Contribution à l'évolution des exigences et son impact sur la sécurité

Mohamad Hani El Jamal

Année 2006

▪ Thèse

Préparée au Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

En vue de l'obtention du Doctorat de l'Université du Paul Sabatier - Toulouse

Spécialité : Systèmes Industriels

Par Mohamad Hani ELJAMAL

Contribution à l'évolution des exigences et son impact sur la sécurité

Soutenance le 12/12/2006 devant le jury

Mr. Abd-El-Kader SAHRAOUI	***Directeur***	Professeur à l'Université de Toulouse le Mirail
Mr. Armand TOGUYENI	***Rapporteur***	Professeur à l'Ecole Centrale de Lille
Mr. Jean-Jacques SCHWARZ	***Rapporteur***	Professeur à l'Université de Claude Bernard, Lyon
Mr. Mario PALUDETTO	***Examinateur***	Professeur à l'Université de Paul Sabatier, Toulouse
Mr. Pierre DE SAQUI SANNES	***Examinateur***	Professeur à l'ENSICA de Toulouse
Mr. Pierre DE CHAZELLES	***Examinateur***	Ingénieur-AIRBUS, Toulouse

A MES PARENTS

A MON PROF DE THESE

A MA FUTURE FEMME…HEBA

REMERCIMENTS

Les travaux présentés dans ce manuscrit sont l'aboutissement de trois années d'études réalisées au Laboratoire d'analyse et d'Architecture des Systèmes (LAAS-CNRS) dans le groupe de l'Ingénierie Système et Intégration (ISI).

Je tiens à exprimer toute ma reconnaissance à Monsieur Malik Ghallab, pour m'avoir accueilli et mis à ma disposition les moyens nécessaires à la réalisation de mes travaux.

Je suis très reconnaissant envers Monsieur Abd-El-Kader Sahraoui mon directeur de thèse pour son encadrement, ses critiques, ses nombreux conseils, son soutien constant tout au long de ma thèse et son rôle déterminant dans l'avancement de mon travail. Je tiens à lui témoigner tout particulièrement ma reconnaissance pour m'avoir supporté dans son bureau durant ces dernières années.

Je remercie très chaleureusement Monsieur Armand Toguyeni d'avoir accepté d'être rapporteur de ma thèse, ainsi pour ses commentaires sur mon mémoire, je lui exprime ma profonde gratitude.

Un grand remerciement à Monsieur Jean Jacques Schwarz, qui m'a fait l'honneur de me rapporter ma thèse, ainsi que pour ses jugements très pertinents sur mon manuscrit.

Je suis reconnaissant envers Monsieur Chris George pour son soutien et son encouragement durant mon séjour à Macao au sein de l'IIST-UNU (International Institute of Software Engineering - United Nations University).

Je remercie Madame Anne Hauxthausen pour son accueil chaleureux durant mon stage à l'université Technique du Danemark (DTU).

J'adresse mes remerciements à Monsieur Mario Paludetto pour m'avoir accueillir dans son équipe, et pour Mr. Hamid Demmou pour ses aides au cours de la rédaction du manuscrit.

J'adresse mes remerciements à Messieurs Pierre De Saqui Sannes, Pierre De chazelle d'avoir accepté de faire partie de mon jury de thèse.

Je remercie l'association scientifique pour le développement qui a financé ma bourse d'étude pendant la durée de mon séjour en France.

A cette occasion, je remercie tous les chercheurs, enseignants, membres du personnel du laboratoire LAAS-CNRS, et particulièrement les membres du groupe ISI pour leurs amitiés et leurs aides pendant ces trois années de thèse.

Merci aussi à tous les amis qui m'ont soutenu durant ces années, soit directement au LAAS, soit à l'extérieur du Laboratoire : Aimed, Adel, Bachar, Bassam, Dani, Heba, Kazbour, Mourad, Mahmoud, Mohamad Ali, Nadim, Okika, Perna, Ramdane, Saade, wassim, Youcef.

Enfin, je remercie toute ma famille pour son soutien tout le long de cette thèse notamment dans les moments les plus difficiles.

ELJAMAL Mohamad Hani

Toulouse, Décembre 2006

ACRONYMES

AMDEC : Analyse des Modes de Défaillances, leurs Effets et de leur Criticité

ARP: Analyse Préliminaire des Risques

DOORS: Design of Object Oriented Real-time systems

EIA-632: Electronic Industries Association

IEC: International Electromechanical Commission

INCOSE: International council on Systems engineering

IIST: International Institute of Software Technology

PVS: Proof Verification System

RAISE: Rigorous Approach for Industrial Software Engineering

RIM: Requirement change Information Model

RSL: RAISE specification language

SEL: Safety Integrity Level

SRS: System Requirements Specification

UML: Unified Modelling Language

WIM: Work product Information Model

WORM: Work product Requirement trace Model

PLAN

INTRODUCTION GENERALE 1

CHAPITRE I : INTRODUCTION GENERALE 3

1. INTRODUCTION 3

2. EVOLUTION DU SYSTEME 4

3. VOLATILITE DES EXIGENCES 5

3.1. L'effet de la volatilité des exigences 6

3.2. Etude statistique sur la volatilité 7

3.3. Volatilité et densité de défaut 7

4. LE COUT D'UNE CORRECTION 8

5. INGENIERIE SYSTEMES 9

5.1. Etude à travers l'EIA-632 10

5.2. Processus de l'EIA-632 12

5.2.1. Processus d'acquisition et de fourniture 13

5.2.2. Processus du management 13

5.2.3. Processus de conception 13

5.2.4. Processus de réalisation 14

5.2.5. Processus technique d'évaluation 14

6. INGENIERIE DES EXIGENCES 15

6.1. Les objectifs de l'ingénierie des exigences 16

6.1.1. L'identification des parties prenantes 17

6.1.2. L'élicitation des exigences 17

6.1.3. L'analyse des exigences 19

6.1.4. La formalisation des exigences 20
6.1.5. La vérification/ validation des exigences 21
6.1.6. La modification des exigences 21
6.2. Description du SRS 21
6.2.1. Buts testables 21
6.2.2. Exigences testables 22
6.3. Caractéristiques des exigences 22
6.3.1. Différents types d'exigences 22
6.3.2. Liens entre les exigences 23
7. IMPACT DU CHANGEMENT 24
8. CONCLUSION 25
CHAPITRE II : ETAT DE L'ART ET PROBLEMATIQUE 27
1. INTRODUCTION 27
2. APPROCHES DE LA RECHERCHE D'IMPACT 27
2.1. Processus d'Analyse 28
2.2. Analyse basée sur un modèle de traçabilité 29
2.3. Spécification intentionnelle 31
3. LA STRUCTURE SELON L'EIA-632 33
3.1. Développement descendant 34
3.2. Réalisation ascendante 35
4. DEVELOPEMENT SELON L'EIA-632 35
5. METHODES D'ANALYSES 36
5.1. Analyse préliminaire du risque 37
5.2. AMDEC 37
5.3. Analyse du risque opérationnel 38
6. GESTION DE LA SECURITE 38

6.1. Processus d'analyses et leurs occurrences 38
6.2. La norme du DOD 39
6.3. La norme de l'IEC 40
7. LA TRACABILITE 42
7.1. Liens de traçabilité 42
7.2. Pré/Post traçabilité 43
8. SPECIFICATION FORMELLE 44
8.1. Les attributs d'une Spécification formelle 45
8.2. Méthodes de vérification 46
8.2.1. Preuve automatique 46
8.2.2. Model-Checking 46
8.3. Spécification Semi Formelle/Formelle 47
8.3.1. UML/PVS 48
8.4. Spécification formelle avec RAISE 49
8.4.1. La méthode RAISE 50
8.4.2. Langage de spécification RAISE : RSL 50
8.4.3. D'UML vers RSL 50
8.4.4. De RSL vers SAL 51
9. CONCLUSION 51
CHAPITRE III : METHODOLOGIE 53
1. INTRODUCTION 53
2. RELATIONS ET TRACES 53
2.1. Relations durant le développement 54
2.2. Les traces dans un module 56
3. SYSTEME DE SECURITE 58
3.1. Exigences Sécuritaires 60

3.1.1. Sécurité fonctionnelle 61
3.1.2. Sécurité non fonctionnelle 64
4. DEMARCHE 65
4.1. Les catégories de la demande de changement 65
4.2. Règles à suivre 65
4.3. L'Approche Générale 66
4.3.1. Gestion de la modification 68
4.3.2. Analyse d'impact sur la sécurité 69
5. CYCLE DE VIE D'UNE MODIFICATION 73
5.1. Phase d'initialisation 75
5.1.1. Processus d'identification 75
5.1.2. Processus de Soumission 75
5.2. Phases d'évaluation 75
5.2.1. Processus de Traçabilité 76
5.2.2. Processus d'évaluation d'impact 76
5.2.3. Processus de Classification 76
5.2.4. Processus d'approbation de la demande 76
5.3. Phase de réalisation 77
5.3.1. Application de la modification 77
5.3.2. Nouveaux liens de traçabilité 77
5.4. Phase de validation 78
5.4.1. Processus de test 78
5.4.2. Validation de la modification 78
6. SYSTEME D'INFORMATION 79
7. CONCLUSION 80

CHAPITRE IV : SPECIFICATION ET OUTILLAGE 83
1. INTRODUCTION 83
2. ARCHITECTURE DE L'OUTIL 84
2.1. Diagramme de cas d'utilisation 84
2.2. Diagramme de classes 86
3. UML VERS RSL 88
3.1. Les types 89
3.2. Système 90
3.3. Projet 90
3.4. Produit Capacitants 92
3.5. Les fichiers 93
3.6. Exigences Techniques 94
3.7. Enabling Technology 95
3.8. Demande de changement 96
3.8.1. Différents types de demande 97
3.8.2. Fonction d'initialisation 98
3.8.3. Fonction d'évaluation 98
4. SPECIFICATION DE LA SECURITE 99
4.1. Lien avec la sécurité 100
4.2. Analyse d'impact 103
5. IMPLEMENTATION DE L'OUTIL 104
5.1. Développement avec JAVA 104
5.1.1. L'aspect orienté objet 105
5.1.2. Indépendance vis-à-vis de la plateforme 105
5.1.3. Le Swing 106
5.2. L'environnement Eclipse 107

6. ETUDE DE CAS 108
6.1. Présentation 108
6.2. Feux de croisement (S1) 108
6.2.1. La décomposition selon l'EIA-632 109
6.2.2. Le système de sécurité 111
6.3. Feux de croisement après évolution (S2) 112
6.3.1. Le cycle de vie de la demande 113
6.4. Outillage 115
7. CONCLUSION 116

CONCLUSION GENERALE 117
BIBLIOGRAPHIES 119
ANNEXE 127
ANNEXE A 129
1. ELICITATION DE L'OUTIL 129
1.1. Les buts primaires 129
1.2. Les cas d'utilisation 130
ANNEXE B 133
1. L'ALGORITHME DE CHANGEMENT 133
ANNEXE C 135
1. BUT 135
2. MODULE 135
3. STAKEHOLDER 135
4. PRODUIT FINAL 136
ANNEXE D 137
1. PRINCIPES DE SECURITE 137

ANNEXE E 139

1. LE MODULE : TYPES 139

ANNEXE F 143

1. SPECIFICATION FORMELLE 143

1.1. Les types 143

1.2. Les fonctionnalités 143

INTRODUCTION GENERALE

Ce travail porte sur la problématique de l'évolution des exigences dans le contexte de l'ingénierie système. L'évolution des exigences est un processus important qui implique la traçabilité et la gestion de configuration. Les causes de l'évolution peuvent provenir de chaque phase du développement du système ; allant de la capture des exigences jusqu'à la phase de l'opération. Pratiquement, les parties prenantes sont les sources de changement des exigences. Une évolution des exigences peut se manifester lors de l'utilisation du système, lorsque l'utilisateur fait des recommandations ; ou bien lors de la phase de test.

La plupart des systèmes actuels sont une évolution des systèmes déjà existants, comme dans le domaine des télécommunications, l'aéronautique, et celui de la robotique. Malheureusement, l'occurrence d'une évolution au cours du développement des systèmes complexes ou bien après leur réalisation, affecte plusieurs aspects comme la sécurité, le coût du développement, et le délai. En effet, l'évolution des exigences est un aspect important qui affecte les activités du développement des systèmes. L'importance de l'évolution des exigences est reconnue aussi bien dans le milieu académique qu'industriel. Les résultats de la recherche et les expériences industrielles sont encore très rares et pas encore bien approfondies.

Notre étude a porté principalement sur la relation entre la demande de changement d'une exigence et l'un des effets indésirables, à savoir la sécurité du système. Les deux questions évoquées durant cette étude, sont les suivantes : comment intégrer le processus d'évolution des exigences au cours du développement? Et, comment autoriser l'implémentation de la demande de changement en garantissant les propriétés invariantes liées à la sécurité, tout en suivant une démarche d'ingénierie système ? Ceci fait émerger la nécessité d'une approche méthodologique pour aborder l'évolution des exigences. Donc, cette thèse a pour but de prospecter les compréhensions courantes de l'évolution des exigences et explorer de nouvelles directions de recherche dont certaines ont été abordées dans ce travail. Ce mémoire est structuré en quatre parties comme indiquées ci-dessous :

- La première partie de la thèse porte sur la problématique du changement des exigences. Dans cette partie, nous allons aborder la question de la volatilité des exigences, ses effets et ses causes, puis nous présenterons notre contexte d'étude.

