui faisait partie intégrante de la méthodologie MéDISIS lors de sa première définition dans [David 2009]. La BCD est définie pour pérenniser les données dysfonctionnelles des composants (contrairement à la BDF qui stocke des données concernant les fonctions). Ainsi dans le Chapitre IV, nous définissons la BCD dans sa forme la plus actuelle. Elle se constitue de 3 parties principales :

- Les mécanismes de déclenchement de la défaillance (aléatoire ou déterministe).
- Les effets de la défaillance sur le fonctionnement normal du composant.
- La logique régissant les comportements opérationnels et défaillants.

Afin d'étendre la portée de la BCD à l'étude de systèmes intégrants des COTS, nous avons montré qu'elle pouvait intégrer les principes d'évaluation de fiabilité de FIDES par l'intermédiaire de *Parametric Diagrams* qui modélisent l'ensemble des équations mathématiques de FIDES. À l'inverse, nous avons aussi montré qu'une étude de fiabilité FIDES profite elle aussi de l'approche ISBM et de l'utilisation de MéDISIS. En particulier, les diagrammes de contexte qui définissent le cycle de vie du système, permettent de renseigner un grand nombre de paramètres utiles à FIDES.

Les ajouts à MéDISIS réalisés dans la thèse ont pour but de permettre l'étude de systèmes complexes intégrant des COTS afin d'être appliqués au projet LEA. Dans le chapitre V, nous présentons donc l'application de MéDISIS à l'étude du système embarqué du véhicule hypersonique LEA qui contient de nombreux COTS. Cependant, nous voyons dans ce chapitre que l'AMDEC fonctionnelle et l'étude de fiabilité ne sont pas les seules activités hors IS nécessaires à la validation de notre système. C'est pourquoi, afin de valoriser les efforts de modélisation réalisés sur le projet LEA, nous avons défini deux autres processus MéDISIS permettant l'analyse d'ordonnancement des tâches à l'aide d'un modèle AADL et la simulation du logiciel de vol et l'injection de faute dans un modèle Simulink.

Ainsi au terme de la thèse, la méthodologie MéDISIS s'est considérablement étendue par rapport à sa première définition, en maintenant ses objectifs principaux de valorisation de l'approche ISBM. La Figure CG.1 présente l'ensemble des processus de traduction et des mécanismes de pérennisation du retour d'expérience offerts par la méthodologie MéDISIS. Les activités pouvant être réalisées avec l'aide de la méthodologie MéDISIS sont à présent au nombre de 5 et permettent l'étude d'un système dès les phases d'analyse fonctionnelle et jusqu'aux phases de conception

détaillée. De plus, les meilleurs outils actuels en termes d'intégration de COTS peuvent à présent être utilisés dans le cadre méthodologique MéDISIS.

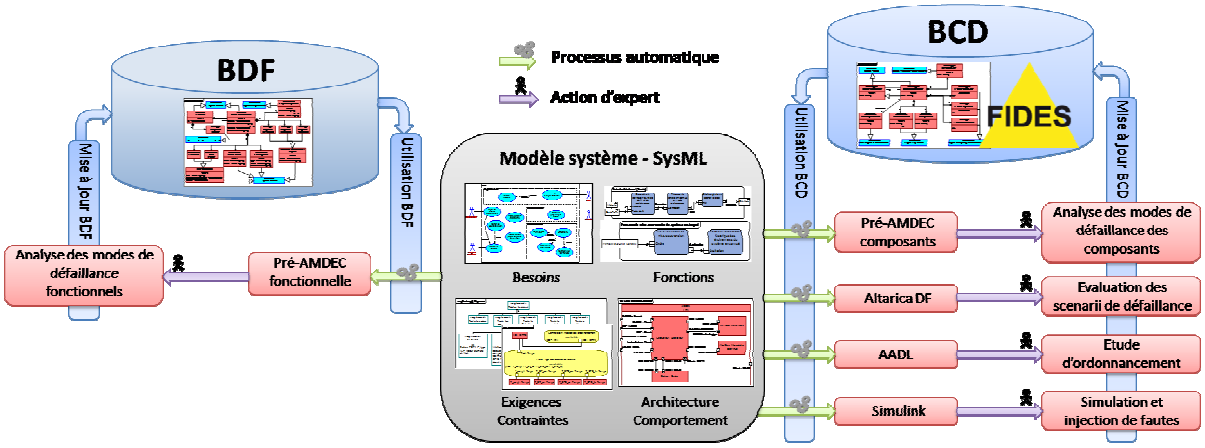


Figure CG.1 : Méthodologie MéDISIS à l'issue des travaux présentés dans la thèse

Les perspectives qu'offre la méthodologie MéDISIS dans son état actuel sont multiples. Le projet LEA est un projet de taille considérable. Au sein de notre équipe, ont participé à ce projet : 1 doctorant pendant 36 mois, 1 ingénieur d'étude pendant 18 mois, et 7 stagiaires de 6 mois et il n'a pas encore abouti puisque les premiers essais en vol sont prévus en 2014. C'est pourquoi les données de retour d'expérience à considérée sont nombreuses et certaines ne sont pas encore disponibles. À l'heure actuelle, une analyse fine en vue de la quantification des bénéfices apportés par la méthodologie MéDISIS est donc impossible. Au terme du projet, et avec la participation de MBDA qui dirige régulièrement des projets similaires au projet LEA, les résultats apportés par l'ISBM et MéDISIS sur le projet LEA et les résultats de méthodes plus classiques sur d'autres projets pourront être comparés afin de quantifier les éventuels bénéfices de chaque méthode. Une étude fine permettrait aussi de mettre en avant les aspects que la méthode MéDISIS à pour objectif d'améliorer : la cohésion des informations entre les différents intervenants et la rapidité de détection des risques projets.

Outre la quantification des bénéfices de MéDISIS, la méthodologie elle-même peut être optimisée. Sur la Figure CG.1, nous observons que la BDF et la BCD sont deux entités distinctes avec des méta-modèles différents. Elles n'ont pas le même périmètre d'utilisation puisque la BDF se concentre sur le maintien d'informations au sein d'un projet quand la BCD vise la création d'une base de données de fiabilité croissante au fur et à mesure des projets. L'union de la BCD et de la BCF en une librairie de composants, de fonctions et de leurs modes de défaillances permettrait d'assurer une cohérence de tous les aspects d'un projet (fonctionnel, organique et dysfonctionnel). Le méta-modèle de cette librairie pourra de plus s'associer à des concepts du langage MARTE afin d'intégrer la description de contraintes temps réelles comme nous avons commencé à le faire avec le langage AADL. Enfin, cette librairie pourra s'approcher des librairies d'entités réutilisables propres à l'ISBM qui propose au modélisateur système une bibliothèque de modèles prêts à l'emploi. L'ingénieur système saura dès la modélisation, lors du choix des entités de la librairie, s'il pourra réaliser des études spécifiques automatiques, selon la présence ou non de données dysfonctionnelles ou temporelles pour les entités sélectionnées.