- Dans la deuxième partie, est consacrée à l'état de l'art. Au début, nous expliquerons les différentes approches et études qui ont été développées pour la recherche d'impact lors d'une demande de changement. Deuxièmement, nous présenterons les différentes techniques pour l'analyse et la gestion de la sécurité au cours du développement des systèmes complexes, et également les deux fameuses normes présentées dans le domaine de sécurité. Troisièmement, nous raffinerons le concept de la traçabilité et l'intérêt d'avoir un modèle de traçabilité durant le développement des systèmes. Finalement, nous présenterons la méthode formelle RSL qu'on va utiliser lors de l'analyse d'impact sur la sécurité.

- La troisième partie de la thèse concerne la méthodologie de la recherche d'impact sur la sécurité. Notre méthodologie sera décomposée en une présentation de tous types de relations rencontrés au cours du système, ces relations contribuent à la recherche d'impact lors d'une évolution. Puis, nous allons détailler notre décomposition du système de sécurité aux niveaux des exigences. Puis, nous allons aborder notre démarche de la recherche d'impact. Finalement, nous allons présenter le cycle de vie de la demande de changement, parmi les phases de ce cycle de vie nous avons intégré la phase d'évaluation d'impact, dans le but de décider si une telle évolution du système affecte un problème de sécurité.

- La quatrième partie de ce mémoire, présente la spécification de l'outil qui contient un modèle de traçabilité basé sur la norme d'ingénierie systèmes, aussi bien qu'il supporte la fonctionnalité de recherche d'impact sur la sécurité lors d'une demande de changement. Au début de cette partie, nous allons présenter la spécification de l'outil en utilisant les diagrammes d'UML. Mais, l'ambiguïté existante au niveau de la spécification avec UML nous a obligé d'utiliser le langage formel RSL.

CHAPITRE I

INTRODUCTION DE LA PROBLEMATIQUE

1. INTRODUCTION

Les systèmes actuels sont de plus en plus complexes, car ils intègrent une grande variété de technologies. Le développement de ce type de systèmes nécessite une longue période. En fait, plusieurs demandes d'évolutions apparaissent durant leur développement. Généralement, une évolution existe dans deux cas. Le premier cas est rencontré lors de la mise à jour du système. Le deuxième cas, est rencontré lors de l'émergence d'une nouvelle technologie.

En fait, plusieurs compagnies utilisent leurs propres méthodes pour appliquer une évolution, d'autres appliquent des méthodes développées dans différents domaines: militaire, électronique, ou de génie logiciel. Parmi ces méthodes, on peut citer la méthode présentée par Kai Gilb [GIB-04]. Cette méthode aide les développeurs à savoir comment appliquer une évolution, en utilisant la notation des tableaux d'estimations. Ces tableaux, présentent des pourcentages indiquant le taux de contribution de chaque solution proposée, pour atténuer le but primaire de l'évolution. Malheureusement, l'application de telle solution ne nous aide pas pour analyser l'impact de l'évolution sur la sécurité.

Dans notre étude, lorsqu'on considère qu'on a une évolution, on se focalise sur l'étude de l'effet du changement sur la sécurité, en analysant si cette évolution peut amener le système dans un état critique. Enfin, cette étude est consacrée à répondre aux questions suivantes :

- Comment peut-on analyser l'impact du changement sur la sécurité?
- Que signifie le terme sécurité dans notre étude ?
- Quelles sont les étapes qu'il faut exécuter durant l'analyse d'impact sur la sécurité?
- Quels sont les outils qui doivent supporter la méthodologie de l'analyse d'impact?

2. EVOLUTION DU SYSTEME

Le 19e siècle a été marqué par une importante réorganisation du travail. Plutôt que de fabriquer des chaussures une à une de façon artisanale, les industriels ont inventé des machines qui pouvaient les reproduire en grand nombre.

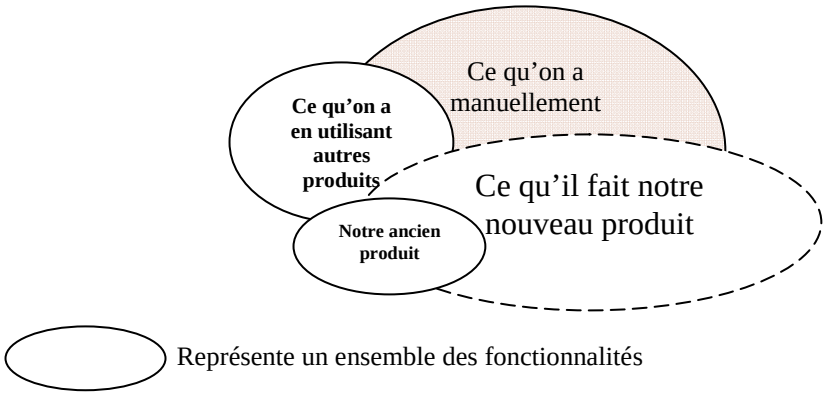

Figure I.1-Représentation graphique d'une évolution d'un système actuel, afin d'obtenir un nouveau système avec de nouvelles fonctionnalités.

A l'aube du 21e siècle, la technologie domine nos vies et son évolution nous semble effrénée. Après l'ère industrielle, les milieux du travail ont été envahis par l'informatique, et les ordinateurs étaient présents partout. Alors, nous avons pu constater comment la société a profité des nouvelles technologies et comment elles ont changé et fait évoluer la société.

La complémentarité entre la société et la technologie intervient de plusieurs façons : l'évolution des idées, l'apparition des nouvelles technologies qui amènent à d'autres technologies, l'évolution des productions manuelles vers des systèmes automatisées. Cette intervention oblige les industriels à répondre aux demandes de la société en faisant évoluer leurs systèmes existants.

En effet, lorsque les parties prenantes demandent l'application d'une modification soit sur un système actuel soit durant son cycle de développement, leur but est toujours de faire évoluer l'ensemble des fonctionnalités actuelles du système, ou de diminuer le coût du développement du système durant son cycle de vie. Mais avant d'appliquer l'évolution sur le système, il faut qu'ils répondent aux questions suivantes : Quel est le type de l'évolution? A quel niveau il faut l'appliquer? Est ce que cette évolution affecte la qualité du produit ? Quelle était l'ancienne fonctionnalité du système ?

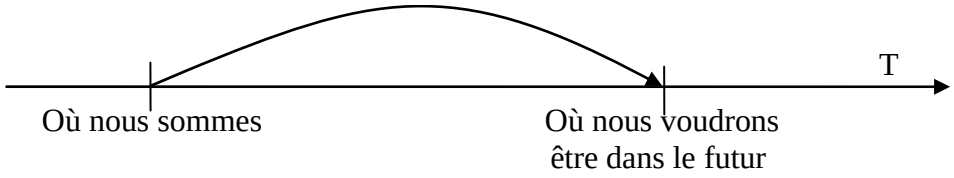

Figure I.2 – Représentation typique d'une évolution.

Enfin, certains développeurs figent les exigences du système qu'ils veuillent développer, en n'appliquant aucune évolution, vu que leurs projets ne supportent pas une méthodologie d'application d'une évolution au cours du développement du système. Mais figer les exigences et ne pas accepter l'application de modifications, à un effet négatif sur le succès des projets. Car, dans ce cas les développeurs réalisent le système qui ne correspond pas à leurs demandes.

3. VOLATILITE DES EXIGENCES

Dans le paragraphe ci-dessus, nous avons présenté les raisons d'appliquer une évolution sur le système. Mais, une évolution amène parfois à une volatilité des exigences[1]. Pratiquement, la diminution du taux de la volatilité des exigences est d'une importance primordiale pour les industriels. Car la volatilité a un effet négatif sur le succès du projet, vu qu'elle entraîne beaucoup de modifications. En fait, une modification d'exigence peut être: l'addition d'une exigence, la suppression d'une exigence, ou les deux ensemble. Plusieurs facteurs affectent la volatilité d'exigences [RUS-04]: changement d'une exigence, changement de la priorité d'une exigence, changement de technologie, le conflit d'exigences, changement d'environnement, ou la complexité du système.

Pratiquement, lors de la phase d'élicitation des exigences, les émetteurs des exigences communiquent avec les acquéreurs d'exigences par plusieurs méthodes. Ils utilisent l'écrit (texte, images, figures, mails) ainsi que l'oral (conversation, réunion, etc….). Tous ces moyens servent à exprimer les exigences statiques et dynamiques du système. Mais, derrière les mots se cachent des significations qui doivent être identiques entre les émetteurs et les acquéreurs. En effet, l'émetteur des exigences comprend nécessairement les exigences qu'il propose, et il s'engage à obtenir une solution et un coût fixe. Parfois, les émetteurs sous-estiment des exigences, ce qui oblige les acquéreurs d'exigences à appliquer des modifications sur le système, dans le but de réaliser le produit avec le budget précisé. Ce genre d'exigences

[1] La volatilité d'exigences est la mesure de l'instabilité des exigences. Alors, une volatilité des exigences entraîne toujours le changement des exigences.

retardées doit être élucidé obligatoirement durant le cycle de vie du système. En effet, trois causes sont à l'origine d'exigences retardées:

- Erreur d'élicitation des exigences,
- Erreur de représentation ou de modélisation des exigences,
- Erreur de compréhension d'exigence.

Ces exigences retardées expliquent parfois l'existence d'une volatilité des exigences. En fait, d'après des études qui ont été basées sur le développement des logiciels et des systèmes d'informations [KRA-89] [BOH-91], on peut remarquer que durant leurs développements existe toujours une volatilité d'exigences. Cette volatilité apparaît car le développement de logiciels commence toujours avec un ensemble d'exigences floues, incomplètes, non bien définies [KRA-89], ce qui affecte la volatilité des exigences dans la plupart des cas.

Par la suite nous allons présenter : l'effet de la volatilité des exigences, une étude statistique sur la volatilité, et finalement la volatilité et la densité de défaut.

3.1. L'effet de la volatilité des exigences

Les études précédentes se sont penchées sur l'étude de l'impact de la volatilité des exigences sur la productivité des logiciels [LAN-98] [BOH-96a]. Le travail présenté dans [ZOW-02] était destiné à décrire leurs constations d'après une étude empirique sur la volatilité des exigences, dans le but de présenter l'impact de la volatilité sur la performance des projets. Les auteurs ont trouvé que la volatilité a toujours un impact sur le coût et sur les délais.

Le but de leur étude était d'examiner les effets de la volatilité, en développant un modèle théorique qui : aide à étudier l'impact de la volatilité, améliore la pratique de l'ingénierie d'exigences dans le but de contrôler la volatilité, et réduit les effets négatifs de la volatilité sur la performance du projet. D'après cette étude, ils ont constaté trois axes majeurs qui expliquent la volatilité:

- Potentiel de changement (changement d'environnement),
- L'instabilité d'exigences (la variation d'exigences des utilisateurs),
- La diversité d'exigences (les parties prenantes ne sont pas d'accord entre elles).

3.2. Etude statistique sur la volatilité

N.Nurmiliani, D.Zowghi et S.Fowell [NUR-04] présentent leurs conclusions sur le pourcentage de changement, le modèle pour supporter le processus de changement, et la mesure de la volatilité des exigences.

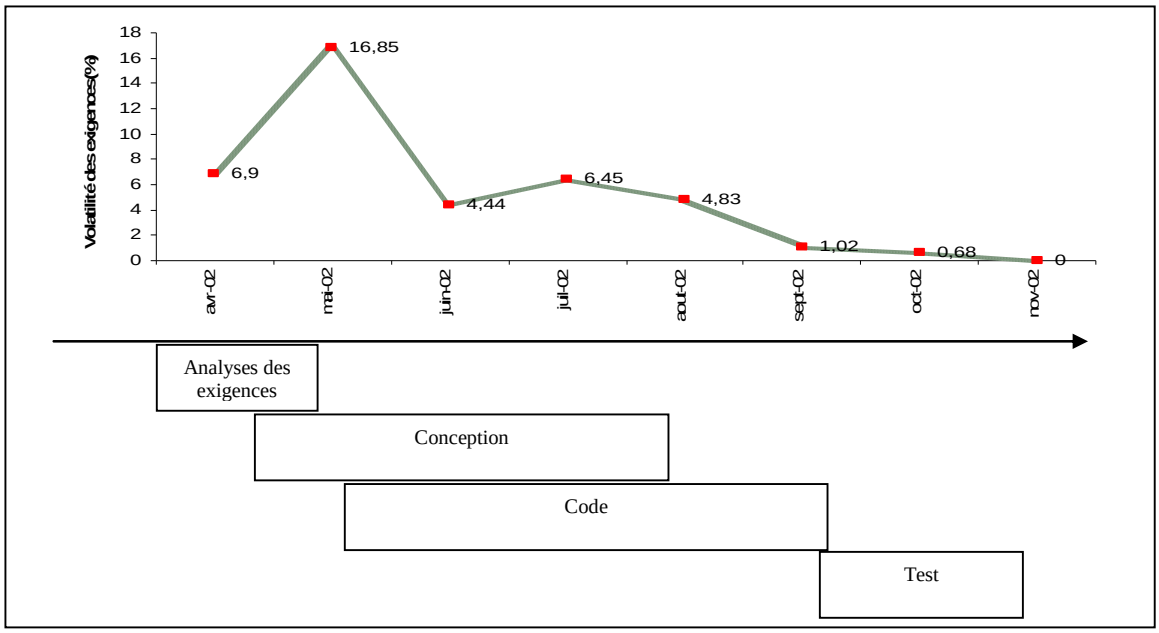

Figure I.3- Plus le taux de la volatilité durant le développement est bas, plus le taux de la réussite du projet est élevé.

En effet, ils ont effectué une étude sur la volatilité durant le développement d'un logiciel (16 mois), le nombre total des changements durant le développement du système étaient de 78 demandes. Ils ont montré que le pourcentage de la volatilité des exigences augmente lorsque les demandes de changement augmentent brusquement à la fin de la phase de l'analyse des exigences, et au début de la phase de conception (cf. figure I.3).

3.3. Volatilité et densité de défaut

Dans un cas idéal, les exigences pour le développement d'un logiciel doivent être complètes, sans ambiguïté, bien déterminées avant la conception, et surtout avant la réalisation. Souvent, il y a une obligation de modifier quelques exigences ce qui affecte l'augmentation de la densité de défaut du système.

Malaiya et Denton [MAL-98], présentent l'influence du changement des exigences durant le cycle de développement des logiciels, et ils présentent l'influence du moment où nous

effectuons la modification. En fait, ils montrent dans leur étude que nous n'avons pas la même densité de défaut si on effectue un changement au début ou à la fin du cycle de développement des logiciels. Egalement, dans leurs travaux nous pouvons constater une différence entre la densité de défaut pour deux projets différents, si un parmi les deux a des exigences non ambiguës [CAR-04].

4. LE COUT D'UNE CORRECTION

Les exigences sont la base de presque toutes les activités du développement des systèmes. Elles sont comme les plans d'un architecte de bâtiments: si les plans sont erronés, les constructeurs construiront les bâtiments incorrectement. Mais les plans d'un architecte peuvent être corrects seulement, s'ils spécifient exactement ce que les propriétaires et les habitants attendent du bâtiment après sa réalisation. Evidemment, chaque habitant peut confirmer que la correction d'une erreur présente après la réalisation du bâtiment sera trop coûteuse.

La construction des bâtiments, est un exemple parmi un ensemble de systèmes complexes qui ont une durée de vie longue. Dans l'industrie, le développement des systèmes est partagé en plusieurs projets. Il n'y a presqu'aucun projet de routine. Chaque projet construit un produit différent de tous les autres : Il emploie différentes technologies et résout les différents problèmes avec différentes tranches de temps, et un ensemble de personnes pour effectuer le travail.

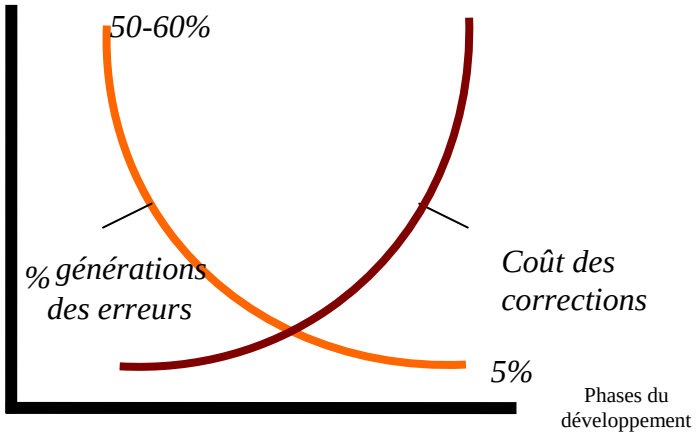

Figure I.4-Le coût d'une correction

Pour la plupart des projets, la moitié des erreurs sont les erreurs venant de la phase d'élicitation des exigences. La corrélation entre coût de correction et pourcentage de générations d'erreurs (cf. figure I.4) nous montre que, plus les modifications seront dans la

phase d'initialisation du projet (élicitation, analyse d'exigences, etc.…) plus on réduit le coût et le temps d'accomplir le projet. Cela signifie que les exigences ont un rôle crucial durant le processus de développement.

Malheureusement, beaucoup de projets s'accélèrent au début du développement des systèmes afin de gagner du temps. Dans ce cas, le résultat sera une mauvaise expression des exigences, des systèmes défectueux, et un coût élevé d'entretien durant les phases qui suivent la phase d'exigences. Au cours des projets, l'existence de la volatilité d'exigences et la génération d'erreurs et l'intégration des nouvelles technologies ont obligé les parties prenantes à élaborer des techniques pour connaître l'impact qui pourra être un :

- Impact sur le coût,
- Impact sur le délai,
- Impact sur la performance (sécurité)

L'analyse du type d'impact exige des statistiques, des estimations, et des expériences professionnelles pour les développeurs.

Enfin, l'augmentation du coût de correction des erreurs au cours du développement du système, et la volatilité des exigences sont parmi les problèmes qui nous obligent à travailler dans le contexte de l'ingénierie système et de l'ingénierie des exigences [JUR-02].

5. INGENIERIE SYSTEMES

Plusieurs définitions existent pour l'ingénierie systèmes. D'après la norme IEEE P1220 [STA-IEEE], on peut dire que l'ingénierie systèmes est une approche collaborative et interdisciplinaire qui décompose un système, le développe et vérifie que la solution décrite sur tout le cycle de vie satisfait aux attentes des parties prenantes.

A défaut de pouvoir associer une unique définition de l'ingénierie des systèmes, il est plus simple de l'identifier par les objectifs à atteindre dans la pratique. On peut citer les différents objectifs de l'ingénierie des systèmes : (1) la satisfaction des parties prenantes (en répondant à leurs demandes), (2) la recherche de qualité technique, (3) tenue des délais, (4) tenue des coûts en respectant le budget alloué.

Parfois, le développement d'une nouvelle technologie souffre des processus, et de méthodes incorrectes utilisés durant le développement du système. Pour cela, les développeurs des systèmes complexes où émergent les nouvelles technologies utilisent des normes qui

aident à accomplir leurs missions durant le développement des systèmes complexes. Parmi ces normes, on cite l'EIA-632.

5.1. Etude à travers l'EIA-632

L'objectif de l'EIA-632 [STA-EIA], est d'être la norme la plus populaire et efficace, et la plus utilisée dans le domaine de l'ingénierie systèmes. En effet, une telle norme a été déployée principalement dans des industries de l'aéronautique, de production et militaires.

Selon beaucoup d'expériences de projets industriels échoués [SAH-04a], on peut évidemment préciser que parfois la cause de l'échec est une négligence pour les produits qui contribuent et qui aident à avoir le système désiré, vu que la plupart des parties prenantes focalisent seulement sur le développement des produits finaux[2].

En effet, l'importance de développer le système en suivant la norme industrielle (EIA-632) est : d'explorer un ensemble de processus qui sera appliqué pour le développement des produits finaux (cf. figure I.5), de décomposer le système entre des produits finaux, et des produits capacitants (cf. figure I.6).

[2] Le produit final exécute les fonctions opérationnelles du système.

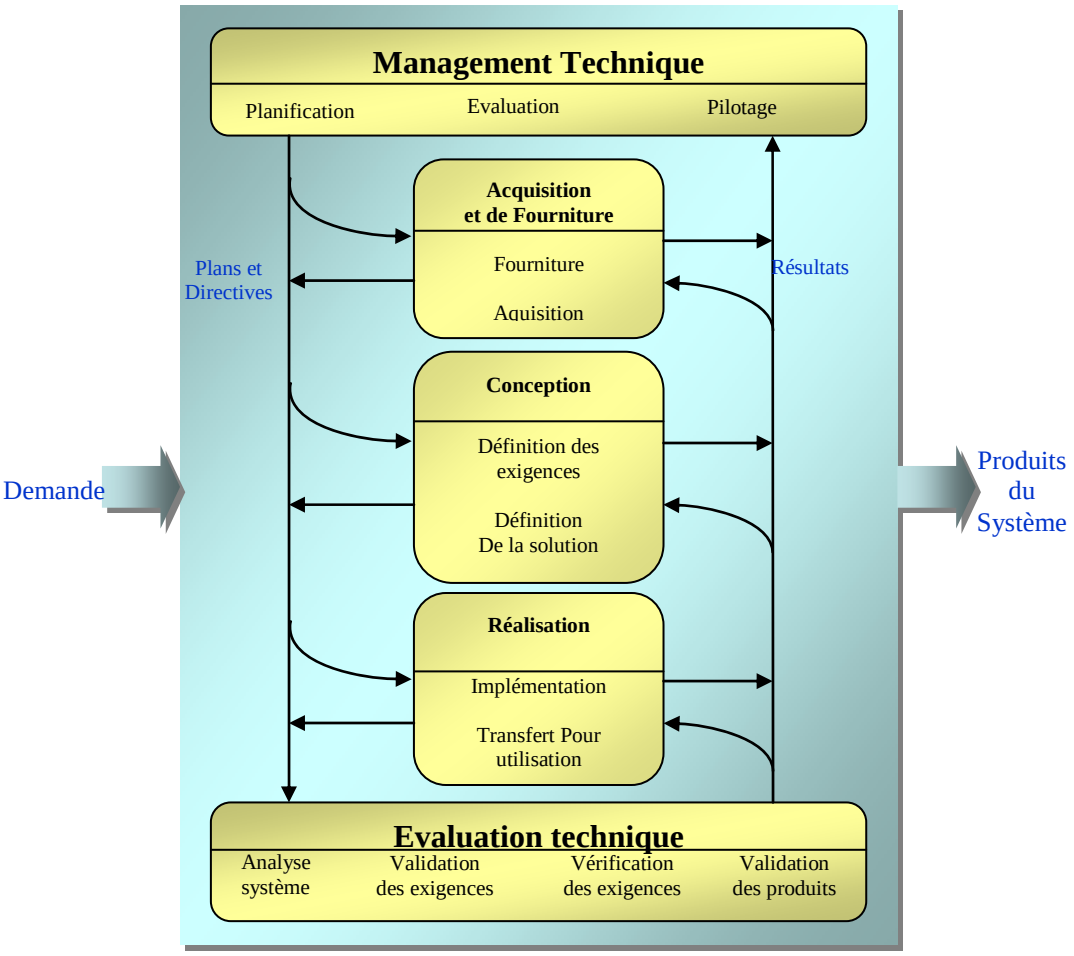

Figure I.5-Processus pour le développement des produits finaux.

Dans le cadre de l'EIA-632, les produits capacitants[3] (cf. figure I.7) seront utilisés pour exécuter l'ensemble des processus associés au développement, à la production, au déploiement, à la formation des opérateurs afin d'utiliser les produits finaux, et à la retraite des produits quand ils ne seront plus utilisables.

Dans la figure ci-dessous on présente sept types de produits capacitants pour : le développement du produit final, la production du produit final, le test du produit final, le déploiement du produit final, la formation des opérateurs pour savoir utiliser le produit final, le support du produit final, et la retraite du produit final (recyclage).

[3] Les produits capacitants éxecutent les processus associés ou les fonctions non-opérationnelles d'un système. En fait, les produits capacitants sont les produits qui permettent d'avoir un produit final, et ils peuvent être utilisés par plusieurs produits finaux.

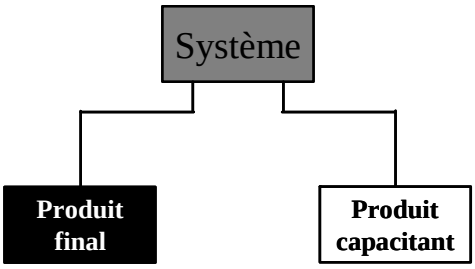

Figure I.6-le système est décomposé entre produit final et produit capacitant

Enfin, l'infrastructure pour le développement agile se fonde fortement sur ce que l'on appelle souvent les problèmes de la logistique, mais en réalité, ce ne sont que les produits capacitants dans le cadre de l'EIA632. Enfin, la structure de décomposition entre : produits finaux et produits capacitants proposée par cette norme, aidera les compagnies et les experts à mieux améliorer leurs processus de l'ingénierie des exigences [BUR-98].