Un autre axe d'amélioration de la méthodologie MéDISIS actuelle serait le raffinement des modèles et documents générés. Par exemple, une analyse plus fine des modèles SysML permettrait de

réduire la taille des AMDEC générées en se concentrant sur les causes et effets impliqués par les modes de défaillances considérés. Une méthode envisagée pour réaliser cela, est de mettre au point une politique d'étude des flux modélisés en SysML afin de déduire du modèle SysML les causes qui pourraient provoquer un mode de défaillance spécifique et les effets qui doivent logiquement en résulter, plutôt que de lister l'ensemble des possibles comme c'est le cas actuellement.

En parallèle des optimisations possibles de MéDISIS, la méthodologie pourra aussi être étendue, notamment dans le cadre d'un projet industriel présentant de nouvelles contraintes. La méthodologie MéDISIS a été initiée lors du projet CAPTHOM (CAPTeur de présence HUMaine) qui consistait à mettre au point un capteur de présence intelligent à des fins domotiques. MéDISIS s'est ensuite étendue dans le cadre du projet LEA et de ses contraintes de système embarqué et de temps réel. Un futur projet industriel dans un domaine différent permettrait :

- de fournir un retour d'expérience supplémentaire sur l'application de la méthode,
- d'envisager la mise au point de nouveau processus spécifique qui étendrait la méthodologie MéDISIS,
- de compléter la BCD actuelle avec des composants propres au domaine industriel du nouveau projet.

Finalement, c'est aussi au-delà de la méthodologie MéDISIS que ces travaux offrent des perspectives. La BCD et les algorithmes d'analyse des modèles SysML pourraient être couplés à des approches de sûreté de fonctionnement à base de modèles qui commencent à émerger. À l'heure actuelle, ces approches ont rarement recours à SysML puisque ce n'est pas un langage qui encourage nativement la modélisation des comportements dysfonctionnels. Cependant, nous avons vu au travers de l'ensemble des travaux de notre équipe que SysML possède les moyens de modélisation des comportements dysfonctionnels et que l'analyse de modèle est possible méthodologiquement et techniquement.

Glossaire

AADL : Architecture Analysis & Design Language

AD : Activity Diagram/Diagramme d'Activité

AdD : Arbre de Défaillance

AFIS : Association Française d'Ingénierie Système

AltaRica DF : AltaRica Data Flow

AMDEC : Analyse des Modes de Défaillance de leurs Effets et Criticité

APR : Analyse Préliminaire des Risques

BCD : Base de données des Comportements Dysfonctionnels

BDD : Block Definition Diagram / Diagramme de définition de blocs

BDF : Base de données des Dysfonctionnements des Fonctions

COTS : Component Off The Shelf

DSL : Domain Specific Language

EIA : Electronic Industries Alliance

IBD : Internal Block Diagram / Diagramme interne de bloc

IEC : International Electrotechnical Commision

IEEE : International Electronic and Electrical Engineers

INCOSE : INternational Council On System Engineering

IS : Ingénierie Système

ISBM : Ingénierie Système à Base de Modèles

MBSE : Model Based System Engineering

MdD : Mode de Défaillance

MéDISIS : Méthode D'Intégration des analyses de Sûreté de fonctionnement au processus d'Ingénierie Système

OMG : Object Management Group

OOSEM : Object-Oriented System Engineering Method

ParD : Parametric Diagram / Diagramme paramétrique

ReqD : Requirement Diagram/Diagramme d'exigences

RdP : Réseau de Petri

RdPSG : Réseau de Petri Stochastique Généralisé

REX : Retour d'Expérience

RPN : Risk Priority Number

SD : Sequence Diagram/Diagramme de Séquence

SdF : Sûreté de Fonctionnement

STM : State Machine Diagram / Diagramme d'état

SysML : System Modeling Language

UCD : Use Case Diagram / Diagramme de cas d'utilisation

UML : Unified Modeling Language

XML : eXtensible Markup Lamguage

XMI : XML Metadata Interchange

Liste des publications de l'auteur

Revues internationales :

1. Cressent, R., David, P., Idasiak, V. & Kratz, F. Designing the database for a reliability aware Model-Based System Engineering process. *Journal of Reliability Engineering and System Safety*, Elsevier. A paraître, doi:10.1016/j.ress.2012.10.014.

Congrès internationaux :

1. Cressent, R., David, P., Idasiak, V. & Kratz, F. Increasing Reliability of Embedded Systems in a SysML Centered MBSE Process: Application to the LEA Project. *Proceedings of the 1st M-BED workshop, during DATE 2010*, Dresden, Germany, 12 March 2010.
2. Cressent, R., David, P., Idasiak, V. & Kratz, F. 2011. Mastering safety and reliability in a Model Based process. *Proceedings of the 57th Annual Reliability and Maintainability Symposium, RAMS2011,* Orlando, Florida, USA, 24-27 January 2011.
3. Cressent, R., David, P., Idasiak, V. & Kratz, F. 2011. Dependability analysis activities merged with system engineering, a real case study feedback. *ESREL 2011*, Troyes, France, 18 au 22 septembre 2011.
4. Cressent, R., Idasiak, V. & Kratz, F. 2012. Reusing FIDES knowledge in the MéDISIS Dysfunctional Behavior Database. *PSAM 11 & ESREL 2012*, Helsinski, Finlande, 25 au 29 juin 2012

Revues nationales :

1. Cressent, R., David, P., Idasiak, V. & Kratz, F. Prise en compte des analyses de sûreté de fonctionnement dans l'ingénierie de système dirigée par les modèles SysML. *Revue Génie Logiciel n° 96, GL & IS*, mars 2011, p33 à 39.

Congrès nationaux :

1. Cressent R., David P., Idasiak V., Kratz F., Apports de SysML à la modélisation des systèmes complexes à fortes contraintes de sureté de fonctionnement. *Recueil d'article de la conférence ITT'09 (Technological Innovation and Transport Systems),* Paris, 2009.
2. Cressent, R., Idasiak, V. & Kratz, F. Rapprocher les études de sûreté de fonctionnement de l'ingénierie système : retour d'expérience. *Recueil d'articles de la 9ème édition du congrès international QUALITA 2011*, Angers, 23 au 25 mars 2011.
3. Cressent, R., Idasiak, V. & Kratz, F. 2011. Intégration des analyses de sûreté de fonctionnement dans le processus de conception système. *4èmes Journées Doctorales / Journées Nationales MACS*, Marseille, 9 et 10 juin 2011.

Autres communications :

1. Cressent, R., David, P., Idasiak, V. & Kratz, F. L'Ingénierie Système Basée sur les Modèles SysML pour les systèmes sûrs de fonctionnement. *Recueil d'articles du Forum AFIS Académie / Entreprise*, Bordeaux, 2-3 décembre 2010
2. Cressent, R., David, P., Idasiak, V. & Kratz, F. Model Based System Engineering with SysML for reliable systems design. *INSIGHT 4th Qtr 2011*, INCOSE.

Références

[Académie Française 1935] : Dictionnaire de l'Académie Française (huitième édition), 1932-1935.

[AFIS 2001] : Groupement d'auteurs du GT Ingénierie des Exigences de l'AFIS : G. Fanmuy, G. Levy, J. Foisseau, P. Lamothe, B. Hermans, P. De Chazelles, E. Choveau. Recommandations pour l'élaboration d'un référentiel d'exigences techniques. 2001

[AFIS 2009]: Document collectif AFIS, Serge Fiorèse, J-P Meinadier, Découvrir et comprendre l'ingénierie système (Version V3), 2009.

[Alur et al 1995] : R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.H. Ho, X. Nocollin, A. Olivero, J. Sifakis, S. Yovine, «The algorithmic analysis of hybrid systems», Theoretical Computer science, n°138, 1995.

[Arlat et al. 2000] : J. Arlat, J-P. Blanquart, T. Boyer, Y. Crouzet, M-H. Durand, J-C. Fabre, M. Founeau, M. Kaâniche, K. Kanoun, P. Le Meur, C. Mazet, D. Powell, F. Scheerens, P. Thévenod-Fosse, H. Waeselynck. Composants logiciels et sûreté de fonctionnement, integration de COTS. Hermes Science Publications. 2000.

[ATEGO] : Atego est l'éditeur du logiciel : Artisan Studio. Il s'agit d'un logiciel de modélisation UML et SysML, proposant aussi des outils de génération de code C depuis des modèles UML et SysML. http://www.atego.com/products/artisan-studio/

[AV] : Normes NF X 50-100, NF X 50-150, NF X 50-151, NF X 50-152 et NF X 50-153 : Analyse fonctionnelle, Analyse de la valeur, Management de la valeur. AFNOR. Aout 1990 à septembre 2007.

[Bernardi et al. 2002]: S. Bernardi, S. Donatelli & J. Merseguer. From UML Sequence Diagrams and StateCharts to analyzable Petri Net models. In Proceedings of the 3rd Int. workshop on software on performance, Rome, Italy, 2002.

[Bishop et al. 2003] : P. Bishop, R. Bloomfield, T. Clement, S. Guerra. Software criticality analysis of COTS/SOUP. Reliability Engineering & System Safety, vol. 81, pp. 291-301, 2003.

[Boiteau et al. 2006] : M. Boiteau, Y. Dutuit, A. Rauzy & J.-P. Signoret. The AltaRica dataflow language in use : modeling of production availability of a multi-state system. Reliability Engineering & System Safety, Vol. 91, pp. 747 755, 2006.

[Bondavalli et al. 1999a]: A. Bondavalli, M. Dal Cin, D. Latella & A. Pataricza. High-level Integrated Design Environment for dependability (HIDE). Proceedings of the 5th International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS-99), pp.87-92, 1999.

[Bondavalli et al. 1999b]: A. Bondavalli, I. Majzik & I. Mura. Automated Dependability Analysis for Supporting Design Decisions in UML. Proceedings of the 4th IEEE High Assurance System Engineering Symposium, Washington, USA, pp.64-71, 1999.

[Bondavalli et al. 2001]: A. Bondavalli, M. Dal Cin, D. Latella, I. Majzik, A. Pataricza & G. Savoia. Dependability analysis in the early phases of UML-based system design. International Journal of Computer Systems Science & Engineering, Vol. 16 n°5, pp. 265-275, septembre 2001.

[Bowles & Pelaez 1995]: J.B. Bowles & C.E. Pelaez. Fuzzy logic priorization of failures in a system failure mode, effects and criticality analysis. Reliability Engineering and System Safety, Vol. 50, pp. 203-213, 1995.

[Bretesche 2000] : Bertrand de la Bretesche, *La méthode APTE : Analyse de la valeur, analyse fonctionnelle*, Pétrelle, 2000 (ISBN 978-2-84440-019-2)

[BS 5760-5]: British Standard. Reliability of systems, equipment and components. Guide to failure modes, effects and criticality analysis (FMEA and FMECA). Décembre 1991.

[Cantor 2003] : M. Cantor. Rational Unified Process for Systems Engineering, RUP SE Version 2.0. IBM Rational Software white paper, IBM Corporation, 8 mai 2003.

[Charpenel et al. 2003] : Charpenel P, Davenel F, Digout R, Giraudeau M, Glade M, Guerveno JP, Guillet N, Lauriac A, Male S, Manteigas D, Meister R, Moreau E, Perie D, Relmy-Madinska F, Retailleau P. The right way to assess electronic system reliability: FIDES. Microelectronics Reliability, Volume 43, Issues 9–11, Pages 1401–1404, September–November 2003.

[Chiol et al 1993] : G. Chiol, M. A Marsan, G. Balbo, G.Conte, «Generalized Stochastic Petri Nets: A Definition at the Net Level and its Implications», IEEE Transactions on Software Engineering, Vol 19, Issue 2, p. 89- 107, February 1993.

[Cressent et al. 2010]: Cressent R, David P, Idasiak V & Kratz F. Increasing Reliability of Embedded Systems in a SysML Centered MBSE Process: Application to the LEA Project. 1st M-BED workshop, during DATE 2010, Dresden, Germany, 12 March 2010.

[Cressent et al. 2011a]: Cressent R, David P, Idasiak V & Kratz F. Mastering safety and reliability in a Model Based process. Proceedings of the 57th Annual Reliability and Maintainability Symposium, RAMS2011, Orlando, Florida, USA, 24-27 January 2011.

[Cressent et al. 2011b]: Cressent, R., David, P., Idasiak, V. & Kratz, F. Prise en compte des analyses de sûreté de fonctionnement dans l'ingénierie de système dirigée par les modèles SysML. Revue Génie Logiciel n° 96, GL & IS, mars 2011, p33 à 39.

[Cressent et al. 2011c]: Cressent R, David P, Idasiak V, Kratz F. Dependability analysis activities merged with system engineering, a real case study feedback. ESREL 2011, Troyes, France, 18 au 22 septembre 2011.

[Cressent et al. 2012a]:Cressent R, Idasiak V, Kratz F. Reusing FIDES knowledge in the MéDISIS Dysfunctional Behavior Database. ESREL 2012, Helsinki, Finlande, 25 au 29 juin 2012.

[Cressent et al. 2012b]: Cressent R, David P, Idasiak V, Kratz F. Designing the database for a reliability aware Model-Based System Engineering process. Reliability Engineering and System Safety – Elsevier. A paraître, doi:10.1016/j.ress.2012.10.014.

[Chomsky 1956]: Three models for the description of language, Noam Chomsky, Department of Modern Languages and Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1956.