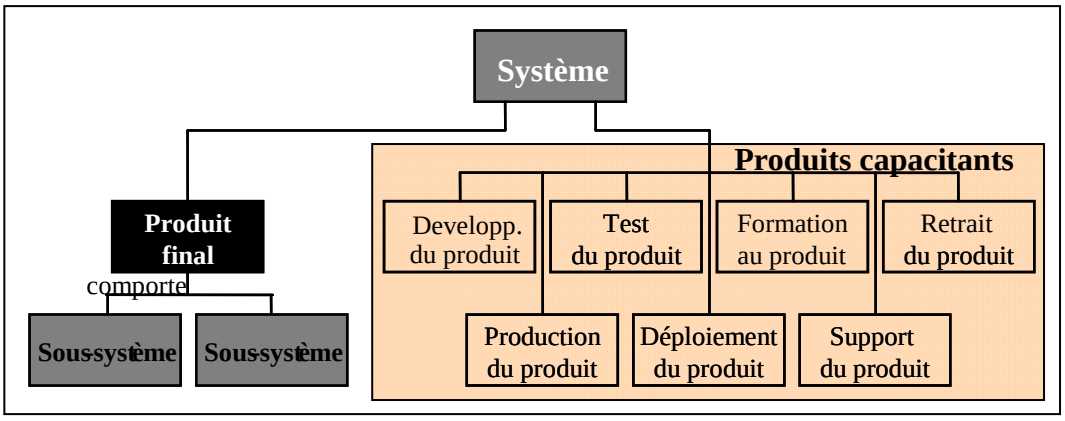

Figure I.7-un module dans le cadre de l'EIA-632

5.2. Processus de l'EIA-632

Après la décomposition du système en produits finaux et produits capacitants (cf. figure I.7), les développeurs obtiennent une hiérarchisation selon plusieurs niveaux (cf. figure II.4). Les processus présentés dans la figure ci-dessus (cf. figure I.5), seront utilisés sur chaque niveau d'hiérarchisation des produits finaux pour les : spécifier, modéliser, et bien sûr pour les développer. Par la suite, nous allons détailler les 5 processus majeurs, pour savoir comment ils réagissent entre eux pour contrôler le développement des produits finaux au sein du cadre de l'EIA-632.

5.2.1. Processus d'acquisition et de fourniture

Le premier processus que nous allons présenter est le processus d'acquisition et de fourniture. Ce processus, sera employé par le développeur pour établir un accord ; afin d'accomplir le travail spécifique, et de faire un contrat. En fait, le processus d'acquisition sera employé par le développeur en agissant en tant qu'acquéreur pour établir un accord avec le fournisseur.

5.2.2. Processus du management

Le processus technique du management doit être employé pour projeter et contrôler les efforts techniques exigés pour satisfaire l'accord établi. Il sera divisé en trois processus : processus de planification, processus de management et processus de pilotage.

- *Processus de planification :* Ce processus est employé dans le but de soutenir la prise de décision de l'entreprise et des différents projets pour préparer les plans techniques nécessaires.
- *Processus d'évaluation :* Le processus d'évaluation est employé pour : (1) déterminer le progrès de l'effort technique vis-à-vis des exigences du projet (2) réviser le progrès du développement.
- *Processus de pilotage :* Le processus de pilotage sera employé pour : (1) contrôler les outcomes venant du processus d'acquisition et de fourniture, du processus de conception, du processus de réalisation, et du processus d'évaluation technique, (2) surveiller les variations du plan et les anomalies pour les exigences (3) distribuer l'information requise et demandée (4) assurer les communications nécessaires.

5.2.3. Processus de conception

Le troisième processus de l'EIA-632, est le processus de conception. Ce processus sera employé pour convertir les exigences convenues de l'acquéreur en un ensemble des produits réalisables, qui satisfont l'acquéreur et les autres parties prenantes. Ce processus est divisé en: le processus de définition des exigences et processus de définition de la solution.

- *Processus de définition des exigences* : Les entrées du processus de définition des exigences sont de deux types : (1) les exigences venant du contrat (2) les exigences venant d'autres processus tels que les plans techniques. Ce processus est utilisé

pour obtenir les exigences techniques du système (Technical Requirements). Ces exigences représentent une description raisonnablement complète du problème qui doit être résolu pour fournir un ensemble de produits finaux dans le but de répondre aux exigences de l'acquéreur et des autres parties prenantes. Parfois, ce processus sera incomplet dans le cas où il y aura un changement du contrat, ou par l'identification d'autres exigences de la part des parties prenantes qui affectent la conception de produits, ou contraignent autrement l'effort technique exigé du nouveau système.

- *Processus de définition de la Solution* : Le processus de définition de la solution est employé pour obtenir une solution acceptable de conception pour un produit final. La solution proposée doit satisfaire : (1) les exigences techniques de système provenant du processus de définition des exigences, (2) les exigences techniques dérivées de ce même processus.

5.2.4. Processus de réalisation

Le processus de réalisation des produits finaux est employé pour convertir les exigences définies et spécifier après le processus de conception en produit final vérifié ou en un ensemble de produits finaux selon les exigences du contrat. A son tour, ce processus est divisé en processus d'implémentation et processus de transfert vers l'utilisation.

- *Processus d'Implémentation* : Le premier but de ce processus est la transformation de la solution caractérisée de conception en ensemble des produits, puis les intégrer, tout cela dans le but de répondre aux exigences définies après l'exécution du processus d'Acquisition et de fourniture.
- *Processus du transfert pour l'utilisation* : Après l'implémentation des produits finaux, il faut exécuter le processus de transfert vers l'utilisation, qui a comme conséquence la livraison des produits aux destinataires appropriées, dans les conditions exigées pour les mettre à la disposition de l'acquéreur, et pour l'usage des utilisateurs.

5.2.5. Processus technique d'évaluation

Il ne faut pas confondre ce processus, et le processus présenté dans la partie management. En fait, le processus technique d'évaluation est prévu pour être appelé par plusieurs processus,

et il sera décomposé en quatre processus: analyse des systèmes, validation d'exigences, vérification du Système, validation de produit final.

- *Processus d'analyse système :* L'analyse de systèmes est employé pour: (1) fournir une base rigoureuse pour aider à la prise des décisions techniques, la résolution des conflits entre les exigences (2) déterminer le progrès en satisfaisant le système technique (3) gérer les risques (4) s'assurer que toutes les décisions seront prises seulement après une étude sur le coût, le programme, et les effets du risque durant le développement ou redéveloppement (re-engineering) du système. C'est à ce niveau, que nous exécutons le processus d'analyse d'impact sur la sécurité lors d'une demande de changement.

- *Processus de validation des exigences :* La validation des exigences est critique pour la réussite du système de développement des produits finaux et pour leur implémentation. On pourra dire que les exigences seront validées quand on est sûr et certain que l'ensemble des exigences soumises correspond aux produits résultants du système, tout en appliquant un ensemble de tests.

- *Processus de vérification des exigences :* Le processus de vérification des exigences du système est employé pour être sûr que : (1) la solution de conception est cohérente avec la source des exigences; (2) les produits finaux à chaque niveau du système durant la phase ascendante (la phase de réalisation des produits finaux après qu'on a terminé la phase descendante qui est la phase de spécification) répondent à leurs exigences spécifiées.

- *Processus de validation des produits* : Le processus de validation des produits finaux présentés dans la structuration du système, aide à s'assurer que les produits finaux répondent aux exigences des acquéreurs et aux exigences techniques présentées dans le processus de la définition.

6. INGENIERIE DES EXIGENCES

Le marketing des nouvelles technologies, devient de plus en plus un problème pour les commerçants, lorsque les clients ne trouvent pas ce qui correspond à leurs demandes. Par exemple, la majorité des compagnies dans le marché des téléphones portables se focalisent sur les exigences les plus intéressantes pour leurs clients. Parmi les exigences d'un client on peut citer: le poids, les fonctions du portable, l'autonomie de la batterie et le prix. De la même

manière, les compagnies dans d'autres domaines comme l'électronique, l'automatique, la pharmaceutique, et le génie logiciel ont défini que les exigences doivent être complètement « comprises » avant le développement du produit.

D'après les différents problèmes rencontrés (volatilité, modification, expression des exigences, etc…) durant le développement, les industriels ont trouvé la nécessitée d'élaborer une étude comme l'ingénierie d'exigences [SAH-02b], pour couvrir la gestion d'exigences durant tout le cycle de vie du développement de systèmes (de l'élicitation jusqu'à la validation).

En fait, l'ingénierie des exigences permet la production des spécifications. Une spécification peut contenir un nombre très varié des exigences (de 50 jusqu'à quelques milliers). Pratiquement, une spécification est nécessaire pour rendre les exigences plus cohérentes, les maîtriser et les gérer au cours de développement du système. Les travaux de L.Hellouin [HEL-01], avaient pour objectif d'élaborer une démarche qui permette d'améliorer le contenu des spécifications relatives aux systèmes embarqués.

La plupart des travaux académiques sur l'ingénierie des exigences ont principalement porté sur la modélisation, et l'expression des exigences et très peu sur les problèmes émergents dus à la nature complexe des systèmes. En fait, la méthodologie développée par [HEL-02], était de s'assurer que les exigences sont déclinées, allouées, suivies, satisfaites (en totalité ou partialement ou pas du tout), vérifiables, vérifiées et justifiées. Ci-dessous, nous allons présenter les objectifs de l'ingénierie des exigences, les caractéristiques des exigences et la description du SRS (le document contenant la spécification des exigences du système).

6.1. Les objectifs de l'ingénierie des exigences

Ci-dessous, nous allons présenter les objectifs des processus de l'ingénierie d'exigences:

- L'identification des parties prenantes,
- L'éliçitation des exigences,
- L'analyse des exigences
- La formalisation des exigences,
- La vérification/ validation des exigences,
- La modification des exigences.

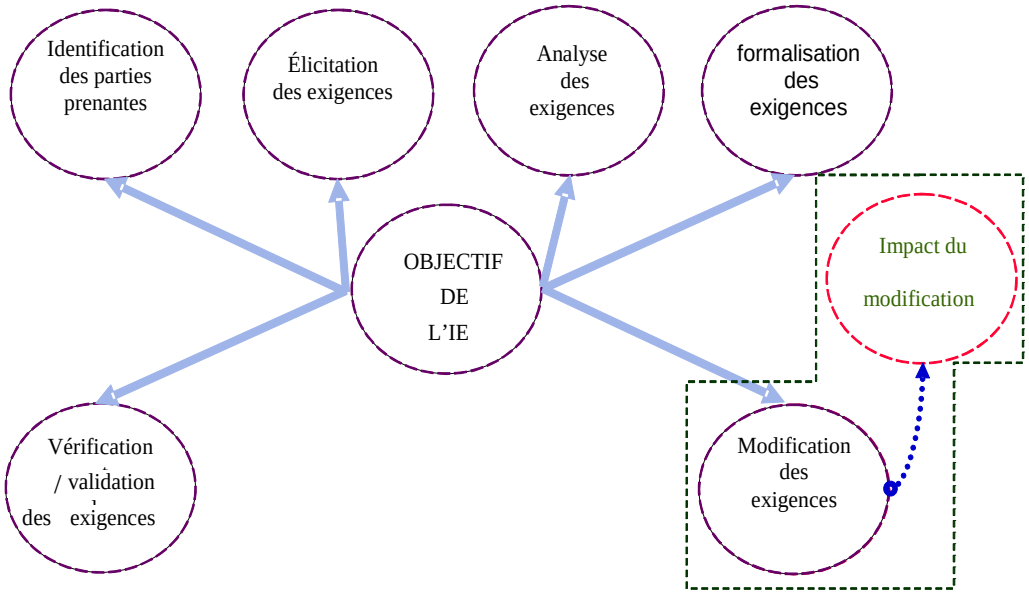

Figure I.8-Les objectifs de l'ingénierie des exigences

6.1.1. L'identification des parties prenantes

Selon [GIB-04], on remarque qu'il faut identifier les parties prenantes du système avant de le développer, C'est à dire autrement la connaissance des personnes liées au développement du système. Alors, lorsqu'on commence à exécuter le processus d'identification des parties prenantes, on répond explicitement à la question suivante: Qui fait Quoi ?

Plusieurs définitions existent dans la littérature pour expliquer la signification des parties prenantes, mais pour faciliter la compréhension on peut dire que : les parties prenantes signifient l'ensemble des personnes ou des groupes qui sont mises en relation avec un produit au cours de son cycle de vie. Parmi les parties prenantes on peut rencontrer: le développeur, le vérificateur, le concepteur, les clients, le fournisseur, etc.

6.1.2. L'élicitation des exigences

Après l'identification de toutes les parties prenantes, il faut exécuter le deuxième processus de l'ingénierie d'exigences qui est le processus d'élicitation d'exigences[4] [SAH-02a].

La majeure partie de l'effort dans le cycle du développement des systèmes n'entre pas dans le processus d'écriture d'exigences (vue la diversité des technologies existantes), mais dans le

[4] Eliciter : expliquer, capturer, rendre explicite, et permettre la compréhension de l'information.

processus de découverte, en essayant de répondre à la question “qu'est ce que doit faire le système?” et non pas “comment faut-il le faire ?”.