[David 2009]: P. DAVID, « Contribution à l'analyse de sureté de fonctionnement des systèmes complexes en phase de conception : application à l'évaluation des missions d'un réseau de capteurs de présence », Universités d'Orléans, ENSI de Bourges, 2009.

[David et al. 2009a]: David P, Idasiak V, Kratz F. Improving Reliability Studies with SysML. Proceedings of the 55th Annual Reliability and Maintainability Symposium, RAMS 2009, Fort Worth, Texas, USA, Jan. 2009.

[David et al. 2009b]: David P, Idasiak V, Kratz F. Automating the synthesis of AltaRica Data-Flow models from SysML. Proceedings of ESREL 2009, Prague, République Tchèque, 7-10 septembre 2009.

[David et al. 2010]: David P, Idasiak V, Kratz F. Reliability study of complex physical systems using SysML. Reliability Engineering and System Safety – Elsevier, Vol. 95 (april), pp. 431-450, 2010.

[Desroches 2010]: V. DESROCHES. Sûreté de fonctionnement des systèmes de systèmes : une discipline… interdisciplinaire. *17e Congrès Lambda Mu, La Rochelle, France, 5-7 octobre 2010.*

[DOORS] : Rational DOORS. Outil de gestion d'exigences. http://www-142.ibm.com/software/products/fr/fr/ratidoor/

[Douglass 2005]: B. Douglass. The Harmony Process. I-Logix white paper, I-Logix, Inc., 25 mars 2005.

[Dugan et al 1984] : J.B. Dugan, K.S. Trivedi, R.M. Geist, V.F. Nicola, «Extende Stochastic Petri Nets: Applications and Analysis», 10th International Symposium on Computer Performance, p. 507-519, 1984.

[EIA 632]: Electronic Industries Alliance. Processes for Engineering a System. 1999

[Estefan 2008]: J. Estefan. Survey of Model-Based Systems Engineering (MBSE) Methodologies, Rev. B. INCOSE MBSE Initiative, 23 Mai 2008

[Feiler & Rugina 2007]: Feiler, P. & Rugina A. "Dependability Modeling with the Architecture Analysis & Design Language (AADL)", Carnegie Mellon Institute, CMU/SEI-2007-TN-043, July 2007.

[FIDES 2004]: FIDES Group. Reliability Methodology for electronic systems – FIDES Guide 2004 Edition A, UTE C 80811. 2004

[FIDES 2011]: FIDES Group. Reliability Methodology for electronic systems – FIDES Guide 2009, UTE 80811 Janvier 2011.

[Friedenthal et al. 2007]: S. Friedenthal, R. Griego & M. Sampson. INCOSE MBSE Initiative. International Council on Systems Engineering 2007, San Diego, 24-29 juin 2007

[Friedenthal et al. 2008]: S. Friedenthal, A. Moore & R. Steiner. A Practical Guide to SysML: The Systems Modeling Language. The MK/OMG press, Elsevier, 2008

[Gireaudeau et al. 2010]: M. Giraudeau, S. Depienne, F. Bayle. Exploitation de données de retours d'expérience multi-industriels pour la consolidation des modèles FIDES. 17e Congrès Lambda Mu, La Rochelle, France, 5-7 octobre 2010.

[Guillerm 2011] : R. Guillerm. Intégration de la Sûreté de Fonctionnement dans les Processus d'Ingénierie Système. Thèse de doctorat de l'Université Toulouse III – Paul Sabatier, soutenue le 15 juin 2011.

[Guiochet 2003]: J. Guiochet. Maîtrise de la sécurité des systèmes de la robotique de service, approche UML basée sur une analyse du risque système. Thèse de doctorat de l'Institut National des Sciences Appliquées de Toulouse, soutenue le 9 juillet 2003.

[Harel 1987] : D. Harel, «Statecharts: A visual Formalism for complex systems », The Science of Computer Programming, p.231-274, 1987.

[Hause 2006]: M. Hause. The SysML Modelling Language. Proceedings of the 5th European Systems Engineering Conference, Edimbourg, 18-20 septembre 2006.

[Hecht et al. 2004] : H. Hecht, X. An & M. Hecht. Computer-aided Software FMEA. The Annual Reliability and Maintainability Symposium, Los Angeles, 26-29 janvier, 2004.

[Hoffmann 2006]: H-P Hoffmann. Harmony-SE/SysML Deskbook: Model-Based Systems Engineering with Rhapsody. Telelogic/I-Logix white paper, Telelogic AB, 24 mai 2006.

[Holt & Perry 2008] : J. Holt et S. Perry. SysML for Systems Engineering. Publié par The Institution of Engineering and Technology, Londres, Royaume-Uni. 2008.

[IDE 2000]: Interim Defence Standard 00-58, Hazop studies on systems containing programmable electronics. Part 1: requirements. Part 2: general application guidance. Issue 2 MoD 2000

[IEC 60812]: International Electrotechnical Commission, Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA). Janvier 2006.

[IEC 61508]: IEC 61508. International Electrotechnical Commission. Functional Safety of Electrical /Electronic /Programmable Electronic Safety-Related Systems. Parts 1 to 7. 1998-2005.

[IEC 61511]: IEC 61511. International Electrotechnical Commission, "Functional Safety: Safety Instrumented Systems for the Process Sector," Geneva, Switzerland (expected 2003).

[IEEE 1220]: Institute for Electrical and Electronic Engineers. IEEE standard for Application and Management of the Systems Engineering Process. 8 décembre 1998

[INCOSE 2007] : International Council on Systems Engineering. Systems Engineering Vision 2020. Version 2.03, TP-2004-004-02, Septembre 2007

[INCOSE 2010]: International Council on Systems Engineering. Systems Engineering Handbook. Version 3.2, 2010

[Ishaque 2011]: K. Ishaque, Z. Salam, Syafaruddin. A comprehensive MATLAB Simulink PV system simulator with partial shading capability based on two-diode model. Solar Energy, Volume 85, Issue 9, September 2011, Pages 2217–2227.

[ISO 15288]: International Electrotechnical Commission. Systems Engineering – System Life Cycles Processes. 1 February 2008

[Klein et al. 1993]: Klein, M., Ralya, T., Pollak, B., Obenza, R., Harbour, M. & Harbour G. "A practitioner's Handbook for Real-Time Analysis". *Kluwer Academic Publishers*, 1993.

[Kruchten 2003]: P. Kruchten. The Rational Unified Process: An Introduction, Third Edition. Addison-Wesley Professional : Reading, MA, 2003.

[Laprie et al 1996]: J.C. Laprie, J. Arlat, J-P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J-C. Fabre, H. Guillermain, M. Kaaniche, C. Mazet, D.Powell, C. Rabéjac et P. Thévenod., «Guide de la Sûreté de Fonctionnement », 2ème édition (Cépaduès), ISBN : 2.85428.382.1,1996.

[Laprie 2004]: Laprie. J-C, "Sûreté de fonctionnement des systèmes : concepts de base et terminologie", Revue de l'Électricité et de l'Électronique (REE), (11), pp.95-105, novembre 2004.

[Lasnier 2009]: G. Lasnier, B. Zalila, L. Pautet, and J. Hugues. OCARINA: An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications. In *Reliable Software Technologies'09 - Ada Europe*, Brest, France, juin 2009.

[Long 2002] : J. Long. Relationships between common graphical representations in Systems Engineering. In 5th International Symposium of the INCOSE, St. Louis, MO, juil. 1995. Updated July 2002.

[Lopez-Grao et al. 2004] : J-P. López-Grao, J. Merseguer & J. Campos. From UML Activity Diagrams to Stochastic Petri Nets: Application to software performance engineering. Procee-dings of the 4th Int. workshop on software and performance, Redwood shores, USA, 2004.

[Lykins et al. 2000] : H. Lykins, S. Friedenthal, A. Meilich. Adapting UML for an Object-Oriented systems engineering method (OOSEM). In: Proceedings of the 10th annual INCOSE symposium, Juillet 2000.

[Magee et al. 2004]: Magee C. and De Weck O. L., Complex System Classification, Fourteenth Annual International Symposium of the International Council on Systems Engineering (INCOSE), Toulouse, France, June 20-24, 2004.

[MagicDraw]: Magic Draw de NO MAGIC. Outil de modélisation SysML/UML. http://www.nomagic.com/

[MathWorks]: MathWorks est la société produisant les logiciels Simulink et Matlab. www.mathworks.fr.

[Marsan et Chiola 1986] : M.A. Marsan, G. Chiola, «On Petri Nets with Deterministic and Exponentially Distributed Firing Times», 7th European Workshop on Applications and Theory of Petri Nets, p. 132-145, Springer-Verlag, Lecture Notes in Computer Science 266, Advances in Petri Nets, 1987, June 1986.

[Merseguer et al. 2002] : J. Merseguer, J. Campos, S. Bernardi & S. Donatelli. A Compositional Semantics for UML State Machines Aimed at Performance Evaluation. In Proceedings of the 6th Int. Workshop on Discrete Event Systems (WODES'02), 2002.

[MIL-HDBK-217F] : USA Department of Defense. Reliability Prediction of Electronic Equipment. Military Standard, MIL HDBK 217f. 1991.

[MIL-STD1629A] : USA Department of Defense. Procedures for performing a Failure Mode, Effects and Criticality Analysis. Military Standard, MIL-STD1629A. 1980.

[Molloy 1982] : M.K. Molloy «Performance analysis using stochastic Petri nets», IEEE Trans on Computer vol C-31, N°9, p.913-917, Sept 1982.

[Moncelet 1998]: G. Moncelet, «Application des réseaux de Petri à l'évaluation de la sûreté de fonctionnement des systèmes mécatroniques du monde automobile», Thèse de Doctorat, N°3076, Université Paul Sabatier, Toulouse, 9 octobre 1998.

[NASA 2007] : National Aeronautics and Space Administration. Systems Engineering Handbook. NASA/SP-2007-6105 Rev1. Décembre 2007.

[Natkin 1980] : S.O. Natkin, «Les réseaux de Petri stochastiques et leur application à l'évaluation des systèmes informatiques», thèse de Docteur Ingénieur, CNAM Paris, juin 1980.

[OMG 2007]: Object Management Group, Systems Modeling Language V1.0, Septembre 2007

[OMG 2010]: Object Management Group, Systems Modeling Language V1.2, Juin 2010

[OMG 2011a]: Object Management Group. Unified Modeling Language. OMG Specification – UML 2.4 Superstructure & UML 2.4 Infrastructure, aout 2011.

[OMG 2011b]: Object Management Group. MOF 2 XMI Mapping (XMI), v 2.4.1, aout 2011.

[OMG 2012]: Object Management Group, Systems Modeling Language V1.3, Juin 2012

[Pai & Dugan 2002] : G. Pai & J. Dugan. Automatic Synthesis of Dynamic Fault Trees from UML System Models. Proceedings of the 13th International Symposium on Software Reliability Engineering, Annapolis, USA, pp. 243-254, 2002.

[Picardi et al. 2004] : C. Picardi, L. Console, F. Berger, J. Breeman, T. Kanakis, J. Moe-lands, S. Collas, E. Arbaretier, N. De Domenico, E. Girardelli, O. Dressler, P. Struss & B. Zilbermann. AUTAS: a tool for supporting FMECA generation in aeronautic systems. 16th European Conference on Artificial Intelligence, ECAI 2004, Valence, Espagne, 22-27 août, 2004.

[Pillay & Wang 2003]: A. Pillay & J. Wang. Modified failure mode and effects analysis using approximate reasoning. Reliability Engineering & System Safety, vol. 79, pp. 69-85, 2003.

[Point 2000]: G. Point. AltaRica : Contribution à l'unification des methods formelles et de la sûreté de fonctionnement. Thèse de doctorat soutenue le 20 janvier 2000 à l'Univerité de Bordeaux I, 2000.

[Price 1996]: C. Price. Effortless Incremental Design FMEA. The Annual Reliability and Maintainability Symposium, Las Vegas, Janvier, 1996.

[Rauzy & Dutuit 1997]: A. Rauzy, Y. Dutuit, «Exact and truncated computations of prime implicants of coherent and non coherent fault trees within Aralia», Reliability Engineering & System Safety, Vol. 58, 1997.

[Rauzy et al. 2008] : A. Rauzy, Z. Brik & E. Arbaretier. Sûreté de Fonctionnement et Analyse de Performance. 16ème Congrès Lambda Mu : les nouveaux défis de la maîtrise des risques, Avignon, France, 7-9 octobre 2008.

[Redmill 2004] : REDMILL, « Analysis of the COTS debate », Safety Science 42 (2004), p355-367.

[Reqtify] : Reqtify. Outil de gestion d'exigences. http://www.geensoft.com/fr/article/reqtify/

[Rhapsody]: Rational Rhapsody d'IBM. Outil de modélisation SysML/UML. http://www-01.ibm.com/software/awdtools/rhapsody/

[Ridoux 1999] : M. Ridoux. AMDEC- Moyen. Techniques de l'Ingénieur, AG 4220, 10 juillet 1999.

[Rolland 1999] : Rolland C., (1999), Requirements Engineering for COTS based Systems**,** International Journal of Information and Software Technology 41, 14 985 - 990

[Roques 2009a] P Roques. UML par la pratique, 5ème édition. Edition Eyrolles. Mars 2009.

[Roques 2009b] P. Roques. SysML par l'exemple - Un langage de modélisation pour systèmes complexes. Ebook Edition Eyrolles. Mai 2009

[SAE 2004]: SAE, Society of Automotive Engineers. SAE Standards: AS5506, SAE Architecture Analysis & Design Language (AADL), November 2004.