Ce processus de découverte et d'élicitation des exigences est souvent vu par beaucoup de chefs de projets comme trop laborieux, trop long, ou trop difficile à entreprendre. Il est quelque fois négligé comme si les exigences sont déjà présentes, et qu’il suffit de les exprimer. Mais, si les exigences sont intangibles, contradictoires, ou imprécises, il devient très difficile d’estimer le temps qu'il faut dépenser durant la conception et l'implémentation.

Actuellement, plusieurs approches existent pour faciliter l’exécution de ce processus. Les travaux de C.Coulin [COU-05a] [COU-5b], ont porté sur cet aspect de manière spécifique et le développement d’un outil correspondant, MUSTER [COU-05c]. Mais ce n’est pas un processus automatique, et les exigences ne seront présentées qu’avec le langage naturel. La plupart des chercheurs pensent que ce processus doit obligatoirement contenir deux types de méthodes : Brainstorming (séance de réflexion) et Interview.

A) Brainstorming

La première méthode d’élicitation des exigences, est la séance de réflexion ou brainstorming. En effet, cette méthode est une manière efficace d'inventer, car c’est un moyen pour obtenir des idées intéressantes, et claires. Pour que la session soit efficace, la plupart des chercheurs dans le domaine de l’ingénierie des exigences indiquent qu’il faut d’abord préciser l’objectif de la session [ROB-04], puis regarder des exemples des produits semblables.

Durant la session il faut numéroter les idées pour que les participants puissent se référer à elles facilement, les expliciter, chaque idée de n’importe quel participant peut être intéressante, mais il faut toujours que les idées servent l’objectif final de la session. Dans [ROB-04] ils ont précisé l’importance d’utilisation des tableaux, des papiers, la numérotation de toutes les idées. Et, ils ont suggéré que dans ce type de session, parfois l’écriture des idées sur des tableaux sera beaucoup plus efficace que l’utilisation des écrans d'ordinateurs, car il est plus intéressant que chaque participant puisse regarder toutes les idées produites.

B) Interview

La deuxième méthode d’élicitation des exigences est les interviews. Cette méthode est très répandue dans le milieu industriel, car l’application de cette méthode n’exige pas beaucoup de temps comme la séance de réflexion présentée ci-dessus, vu que la plupart des personnes qui

utilisent cette méthode sont des spécialistes d'analyse commerciale ou des ingénieurs des exigences qui sont déjà familiers avec ce type de méthode.

6.1.3. L'analyse des exigences

Après l'exécution du processus d'élicitation. Il faut lancer le processus d'analyse d'exigences en étudiant l'ensemble des exigences soumis par les parties prenantes pour les analyser, les compléter, les arranger, les tracer, et pour éliminer l'ambiguïté et la redondance entre les exigences [MAC-01]. Tous cela pour obtenir à la fin un document cohérent, qui sera présenté en langage naturel (contenant parfois des diagrammes d'UML[5]). En fait, ce document définit les exigences que devra respecter le système qu'il faut réaliser selon le point de vue du maître d'ouvrage[6], mais après une validation par le maître d'œuvre[7].

La spécification présentée dans ce document définit complètement le problème auquel les activités de conception devront apporter une solution, son importance est capitale car elle engage directement la quasi-totalité du coût du système. C'est souvent à ce niveau, que le projet joue un rôle en termes de rentabilité pour le maître d'œuvre, et en termes de satisfaction pour le maître d'ouvrage. Les échecs au niveau du projet sont le plus souvent dus à une spécification non figée conduisant à des nombreuses remises en cause. Les échecs au niveau de la mise en œuvre sont presque toujours dus à une spécification inadéquate, et à une mauvaise exécution de ce processus.

Enfin, ce processus est le processus le plus critique pour tout au long du cycle du développement du système. Ce n'est pas du point de vue sécurité qu'il est critique, mais c'est à ce niveau qu'il faut détecter la majorité des erreurs de développement. En fait, après ce processus, les développeurs s'engagent pour lancer effectivement le développement du système. Alors, on peut en conclure que, plus la qualité de ce processus sera prise en considération plus la qualité du produit sera améliorée.

[5] DOORS contient une intégration du diagramme de cas d'utilisation.
[6] Entité ou des personnes qui demandent des services.
[7] Le maître d'œuvre est celui qui organise la réalisation et la conçoit techniquement, coordonne la réalisation, contrôle le résultat, et prépare l'exploitation.

6.1.4. La formalisation des exigences

Le processus de formalisation est exécuté afin de donner une solution pour l'ensemble des exigences soumises par les parties prenantes. Au cours de ce processus, il faut toujours que nous posions la question suivante "Qu'est ce que nous sommes en train de faire ?" Après la phase d'analyse il faut donner un modèle pour le système, en répondant exactement aux exigences du produit à développer.

En effet, une exigence peut être décrite par de nombreux modèles. Par principe une exigence peut être satisfaite par plusieurs technologies. Mais, elle n'a pas la même signification pour tous les métiers. Par exemple, pour le métier logiciel les développeurs ont développé des représentations formelles basées sur des notations mathématiques (Z, B, VDM, RAISE). Par contre, dans d'autres domaines comme celles de l'électricité et de l'électronique, on trouve l'utilisation des schémas électroniques (Max+, Orcad), et pour les systèmes temps réel les développeurs et les analystes proposent d'autres solutions [SCH-91].

En effet, la notation du modèle est très importante, pour que l'équipe de réalisateurs, les constructeurs puissent comprendre et contribuer facilement au contenu du modèle. Actuellement, au cours du développement des systèmes on rencontre de plus en plus des représentations graphiques pour les modèles. En fait, les représentations graphiques (diagrammes d'UML, data flow diagram) du système doivent être assez rigoureuses pour que les réalisateurs puissent comprendre précisément ce qui est nécessaire et ce qu'il faut faire.

Les discussions à propos du meilleur modèle d'un système ou de la meilleure solution, sans avoir une bonne idée sur le genre de défi, sont stériles. Egalement, les bonnes représentations graphiques (diagrammes UML ou autres représentations) qui ne répondent pas aux exigences spécifiques (présenter après l'exécution du processus d'analyse) ne seront jamais des solutions.

Généralement, un modèle peut se baser sur plusieurs vues : Vue opérationnelle, décrivant le niveau des services rendus aux utilisateurs et les conditions d'utilisation et d'exploitation du système. Vue fonctionnelle, décrivant ce que doit faire le système pour exécuter sa mission en répondant aux exigences opérationnelles, c'est-à-dire les fonctions et leurs performances. Et finalement, une Vue physique, traduisant les contraintes techniques imposées au système, notamment les caractéristiques physiques et les interfaces avec l'environnement naturel technique et humain, cette vue apparaît dans les phases avancées de la modélisation.

6.1.5. La vérification/ validation des exigences

L'existence d'une variété de définitions pour la vérification et validation des exigences, nous oblige à préciser que la vérification est employée pour vérifier que la conception corresponde aux exigences du système, et que l'implémentation corresponde à son tour à la conception développé et proposé par les développeurs. Alors le processus de la vérification est incorporé entre les différentes phases du développement du système. Par contre, la phase de validation met en jeu l'implication des parties prenantes (utilisateurs, exploitants, donneur d'ordre, chef de projet, équipe d'analyse système) pour valider les exigences après la réalisation du système prévu par rapport aux attentes utilisateurs, aux contraintes du programme du maître d'ouvrage, aux contraintes du projet et de l'entreprise, aux contraintes externes.

6.1.6. La modification des exigences

Le but de ce processus est de gérer les demandes de modifications, en cherchant les exigences impactées avec la demande, quelle partie de conception (Implémentation) sera impactée ? Comment établir une étude d'impact de la demande ? Et comment intégrer la demande dans le cycle de développement ?

6.2. Description du SRS

Le SRS (System Requirement Specification) résulte de l'exécution du processus d'analyse des exigences (§6.1.3). Il contient la spécification technique des exigences. Parfois, l'exécution de l'ensemble des processus de l'ingénierie d'exigences produit des documents difficiles à comprendre, trop longs et peut être qu'ils seront ignorés par les développeurs du système. C'est pour cela que le SRS doit avoir certaines caractéristiques, comme: des buts testables, ou des exigences testables.

6.2.1. Buts testables

Pratiquement, les buts sont un dispositif pour diriger tous types de projet. Sans buts mesurables, il est trop facile de dépenser le temps sur des sujets qui ne sont pas prioritaires. Et, c'est trop facile d'oublier les valeurs essentielles du projet. En fait, il faut que chaque projet contient des buts clairs et testables, pour que les parties prenantes puissent indiquer si un but a été atteint ou non.

6.2.2. Exigences testables

Une exigence qu'on ne peut pas tester, n'est pas une exigence. Par exemple, si parmi les exigences il en existe une qui dit "le produit sera facile à utiliser," il faut trouver une manière de la rendre testable. Toutes les exigences n'ont pas la même priorité pour une organisation. Ceci rend intéressant de savoir la valeur de chaque exigence avant de décider de l'implémenter ou non. Dans Volere [TEM-Vol], on trouve que chaque exigence doit avoir un critère (fit criterion). Ce critère, est un moyen pour aider les parties prenantes à tester l'exigence, et de déterminer avec précision si la solution a été atteinte ou non.

6.3. Caractéristiques des exigences

Les exigences sont un moyen de communication des différentes parties prenantes entre elles. Le terme exigence signifie la connaissance des fonctionnalités et des qualités que doit avoir le produit qu'on est en train de développer.

6.3.1. Différents types d'exigences

Les exigences fonctionnelles expriment les fonctions attendues par le système. Par contre, les exigences non fonctionnelles, sont une traduction des contraintes globales de qualité de service ou d'aptitude du système (fiabilité, opérabilité, conviviabilité), ou des contraintes opérationnelles (conformité à des normes d'utilisation), ou des contraintes de conception (réutilisation d'existant). Dans la figure ci-dessous (cf. figure I.9), on montre la décomposition des exigences entre exigences fonctionnelles[8], et d'autres non fonctionnelles[9] [GIO-05].

Particulièrement, dans l'industrie de génie logiciel [JAC-99] on peut constater que la plupart des équipes de développement sont excessivement concentrées sur ce qu'elles appellent fonctionnalité, ou les exigences fonctionnelles.

[8] Une exigence fonctionnelle exprimant une fonction attendue par le système.
[9] Les exigences non fonctionnelles sont les propriétés comportementales que les fonctions indiquées doivent avoir.

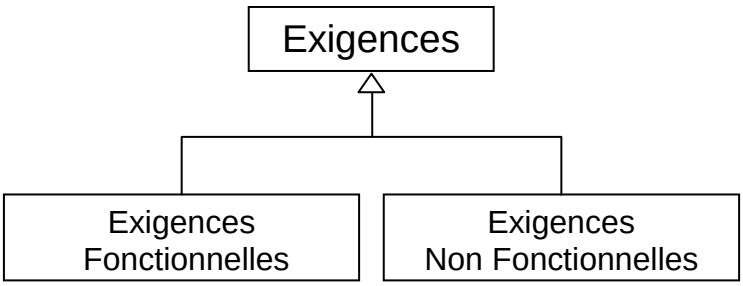

Figure I.9- Décomposition des exigences

Dans chaque SRS, existe une section concernée pour les exigences non fonctionnelles [CYS-03]. Cette section sera négligée si elle contient des mots trop vagues, et difficiles à comprendre. Ce qui nous explique pourquoi les exigences non fonctionnelles seront ignorées par les parties prenantes la plupart du temps. Car ils considèrent qu'elles sont non mesurables et peu claires [BOH-96a].

6.3.2. Liens entre les exigences

Dans ce paragraphe, nous présentons les différents types des liens qui existent entre deux exigences différentes.

A) Dépendance et déclinaison

Deux types de liens existent entre deux exigences différentes : lien de dépendance, lien de déclinaison. Le lien de dépendance n'est pas pour lier des exigences de niveaux de décomposition différentes, mais des exigences d'un même niveau qui sont soit complémentaires, soit analogues, ou similaires. Ce type de lien facilite l'analyse de l'impact (également pour le lien de décomposition). Par exemple, si on a une relation(R) de dépendance entre E1 et E2 on peut présenter cette relation avec : (E1) R (E2). Et dans le cas où on a une évolution sur E1, par l'existence de ce lien on pourra savoir qu'il y aura un impact sur E2 (l'inverse est vrai).