[SAE 2006]: SAE, Society of Automotive Engineers. SAE Standards: AS5506/1, SAE Architecture Analysis and Design Language (AADL) *(Annex Volume 1: Annex A: Graphical AADL Notation, Annex C: AADL Meta-Model and Interchange Formats, Annex D: Language Compliance and Application Program Interface Annex E: Error Model Annex)*, June 2006

[SAE 2009]: SAE, Society of Automotive Engineers. SAE Standards: AS5506A, SAE Architecture Analysis and Design Language (AADL), Specification V2, January 2009.

[Seidner 2009] Seidner Charlotte. Vérification des EFFBDs : Model checking en Ingénierie Système. Institut de Recherche en Communications et en Cybernétique de Nantes (IRCCyN). 2009.

[Singhoff 2007]: Singhoff F., February 2007. *The Cheddar AADL Property sets (Release 2.x)*. LISyC technical report number singhoff-03-07

[Sinnanmon et Andrew 1997]: R.M. Sinnamon & J.D. Andrews, «New approaches to evaluating fault trees», Reliability Engineering & System Safety, Vol. 58, 1997

[Stood]: Stood est un outils permettant la modélisation en UML et en AADL. www.ellidiss.com/stood.asp.

[Teoh & Case 2004]: P. Teoh & K. Case. Failure modes and effects analysis through knowledge modelling. Journal of Materials Processing Technology 153-154, pp. 253-260, 2004.

[Topcased] : Topcased est un outil open source basé sur Eclispe. Il permet la modélisation en UML, SysML (module Papyrus) et en AADL (module Adele/Osate). http://www.topcased.org

[Tourtelier et al. 2010]: D. Tourtelier, F. Davenel, J. Rigo, C. Moreau, M. Thuault. Influence du profil de vie sur l'évaluation de la fiabilité de systèmes électroniques. 17e Congrès Lambda Mu, La Rochelle, France, 5-7 octobre 2010.

[Vanderperren 2005] Yves Vanderperren & Wim Dehaene, SysML and Systems Engineering Applied to UML-Based SoC Design, Proceedings of 2nd UML-SoC Workshop at 42nd DAC, Anaheim (CA), USA, 2005.

[Villemeur 1988]: A. Villemeur. Sûreté de fonctionnement des systèmes industriels. Editions Eyrolles, 1988.

[Willard 2007]: B. Willard. UML for systems engineering. Computer standard and interface, Vol. 29, pp. 69-81, 2007.

ANNEXES

ANNEXE 1 : Différentes méthodes d'analyse de sûreté de fonctionnement

Cette annexe présente un panel de méthodes d'analyse de sûreté de fonctionnement statique et dynamiques.

1. Analyse statique de la sûreté de fonctionnement

L'évaluation de la sûreté de fonctionnement d'un système consiste à analyser les défaillances des composants pour estimer leurs conséquences sur le service rendu par le système. Les principales méthodes traditionnelles utilisées lors d'une analyse de la sûreté de fonctionnement sont décrites ci-dessous.

1.1. L'Analyse Préliminaire des Risques

L'Analyse Préliminaire des Risques (APR) est une extension de l'Analyse Préliminaire des Dangers (APD) qui a été utilisée pour la première fois aux Etats-Unis, au début des années soixante [Villemeur 1988]. Depuis, cette utilisation s'est généralisée à de nombreux domaines tels que l'aéronautique, chimique, nucléaire et automobile.

Cette méthode a pour objectifs : 1) d'identifier les dangers d'un système et de définir ses causes. 2) d'évaluer la gravité des conséquences liées aux situations dangereuses et les accidents potentiels. L'APR permet de déduire tous les moyens, toutes les actions correctrices permettant d'éliminer ou de maîtriser les situations dangereuses et les accidents potentiels. Il est recommandé de commencer l'APR dès les premières phases de la conception. Cette analyse sera vérifiée, complétée au fur et à mesure de l'avancement dans la réalisation de système. L'APR permet de mettre en évidence les événements redoutés critiques qui devront être analysés en détail dans la suite de l'étude de sûreté de fonctionnement. L'APR est donc une étude générale pouvant être appliquée à tout type de systèmes

1.2. L'Analyse des Modes de Défaillance, de leurs Effets et de leurs Criticités

L'Analyse des Modes de Défaillance, de leurs Effets et de leur Criticité (AMDEC) est une méthode d'analyse qualitative des défaillances de systèmes complexes utilisée pour la première fois à partir des années soixante pour l'analyse de la sécurité des avions, avant de s'étendre à l'industrie manufacturière dans les années 80 [Villemeur 1988]. Elle est maintenant utilisée dans la quasi-totalité des domaines. L'AMDEC est née d'attentes très précises en matière d'analyse et de conception. Elle permet une recherche systématique et exhaustive des risques encourus par le système, facilitant le travail des experts [Ridoux 1999] [Picardi et al. 2004]. Cette méthode est applicable durant tout le cycle de vie du système y compris différentes phases de conception. Réalisée au plus tôt dans les phases de conception, elle permet de détecter très tôt les points sensibles sur lesquels focaliser toutes les attentions et ainsi diminuer l'impact financier de telles défaillances [Price 1996]. Réalisée en phase d'exploitation, elle devient une démarche d'amélioration du système [Teoh & Case 2004].

C'est une méthode inductive et qualitative qui propose d'explorer le système composant par composant. Pour chacun d'eux, on recherche les modes de défaillance et leurs effets sur le système, en détaillant leur criticité et leur probabilité d'occurrence pour ainsi souligner les points sensibles du système. On tente ensuite de proposer des solutions pour circonvenir les problèmes soulevés. Le

but d'une AMDEC est de permettre l'examen complet d'un système afin de mettre en évidence ses points critiques et permettre aux concepteurs de réaliser une solution atteignant ses objectifs en termes de SdF. L'AMDEC organise donc les activités suivantes [Pillay & Wang 2003] :

- Évaluer les effets des MdD des composants sur les performances.
- Identifier les composants critiques pour la sécurité.
- Développer des améliorations du système pour renforcer la sécurité et la fiabilité du système.

[Hecht et al. 2004] relève l'importance des AMDEC au cœur des activités de V&V (Vérification et Validation), où l'AMDEC a pour rôle de prouver que tous les MdD ont été identifiés et qu'une réponse satisfaisante leur est apportée. Elle a également pour rôle d'organiser l'effort de réduction des risques et l'activité de validation en soulignant les phénomènes dangereux dont l'étude est prioritaire. La réalisation d'une AMDEC passe par les étapes suivantes :

- Rechercher la décomposition du système et identifier les fonctions.
- Répertorier pour chaque composant/fonction des MdD envisageables.
- Déterminer la cause des défaillances.
- Rechercher les effets des défaillances à l'échelle du composant, du système et de l'environnement.
- Réaliser la cotation des risques : gravité (g), fréquence (f), (parfois détectabilité (d)).
- Calculer le RPN (Risk Priority Number). RPN = g * f * d
- Élaborer des actions correctives et des moyens de détection pour réduire le risque.

Les cotations sont établies par les membres du groupe de travail de façon numérique. Des échelles de cotation doivent préalablement être fournies pour chaque notion. Le RPN peut être obtenu sous forme numérique ou suivre une table de résultat comme celle proposée dans [Guiochet 2003]. De très bons exemples d'échelles de cotations peuvent être trouvés dans [Bowles & Pelaez 1995]. De nombreuses alternatives peuvent être trouvées dans les documents normatifs décrivant les AMDEC : MIL-STD1629A [MIL-STD1629A], BS 5760-5 [BS 5760-5], IEC 60812 [IEC 60812]. L'AMDEC aboutit à la rédaction d'un tableau dont chaque ligne décrit un risque encouru par le système. Ces lignes sont généralement ordonnées dans le document final par importance de leur RPN.

La littérature distingue deux types principaux d'AMDEC : les AMDEC composants et les AMDEC fonctionnelles. L'AMDEC fonctionnelle étudie les MdD applicables aux fonctions du système et leur impact sur les autres fonctions. Ce type d'AMDEC est réalisable dès les premiers résultats de l'activité d'Analyse Fonctionnelle. L'AMDEC composant repose quant à elle sur l'architecture organique du système, détaillant les MdD des composants et leur impact sur les composants voisins. Ce type d'AMDEC nécessite donc que l'architecture organique soit définie auparavant.

Sur la plupart des systèmes physiques, les deux types d'AMDEC peuvent être réalisées voir corrélées en une analyse unique mêlant fonctions et composant. Pour les études de système purement logiciel, seule l'AMDEC fonctionnelle est réalisable. L'AMDEC fonctionnelle n'est donc pas a négligée notamment lors de l'étude d'un système embarqué où l'aspect logiciel est fortement impactant.

1.3. Les Arbres de Défaillances

L'analyse par Arbre de Défaillance (AdD) est une analyse déductive qui permet de représenter graphiquement les combinaisons d'événements qui conduisent à la réalisation de l'événement redouté. L'arbre de défaillance, dont la racine correspond à l'événement redouté, est formé de

niveaux successifs tels que chaque événement soit généré à partir des événements du niveau inférieur par l'intermédiaire d'opérateurs logiques. Le critère d'arrêt de la décomposition arborescente d'un AdD est la connaissance que l'on a et l'appréciation de l'intérêt de la poursuite du processus de décomposition.

L'analyse quantitative consiste à assigner à chaque événement de base une probabilité d'occurrence. Plusieurs méthodes d'évaluation de la probabilité d'occurrence de l'événement sommet existent. Des exemples basés sur les diagrammes de décision binaire sont présentés dans [Rauzy & Dutuit 1997] et [Sinnanmon et Andrew 1997].L'analyse par arbre de défaillance caractérise de façon claire les liens de dépendance, du point de vue dysfonctionnement, entre les événements redoutés du système. De plus, puisque la méthode est fondée sur une étude des événements de défaillance, elle peut s'adapter aussi bien à l'analyse de l'architecture fonctionnelle d'un système qu'à l'analyse de l'architecture organique. Cependant, dans les deux cas, elle nécessite une connaissance fine des modes de défaillance des fonctions et/ou composants afin de réaliser une étude quantitative. Enfin, l'une des limites des arbres de défaillance est que l'ordre d'occurrence des événements menant vers l'état redouté n'est pas pris en compte. Or comme nous allons le voir par la suite, cette notion d'ordre dans les événements menant vers une défaillance est significative dans les systèmes complexes. C'est pour cela qu'une extension des AdD a été réalisée : les Arbres de défaillances dynamiques permettant la prise en compte de l'ordre de déclenchement des événements.

2. Analyse dynamique de la sûreté de fonctionnement

Les méthodes classiques de la sûreté de fonctionnement, comme celles présentées dans le §2 sont adaptées à des systèmes statiques, c'est-à-dire des systèmes dont les relations fonctionnelles entre leurs composants restent figées. L'analyse dynamique des défaillances est cependant essentielle pour l'étude des systèmes complexes. D'autres méthodes ont donc été introduites, nous allons en présenter quelques-unes ci-dessous, afin d'identifier les éléments sémantiques nécessaires à ce type d'études qui n'était pas présent avec les méthodes statiques.

2.1. Chaines et graphes de Markov

La méthode de graphes de Markov est utilisée pour analyser et évaluer la sûreté de fonctionnement des systèmes réparables. La construction d'un graphe de Markov consiste à identifier les différents états du système (défaillants ou non défaillants) et chercher comment passer d'un état à un autre lors d'un dysfonctionnement ou d'une réparation.

Les états sont classés en deux catégories :

- Des états de fonctionnement : ce sont les états où la fonction du système est réalisée. Des composants du système pouvant être en panne, l'état du bon fonctionnement est l'état où aucun composant n'est en panne,
- Des états de panne : ce sont des états où la fonction du système n'est plus réalisée, un ou plusieurs composants du système étant en panne.

Le processus d'analyse comprend trois parties :

- Le recensement et le classement de tous les états du système en état de bon fonctionnement ou états de panne.
- Le recensement de toutes les transitions possibles entre ces différents états et l'identification de toutes les causes de ces transitions.
- L'évaluation des probabilités de se trouver dans les différents états au cours d'une période de vie du système ou le calcul des caractéristiques de sûreté de fonctionnement.

La modélisation avec les graphes de Markov permet de prendre en compte les dépendances temporelles et stochastiques plus largement que les méthodes classiques. En dépit de leur simplicité conceptuelle et leur aptitude à pallier certains handicaps des méthodes classiques, les graphes de Markov souffrent de l'explosion du nombre des états [Moncelet 1998], car le processus de modélisation implique l'énumération de tous les états possibles et de toutes les transitions entre ces états.

2.2. Réseaux de Petri stochastiques

Les réseaux de Petri stochastiques [Molloy 1982], [Natkin 1980], [Chiol et al 1993] sont obtenus à partir des réseaux de Petri classiques en associant des durées de franchissement aléatoires aux transitions. Ils permettent de prendre en compte, de manière plus structurée que les graphes de Markov, l'occurrence des défaillances et leur influence sur le comportement du système. En effet, le parallélisme étant pris en compte ils permettent d'expliciter l'architecture du système en décrivant indépendamment les états des divers objets composant le système et leurs interactions.

Une extension nommée "Réseaux de Petri Stochastiques Généralisés "(RDPSG) [Marsan et al 1984] permet de prendre en compte, en plus de transitions avec des lois exponentielles, d'autres transitions dites « immédiates » tirées sans délai et qui sont prioritaires par rapport aux transitions à délai aléatoire. On peut citer d'autres extensions telles que les Réseaux de Petri Stochastiques Etendus (RdPSE) [Dugan et al 1984] et les Réseaux de Petri Stochastiques et Déterministes (RdPSD) [Marsan & Chiola 1986]. Les RdPSE permettent de prendre en compte des lois de distribution quelconques et les RdPSD combinent des délais exponentiellement distribués et des délais constants.