Dans la figure ci-dessous, on présente les deux types de lien différents : les liens de dépendances et les liens de déclinaisons (décomposition). En fait, on peut distinguer les liens de dépendance entre E1.2 et E2.1 (le deuxième est présenté entre E1.1 et E2.3). Par contre, le lien de décomposition par exemple est présenté entre E1 et {E1.1, E1.2}

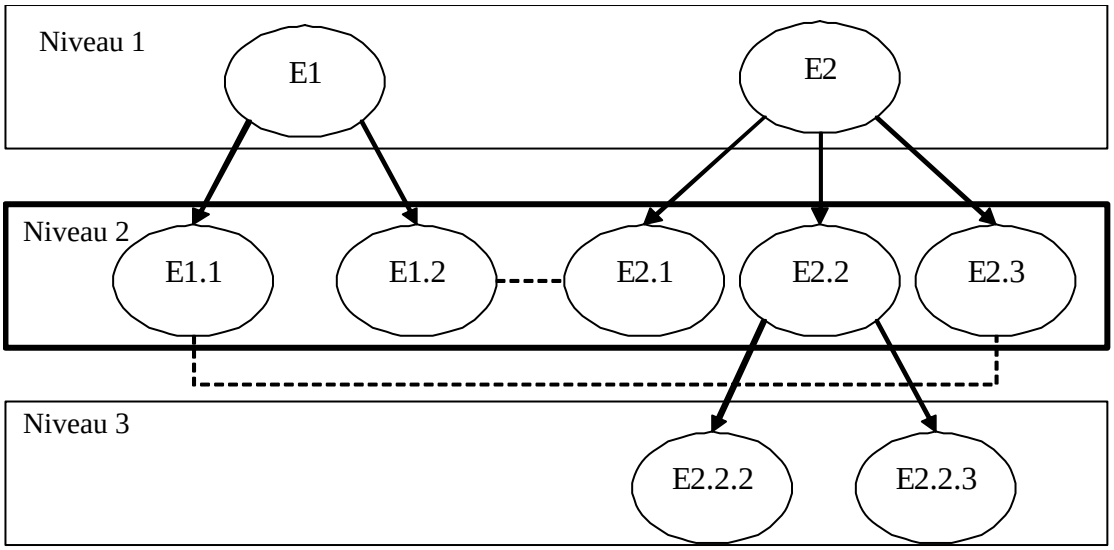

Figure I.10- Liens entre les exigences.

B) Lien d'évolution

Disposer des différentes versions d'une exigence, c'est connaître son origine, les faits manquants qui sont à l'origine, C'est mieux comprendre le besoin et parfois sa formulation. Le schéma suivant montre l'évolution d'une exigence. Par exemple, l'évolution d'une exigence peut être repérée par un numéro de version (cf. figure I.11)

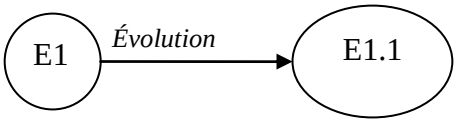

Figure I.11- Lien d'évolution, après l'évolution de E1 on obtiendra E1.1

7. IMPACT DU CHANGEMENT

Une enquête récente, faite par Customer Focus Group (CFS) [GRO-Foc] dans la fabrication des avions, présente comment les commerçants s'intéressent à intégrer de telles nouvelles technologies, plutôt que d'intégrer d'autres qui pourront avoir plus d'avantages, mais qui n'assurent pas la sécurité. Dans la même enquête, ils présentent comment une nouvelle technologie n'affecte pas la performance dans un tel domaine peut avoir un impact sur la performance dans une autre application.

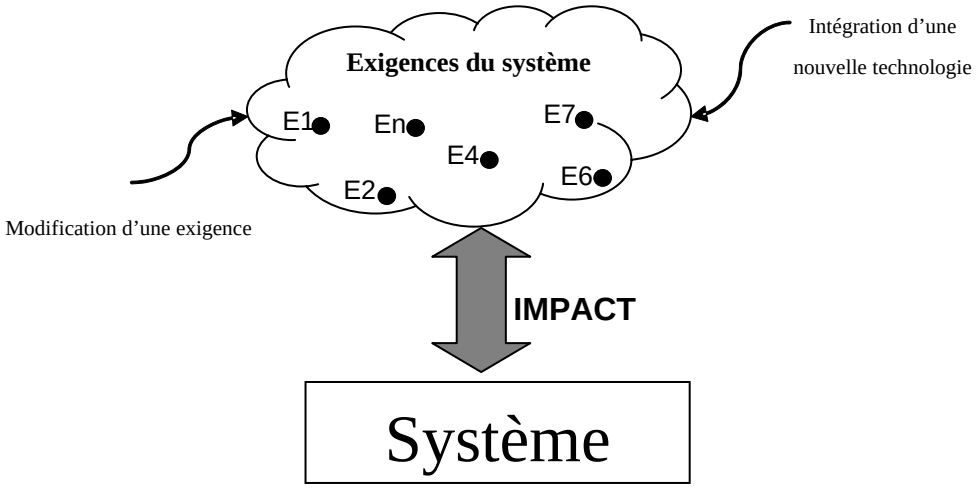

Figure 1.12- Impact du changement sur le système

Par définition, un système est en sécurité [LEV-03a] s'il est libre de tout accident inacceptable. En fait, dans notre étude on se concentre à étudier l'impact du changement sur la sécurité du système. Pour atteindre ce but, on intègre la notion des exigences de sécurité dans les systèmes qui peuvent être présents dans un cas critique pour son environnement. Cette nouvelle notation sera la base de la recherche d'impact lors d'une demande d'évolution.

8. CONCLUSION

Ce chapitre a été divisé en deux parties majeures. Dans la première partie, nous avons présenté que la majorité des études ont porté sur des études statistiques pour l'analyse de l'évolution (changement), en élaborant des études sur les causes et les effets de la volatilité. Dans la deuxième partie, nous avons présenté notre contexte d'étude : l'ingénierie système et l'ingénierie des exigences. Pour l'ingénierie système nous avons précisé les différents processus existants dans le cadre de l'EIA-632. Pour l'ingénierie des exigences, nous avons présenté ces objectifs, en présentant le processus : d'identification des parties prenantes, d'élicitation des exigences, d'analyse des exigences, de formalisation des exigences, de vérification et de validation pour les exigences, et de modification des exigences.

Dans le chapitre suivant, nous allons présenter l'état de l'art. Au sein du deuxième chapitre, nous présenterons les différentes approches et études qui ont été développées pour la recherche d'impact lors d'une demande de changement. Puis, nous montrerons les différentes techniques pour l'analyse de la sécurité, la traçabilité et la méthode formelle que nous allons utiliser pour pouvoir analyser l'impact de la demande de changement sur la sécurité.

CHAPITRE II

ETAT DE L'ART ET ANALYSE CRITIQUE

1. INTRODUCTION

Lorsque les parties prenantes exigent une demande de changement au cours du développement du produit, deux possibilités vont se présenter: la première possibilité est d'incorporer la demande au cours du développement (cette possibilité exige beaucoup de ressources), la deuxième possibilité est de reporter la demande dans un autre projet, pour qu'il soit inclus dans une nouvelle version ou totalement rejeté.

Cependant, l'application du changement implique une analyse d'impact. L'analyse d'impact est utilisée avec différentes formes pour contrôler les changements. L'analyse d'impact, comme définie par Arnold et Borner "est l'activité d'identification de ce qu'il faut modifier pour accomplir un changement, ou d'identification des conséquences potentielles d'un changement" [ARN-93].

Dans notre cas d'étude on considère le cas où nous allons incorporer la demande de changement au cours du développement du produit. Pour cela, nous voulons élaborer une méthode, afin de contrôler la demande et analyser son impact.

Par la suite de ce chapitre, nous allons présenter les différentes approches qui ont été développées dans le cadre de la recherche d'impact, la définition de la sécurité, deux normes différentes pour la sécurité, un modèle de traçabilité, les différentes méthodes d'analyse de la sécurité, et la spécification formelle qui nous permette d'analyser l'impact du changement sur la sécurité.

2. APPROCHES DE LA RECHERCHE D'IMPACT

Les changements d'exigences représentent des risques pour le succès et l'accomplissement d'un projet. Il est critique pour un chef de projet de déterminer l'impact du changement des

exigences. Les activités de base de changement pour n'importe quel type de système et particulièrement les logiciels [ARN-96], peuvent être simplifiées par: l'initialisation du changement, l'application du changement, et le test du changement.

Ci-dessous, nous allons présenter trois approches pour l'analyse d'impact du changement. La première approche est basée sur un ensemble de processus, la deuxième approche est basée sur un modèle de traçabilité qui sert à chercher l'impact métrique du changement, et l'approche basée sur l'utilisation de la spécification intentionnelle [NAV-01b].

2.1. Processus d'Analyse

L'analyse d'impact du changement au cours du développement de logiciels connaît une attention considérable [KOT-98]. Dans [BOH-02] on remarque le besoin croissant d'identifier les conséquences des demandes de changements. Le modèle présenté dans ce travail (cf. figure II.1), décrit un processus de changement de logiciel qui intègre l'analyse d'impact.

Dans son approche, il a intégré le processus d'analyse d'impact lors d'une demande de changement dans la phase de la maintenance ; puisqu'il suppose que les conséquences de la demande sont détectées dans un logiciel déjà existant. En effet, il a employé la méthode d'analyse fonctionnelle et structurée (SADT) pour présenter son approche.

Les activités primaires du processus de la demande de changement présentées dans son approche impliquent : (1) la gestion de la demande de changement (2) la compréhension de la demande, (3) la spécification et la conception de la demande, (4) l'implémentation de la demande et (5) finalement la mise en œuvre d'un ensemble de tests sur le changement et sur le logiciel après intégration de la modification.

En fait, la gestion de la demande contrôle la séquence des activités de la demande, tout en respectant le but de la demande de changement, les contraintes, et les ressources allouées pour le changement. La compréhension de la demande de changement sert à évaluer l'impact de la demande de changement proposée, et à déterminer les effets du changement. Par contre, la spécification et la conception de la demande ont pour but de spécifier la demande au niveau du logiciel. La phase de l'implémentation de la demande doit gérer l'application et l'intégration de la demande de changement et la mise à jour de toute la documentation du logiciel.

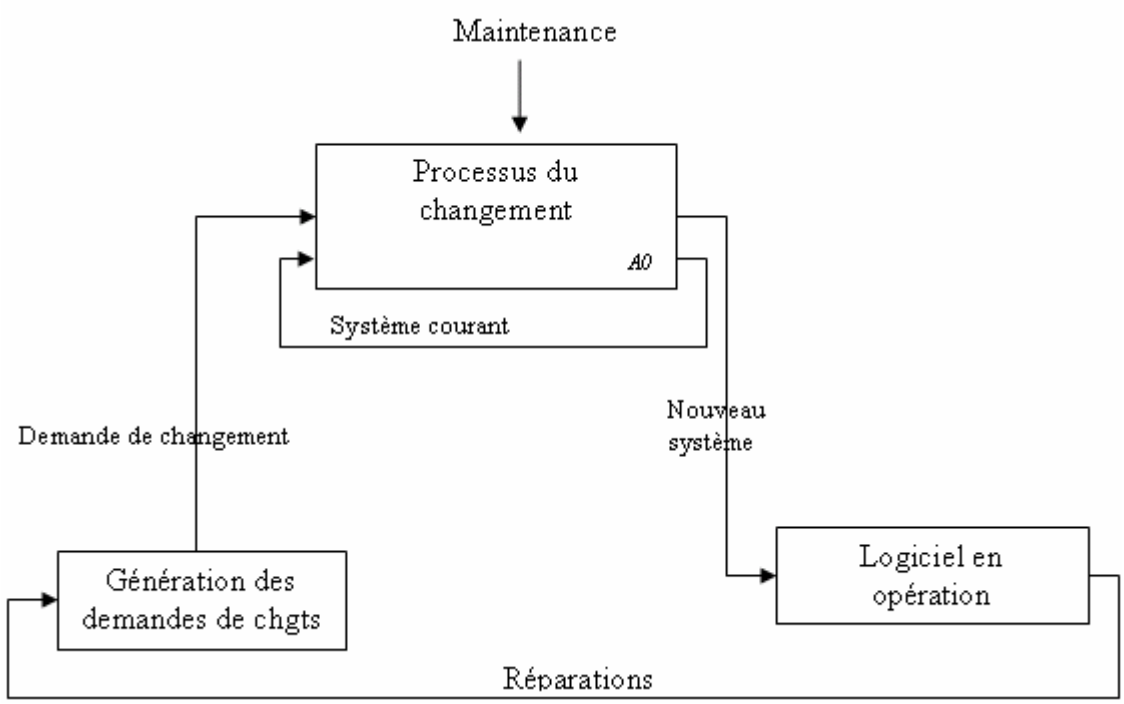

Figure II.1-La recherche de l'impact du changement dans cette approche est basée sur un ensemble des processus.

Après l'implémentation de la demande de changement, Boher [BOH-02] indique la nécessité d'appliquer un ensemble de test de changement, afin d'assurer la bonne intégration des nouvelles exigences dans le logiciel.

2.2. Analyse basée sur un modèle de traçabilité

La deuxième approche d'analyse d'impact lors d'une demande de changement [STE-03], concerne l'analyse d'impact métrique. Pour prévoir l'application du changement, la méthodologie présentée dans [STE-03] est basée sur la traçabilité de l'information afin d'analyser l'impact métrique du changement. Cette méthodologie sert à calculer l'impact métrique du changement, en se basant sur des algorithmes calculant les efforts, et la complexité du développement dans chaque phase lors de l'application du changement. Sa méthodologie est appelée TIAM (trace-based impact Analysis). En effet, TIAM a deux caractéristiques principales: premièrement TIAM est une méthodologie basée sur la traçabilité de l'information, deuxièmement TIAM intègre des attributs permettant la quantification du travail sur le produit (Work-products) dans chaque phase. En fait, ces deux caractéristiques

nous permettent d'analyser l'impact en déterminant l'impact métrique du changement. Pour établir sa méthodologie (TIAM), il a défini le modèle WORM[10] (cf. figure II.2).