2.3. Les automates d'états

Les *automates* sont l'un des formalismes états transitions les plus utilisés dans la description des systèmes à événements discrets. Ce formalisme a été étendu sous la forme des automates hybrides pour modéliser correctement les systèmes dynamiques. Les automates hybrides sont une extension des automates temporisés (Les automates temporisés sont des automates à états finis étendus par un ensemble d'horloges dont les valeurs croissent uniformément avec le passage du temps et qui peuvent être remises à zéro) [Alur et al 1995]. Informellement, un *automate hybride* est l'association d'un automate à états finis et d'un ensemble d'équations dynamiques continues pilotées par ce dernier.

Le point faible de ce formalisme est l'explosion combinatoire du nombre d'états du graphe. Pour éviter ce problème, dans le cas de la modélisation des systèmes complexes pouvant être découpés en sous-systèmes, il est possible de construire un modèle d'automate pour chacun d'eux et de les composer ensuite pour élaborer l'automate correspondant au système global. La composition se fait par synchronisation entre les automates des différents sous-systèmes, soit par messages, soit par variables partagées. Toutefois cette composition entre automates rend difficile l'analyse de leurs propriétés. D'où le besoin de disposer de mécanismes de structurations plus puissants offerts par des modèles de plus haut niveau comme les Statecharts [Harel 1987].

Les relations de cause à effet menant vers l'état redouté ne sont pas représentées d'une façon claire et homogène avec les automates. En effet au sein d'un automate représentant un objet séquentiel (élément d'un produit d'automates ou d'un ensemble d'automates communicants), les relations de cause à effet sont représentées par les événements qui relient, chacun, un état origine et un état destination. Chaque événement correspond à une causalité explicite entre deux états.

Par contre entre deux automates, ces relations de cause à effet sont la conséquence directe ou indirecte de synchronisations par messages ou de communication par variables partagées. L'existence d'une relation de causalité ou non dépendra de la valeur du message ou de la façon suivant laquelle la variable partagée est modifiée et testée. Cela donne une représentation non unifiée des relations de cause à effet inter et intra automates.

2.4. Le langage AltaRica DF

Il est issu d'une collaboration entre le LaBRI (Laboratoire Bordelais de Recherche en Informatique) et plusieurs industriels. L'objectif est d'établir divers ponts entre la sûreté de fonctionnement et les méthodes formelles, les analyses quantitatives des dysfonctionnements et les analyses qualitatives des comportements fonctionnels, afin de fournir aux ingénieurs concepteurs de systèmes complexes, un atelier outillé, [Point 2000][Boiteau et al. 2006]. Une description du système dans ce langage, basé sur les automates à contraintes, permet de simuler un modèle et de générer un arbre de défaillance ou de vérifier qu'il satisfait des propriétés logiques. L'objectif depuis 1996, est de développer un outil générique permettant l'analyse des systèmes incorporant :

- un langage de spécification de haut niveau AltaRica
- un model-checker MecV
- des outils de sûreté de fonctionnement

Dans l'atelier AltaRica, un modèle AltaRica est transformé en des formalismes de plus bas niveau, comme les systèmes de transitions, et rend ainsi possible la vérification grâce au model checking.

L'atelier AltaRica est une réponse à certains aspects du contrôle de processus. Une de ses limites est l'impossibilité d'exprimer des modèles nécessitant des opérateurs de modélisation temps réel. Des travaux ont cependant étaient menés, ils avaient pour but d'étendre le modèle AltaRica à ce genre de systèmes et ils ont donné naissance à Timed AltaRica et Hybrid AltaRica.

Robin Cressent

Valorisation de l'Ingénierie Système à Base de Modèles, pour l'analyse de sûreté de fonctionnement des systèmes complexes critiques intégrant des COTS

À l'heure actuelle et depuis plusieurs années, les nouveaux systèmes développés par les industriels ne cessent de se complexifier, de faire intervenir toujours plus de technologies différentes et cela, pour maximiser la rentabilité, améliorer les fonctionnalités, voire proposer de nouveaux services. L'approche d'ingénierie système à base de modèles (ISBM) adresse particulièrement ces problématiques et est de plus en plus plébiscitée par les industriels. Pour autant, l'ISBM ne permet pas d'assurer la sûreté de fonctionnement (SdF) de nos systèmes modernes. C'est pourquoi ces travaux visent à assurer la cohésion de l'ISBM avec les études de SdF ainsi que sa valorisation pour ces études.

Le langage de modélisation SysML est choisi pour réifier l'ensemble des résultats des activités d'ingénierie système sous la forme d'un modèle système. Ce modèle système est ensuite manipulé par les processus de la méthodologie MéDISIS, définie dans la thèse, afin de faciliter les études de SdF. La thèse aborde notamment la génération d'AMDEC fonctionnelle et l'application de la méthodologie FIDES pour l'évaluation de fiabilité, avec l'aide de bases de données pérénisant les informations dysfonctionnelles.

Les principes précédents sont aussi appliqués à un projet industriel conséquent : le projet LEA qui consiste à la réalisation d'un prototype de véhicule hypersonique. La thèse met l'accent sur l'étude de SdF des composants de type COTS qui sont courant au sein du projet LEA.

Mots clés : SysML, Sûreté de fonctionnement, Ingénierie Système, AMDEC, FIDES, AADL, Simulink.

Promotion of the Model-Based System Engineering approach for dependability analyses of critical complex systems integrating COTS

Nowadays, industrial systems are getting more and more complicated, integrating various technologies. Their designs involve many different engineering fields to maximize rentability and to offer the most up to date functionalities and services. The Model-Based System Engineering (MBSE) approach address specifically these issues by allowing a more global way of designing a complex system from many points of view. However, MBSE does not ensure dependability. That is the reason why, in this thesis, our work aims to connect the MBSE approach with dependability analysis.

The SysML modeling language is used to reify the results of system engineering activities, to obtain a model of the system. This model is then computed to extract data that will help dependability analysis. These automatic processes of extraction and redaction are part of the MéDISIS methodology which is defined in this thesis. Two aspect of the MéDISIS methodology are discussed: the generation of functional FMEA and the use of the FIDES methodology in association with SysML to evaluate the failure rate of COTS.

This work and the whole MéDISIS methodology are applied in the industrial context of the LEA project. This project, financed by MBDA, consists in designing an hypersonic vehicle.

Keywords : SysML, Reliability analysis, System Engineering, FMEA, FIDES, AADL, Simulink.

Laboratoire PRISME
ENSIB
88, boulevard Lahitolle
18020 Bourges Cedex