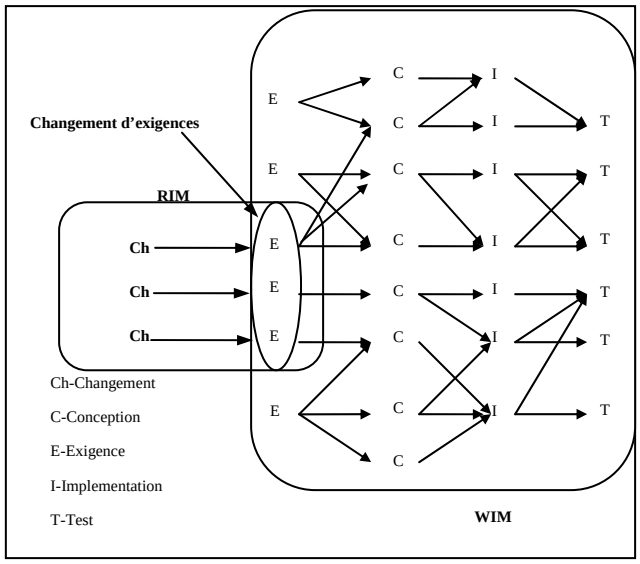

Figure II.2-le modèle WORM est composé du modèle WIM et du modèle RIM

En fait, WORM comprend 2 modèles : WIM et RIM. Le modèle WIM[11] est formé des nœuds (n), des traces entre les nœuds (T), la définition de l'influence (i) entre les nœuds en quantifiant les traces entre trois niveaux (faible, moyenne, forte), la complexité (c) dans chaque nœud, l'effort(e) effectué dans chaque phase, et la phase (p). Le modèle RIM[12], contient des informations sur les exigences qu'il faut modifier. Ce dernier est formé d'un ensemble de demandes de changement (ddc), les nœuds d'exigences qu'il faut changer (nc), la trace (T') qui se trouve entre la demande de changement et l'exigence à modifier.

WIM = < n, T, i, c, e, p> **RIM = <ddc, nc, T'>**

Le modèle WORM est la base pour l'analyse d'impact en termes d'effort et du work-product lors de la demande du changement. Après la décomposition du système de développement en WIM et RIM, l'application du TIAM pour la recherche d'impact passe par deux étapes: la première étape est pour calculer l'impact métrique du changement, la deuxième étape est pour la classification des changements d'exigences dans des classes compatibles.

[10] WORM: Work product Requirements trace Model
[11] WIM : Work product Information Model
[12] RIM : Requirement change Information Model

2.3. Spécification intentionnelle

Durant le développement de logiciels, les experts ont tendance à se concentrer d'abord sur l'analyse de la structure fonctionnelle du logiciel, à un niveau élevé d'abstraction. Puis, ils essayent d'optimiser leur recherche d'une solution, en se concentrant sur l'obtention d'un modèle encore plus détaillé. Dans [LEV-00], on distingue sept niveaux de dépendance (cf. figure II.3) pour le développement de logiciels, afin de réduire les difficultés rencontrées au cours de leur développement et pour garantir l'analyse d'impact du changement. Pour qu'ils atteignent leur but, ils ont employés deux niveaux d'abstraction : (1) des abstractions entières où chaque niveau de la hiérarchie représente une agrégation des composants du système vers le niveau plus bas, (2) des abstractions de l'information où les niveaux plus élevés contiennent la même information conceptuelle mais cachent quelques détails c.-à-d., chaque niveau est une amélioration de l'information existante dans le niveau plus élevé.

L'utilisation de cette hiérarchisation, sert à ce que chaque niveau réponde à « quoi faire » dans le niveau successif, tandis que le niveau le plus bas décrit le « comment il faut le faire ». En fait, la distinction entre les niveaux est la suivante :

- Niveau 0 : la gestion du projet, plan de la sécurité,
- Niveau 1 : les buts du système,
- Niveau 2 : les principes du système,
- Niveau 3 : la conception en utilisant les boites noires et SpecTRM-RL comme langage formel,
- Niveau 4 : la conception,
- Niveau 5 : la représentation physique du système,
- Niveau 6 : le niveau d'opération

En effet, les traces entre les niveaux sont de type plusieurs à plusieurs (many to many), c.à.d. les composants des niveaux plus bas, peuvent atteindre plusieurs objectifs, et les buts au niveau plus élevé peuvent être réalisés en utilisant plusieurs composants du modèle plus bas. Ces liens entre les niveaux, peuvent être suivis dans l'une ou l'autre direction, afin d'analyser l'impact lors d'une demande de changement.

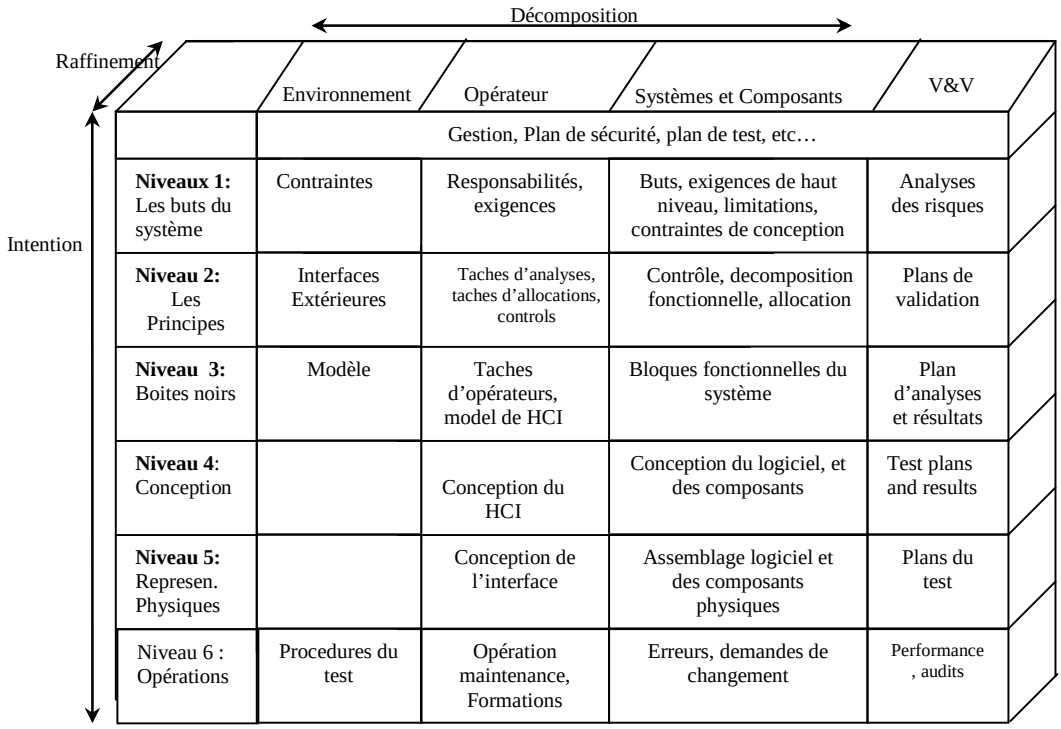

Figure II.3-dans la spécification intentionnelle, le développement du système est formé de 7 niveaux. Dans leur approche, chaque niveau sera une abstraction pour le niveau juste en bas.

En fait, dans le cadre de la spécification intentionnelle [NAV-01a] [NAV-01b], les changements dans des niveaux supérieurs se propageront vers le bas, exigeant des changements dans les niveaux inférieurs. Dans cette étude, le couplage de systèmes durant le développement est divisé en:

- Système désaccouplé : les traces (relations) entre les exigences et la conception sont toutes linéaires (one to one), c.à.d une conception correspond a une seule exigence,
- Système faiblement couplé : les traces (relations) entre les exigences et la conception sont un à plusieurs (one to many), c.à.d plusieurs parties de la conception correspondent à une exigence,
- Système étroitement couplé : les traces (relations) entre les exigences et sa conception sont multiples (many to many), c.à.d plusieurs exigences correspondent à une seule partie de la conception, et une conception correspond pour plusieurs exigences dans le SRS.

Actuellement, c'est quasi-impossible qu'une conception d'un système complexe soit complètement désaccouplée; car elle est généralement étroitement couplée. Pour faciliter l'application du changement, ils ont précisé une méthode pour passer de la conception étroitement couplée vers celle faiblement couplée. En fait, cette méthode réduit la difficulté rencontrée lors de l'application du changement, et facilite son analyse d'impact.

3. LA STRUCTURE SELON L'EIA-632

Pendant le développement d'un système, la démarche est de passer à travers plusieurs phases: l'élicitation des exigences, l'analyse des exigences, la spécification des exigences, la conception, l'intégration et la validation [GHA-02] [GHA-04] [SEK-02]. En effet, dans le cadre de l'EIA-632 [STA-EIA], le développement du système désiré est décomposé avec une hiérarchie des modules. Car, il est trop rare qu'un module[13] simple définisse la solution complète de l'acquéreur, et répond aux exigences des parties prenantes. Car, le développement d'un module exige un développement ultérieur (cf. figure II.4).

Pour la simplicité, on considère que le produit final présenté au niveau supérieur exige une décomposition en deux sous-systèmes. Chaque sous-système sera pris comme un module, qui s'intègre dans le développement du système désiré mais dans un niveau plus bas.

Le développement des modules du bas niveau est lancé dès que le contenu défini du module sera déterminé. Le contenu du module est représenté en tant que produit final qui a des caractéristiques, et des exigences identifiées pour les produits permettant d'avoir ce produit final. Dans la structuration du système dans le cadre de l'EIA632, les modules à plusieurs niveaux forment l'hiérarchisation du système comme démontré dans la figure ci-dessous.

13 Le Module représente un building-block. Il est formé d'un produit final et des produits capacitants.

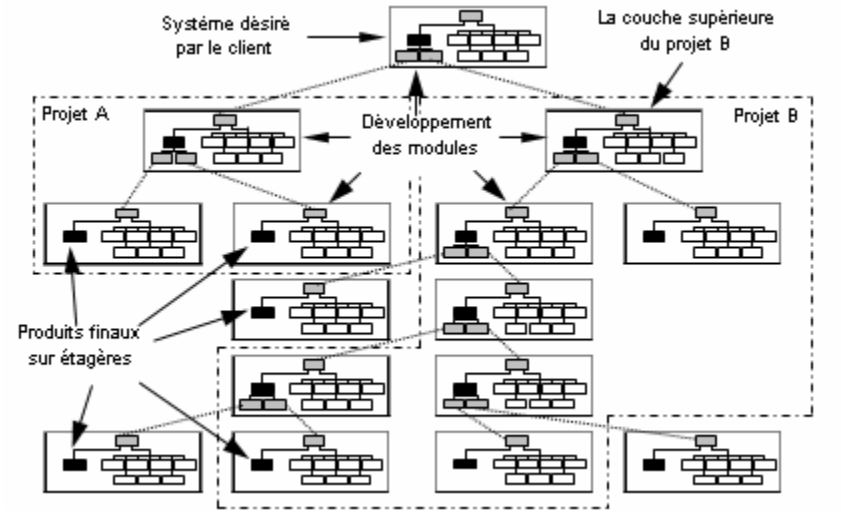

Figure II.4-Dans le cadre de l'EIA-632 chaque système sera divisé en plusieurs module (Building-Block), et chaque module sera formé des produits finaux et des produits permettant la réalisation des produits finaux.

La décomposition du système désiré continue jusqu'à l'identification de trois catégories de produits finaux :

- Produits finaux sur étagère,
- Produits finaux pouvant être implémenté directement,
- Produits finaux pouvant être fournis par un sous-traitant.

3.1. Développement descendant

Dans le cadre de l'EIA-632, le développement d'un système peut être divisé en plusieurs projets. Chaque projet peut avoir plusieurs niveaux de décomposition. En fait, un accord doit être employé pour le développement de chaque module. Dans la figure II.4, on présente un exemple simple pour la structuration du système dans le cadre de l'EIA-632.

En effet, le module racine de la hiérarchie contient le produit final qui doit répondre aux exigences du client primaire (cf. figure II.4). Ce module supérieur représente ce qu'on appelle souvent le projet principal. Dans cet exemple, le projet principal est composé de deux projets: Projet A et projet B. Effectivement, le module supérieur dans chaque projet (A ou B), représente la couche supérieure de développement pour le projet respectif, mais la deuxième couche pour le projet principal (cf. figure II.4). Le projet A est décomposé en deux couches de développement, tandis que le projet B est formé de 5 couches. Les lignes reliant les couches à

deux niveaux différents, reflètent les exigences définies assignées d'un module parent à son module enfant.

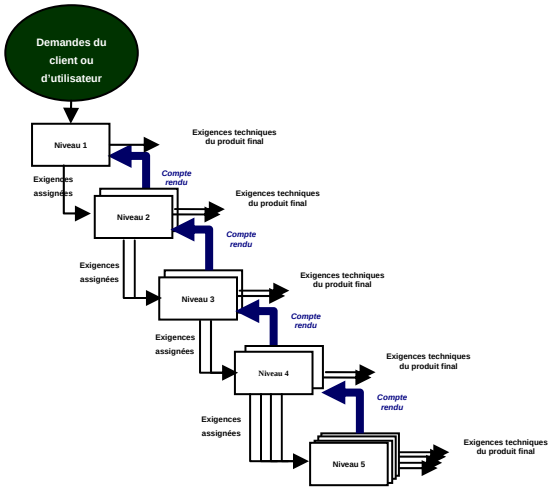

Figure II.5- Développement descendant

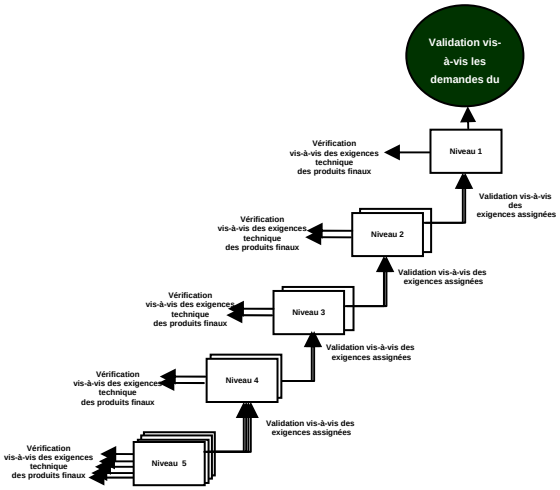

Figure II.6-Réalisation ascendante

Enfin, chaque projet applique les processus de conception et définition (voir chapitre I) à chaque niveau de décomposition (cf. figure II.5) pour chaque module, afin de développer le système approprié.

3.2. Réalisation ascendante

La réalisation ascendante des produits finaux, est représentée dans le schéma ci-dessus (cf. figure II.6). Les produits finaux pour le niveau le plus bas sont implémentés ou fournit par un fournisseur, etc.…puis ils seront vérifiés en exécutant le processus de vérification et de validation qu'on a présenté au premier chapitre. Une fois les produits finaux vérifiés au niveau le plus bas, le passage vers le niveau juste en haut (pour réaliser le module parent vis-à-vis d'un accord déjà établi) sera possible. Et chaque produit final doit être vérifié vis-à-vis de ces exigences avant la délivrance au fournisseur ou bien avant le passage vers le niveau juste au dessus.

4. DEVELOPEMENT SELON L'EIA-632

Le développement du système dans notre cas d'étude est basé sur l'utilisation de l'EIA-632. En fait, nous avons choisi l'EIA-632 vu sa réputation et sa bonne pratique dans le domaine industriel. De nombreux utilisateurs de l'EIA-632 confondent les 5 processus du développement (processus de management, processus de conception, processus de définition,

et processus de validation) des produits finaux avec le développement du système selon l'EIA-632(cf. figure II.4). En effet, chaque projet applique les processus de la conception pour chaque module; afin de développer le système approprié, et particulièrement pour répondre aux exigences des différentes parties prenantes, associées avec un seul module.

Après la décomposition du système dans une hiérarchisation des produits finaux, chacun d'eux a son propre cycle de vie. Les processus de l'EIA632 (présenté dans le chapitre 1) sont applicables pour le développement de chaque produit final présenté dans chaque module. En premier lieu les produits du plus bas niveau existant dans un projet seront réalisés ou fournit par un fournisseur.

Le développement descendant (cf. figure II.5) et la réalisation ascendante (cf. figure II.6) seront corrélés avec un ensemble des phases dans lesquelles les processus de l'EIA-632 seront appliqués, pour former le cycle de vie dans le cadre de l'EIA-632. En fait, le cycle de vie dans le cadre de l'EIA-632 sera décomposé en 5 phases[14] : (1) la phase de prédéfinition nommée conception, (2) la phase de définition (3) la phase de conception du sous-système, (4) la phase du conception détaillée nommées création, (5) la phase d'intégration des produits finaux, de test et d'évaluation nommée réalisation.

5. METHODES D'ANALYSES

Les systèmes critiques existent dans de nombreux domaines tels que le transport, la médecine, les centrales nucléaires. La sécurité est d'une importance primordiale dans de tels systèmes. En effet, beaucoup de recherches ont été faites pour augmenter la capacité des ingénieurs à rendre de tels systèmes sûrs et fiables. Mais, lorsqu'on parle de la sécurité du système, il faut directement réfléchir aux exigences de sécurité [ELJ-04b]. Par ailleurs, la connaissance des exigences de sécurité exige des analyses spécifiques sur le système qui permettent de déterminer ses propriétés et les états critiques du système.

Par exemple, lorsque les ingénieurs développent des logiciels pour des applications critiques, les essais et les vérifications constituent une partie importante du cycle de développement. Avec la croissance de la complexité du système, l'essai et la vérification deviennent de plus en plus importants. Parmi les méthodes traditionnelles utilisées pour l'analyse de la sécurité on peut citer : l'analyse causale.

14 La phase du développement descendante correspond à la phase : conception et création
La phase du ascendante de réalisation correspond à la phase : réalisation

En effet, toutes les tentatives qui ont été faites pour améliorer les techniques traditionnelles pour la sécurité telles que l'analyse d'arbre de défaillances et l'évaluation probabiliste des risques, n'ont pas été très concluantes. Actuellement, le modèle traditionnel de chaîne d'événements avec son analyse sur l'échec des composants est inadéquat pour le développement des logiciels critiques qui contrôlent de plus en plus les systèmes actuels. Car souvent, on rencontre des accidents pour les systèmes qui sont contrôlés automatiquement. Malheureusement, les accidents de tels systèmes viennent d'autres sources et d'autres causes, vu que dans certains cas le logiciel fonctionne exactement comme indiqué, c.-à-d., le logiciel est conforme à ses exigences.

La connaissance des exigences de sécurité, exige plusieurs méthodes et plusieurs types d'analyses. Parmi ces méthodes on cite l'analyse préliminaire du risque (ARP), l'AMDEC, et l'analyse du risque opérationnel.

5.1. Analyse préliminaire du risque

L'analyse préliminaire du risque fournit une première évaluation des risques liés au système. L'APR identifie les risques critiques associés à un tel système, les composants dangereux, les contraintes environnementales, les procédures de secours.

5.2. AMDEC

La méthode AMDEC fournit des informations permettant d'identifier les risques inacceptables, de les évaluer, de les éliminer et de les commander. Alors, l'AMDEC sert à déterminer les conséquences ou les effets de défaillances des sous éléments sur une exploitation du système, et de classifier chaque échec potentiel selon sa criticité. L'AMDEC sera employée pour identifier et analyser les échecs possibles dès la phase de conception ; de sorte que les mesures appropriées soient prises pour éliminer, réduire au minimum, et assurer la sécurité.

L'AMDEC examine le système élément par élément, en utilisant la logique inductive pour évaluer les risques associés à un système. Enfin, l'AMDEC doit passer en revue pour vérifier s'il y aura de nouveaux risques liés au système, lors de l'application des modifications. Cette méthode peut être simplifiée avec les cinq étapes suivantes: le développement de l'organigramme, la réalisation d'une analyse fonctionnelle, l'analyse du mode de défaillance, l'évaluation des risques, l'optimisation.

5.3. Analyse du risque opérationnel

L'analyse de risque opérationnel (ARO) identifie les risques liés au personnel pendant la production, l'installation, l'essai, la formation, et l'opération du système.

6. GESTION DE LA SECURITE

Dans le paragraphe précédent, on a présenté quelques méthodes d'analyse de la sécurité. Ce type d'analyses ne suffit pas sans l'existence d'une gestion de la sécurité. Dans la norme IEC 615-08/11 [STA-IEC] la gestion de la sécurité concerne la gestion des activités nécessaires à assurer que la conception prenne en compte les exigences de sécurité, le maintien des instruments de sécurité durant le cycle de vie du système, et le programme de certification de la sécurité. En fait, la certification de la sécurité est employée pour produire un document formel qui assure l'intégration de la sécurité durant le cycle de vie du système.

Dans un cas d'étude pour les systèmes de transport [ADD-00] [Site 2], le plan de gestion pour la sécurité est désigné pour éliminer et contrôler les risques durant le développement du système. Dans cette étude, on remarque que les risques qui ne peuvent pas être éliminé avec la conception, il faut les contrôler en fournissant des composants pour les détecter, les éliminer, et atténuer leurs effets.

Enfin, durant le développement des systèmes critiques, les analyseurs développent le plan de gestion pour la sécurité, dans le but de suivre les exigences de sécurité et les instructions de sécurité fournissent après l'exécution des différents processus d'analyse de sécurité durant les différentes phases du système. Par la suite, nous allons présenter les processus d'analyse de sécurité et leurs occurrences durant le développement des systèmes, la norme de DOD, et la norme de l'IEC.

6.1. Processus d'analyses et leurs occurrences

Dans un cas d'étude pour le développement des systèmes, on peut remarquer l'occurrence des différentes méthodes d'analyse durant le cycle du développement (cf. figure II.7). Au début, l'analyse du risque préliminaire (ARP) est exécutée durant la première phase du développement du système et ça continue durant la phase de la conception préliminaire. La méthode AMDEC est exécutée au début de la conception préliminaire. En effet, l'analyse d'arbre de défaut (FTA) est exécutée durant la conception préliminaire et jusqu'à la fin de la

conception finale. Et finalement, l'analyse de risque opérationnelle (ARO) est préparée pendant la dernière partie de la conception finale.

La phase de construction/installation commence avec la fabrication ou la construction, et elle se termine par l'installation, et les tests des unités construits qui constituent le système. L'AMDEC doit être mis à jour si des risques additionnels sont identifiés pendant cette phase. La phase d'intégration/test/certification de sécurité, commence quand l'équipement est installé, et se prolonge à toute la période d'essai.

Analyse du risque	**Système de transport**				Opération/ Intégration/ Test/ Certification de la sécurité
	Planification	Conception préliminaire	Conception finale	Construction/ Production/ Installation	
ARP	▬	▬			
AMDEC		▬	▬	▬	
FTA			▬		
ARO				▬	▬

Figure II.7-L'occurrence des différentes méthodes d'analyses durant le cycle de développement pour un système de transport

L'ARO peut être mise à jour si des risques additionnels sont identifiés pendant cette phase. Ce processus fournit la documentation nécessaire exigée pour la sûreté du système.

6.2. La norme du DOD

Le DOD (département de la défense américain) [STA-Dod] est en charge à la protection des personnels des accidents fatals et des dommages. En fait, le but de la norme est d'identifier les risques, les évaluer et les contrôler. Au sein de la norme, on remarque la nécessité d'appliquer de nombreuses techniques afin d'atteindre un niveau de risque acceptable ; tout en respectant les contraintes du temps, et du coût au long du cycle de vie du système.

Dans la norme, on remarque que le but du système de sécurité est d'assurer la sécurité durant le cycle de vie du système. En fait, lorsqu'il y aura la soumission des modifications, il faut identifier les nouveaux risques associés au système ; pour que chaque risque soit éliminé ou réduit à un niveau acceptable. Enfin, les efforts pour la construction de la sécurité de systèmes doit contenir un ensemble d'exigences pour : la documentation du système de

sécurité, l'identification des risques, l'évaluation du risque, le calcul de la probabilité de leur occurrence, la réduction de leur occurrence, la transformation du risque vers un niveau acceptable, et la vérification de la diminution du risque.

6.3. La norme de l'IEC

La norme de l'IEC-61511[STA-IEC], est concernée par le "cycle de vie de la sécurité" afin de structurer les exigences, de spécifier, de concevoir, d'intégrer, d'opérer, et de modifier le système équipé par la sécurité. En fait, la sécurité comme défini dans la norme est l'atténuation d'un niveau inacceptable du risque qui peut affecter les matériels physiques du système, la vie des personnages qui interactent avec le système. C'est pour cela qu'on identifie la définition des fonctions pour la sécurité. Ce type de fonctions a pour but d'intégrer des fonctions qui ont la capacité d'effectuer des actions nécessaires pour réaliser un état sûr et sécurisé pour maintenir l'équipement sous la commande.

En fait, le cycle de vie de la sécurité présenté dans la norme, est composé de plusieurs phases. Chaque phase (cf. figure II.8) a un ensemble d'entrées et de sorties définies, et vers la fin de chaque phase, un contrôle (ou la vérification) sera effectué afin de confirmer que les sorties exigées sont comme prévu. Alors, le but de la norme est de désigner pour fournir une confiance pour l'opérateur, le régulateur, et de toute personne en relation avec le système.

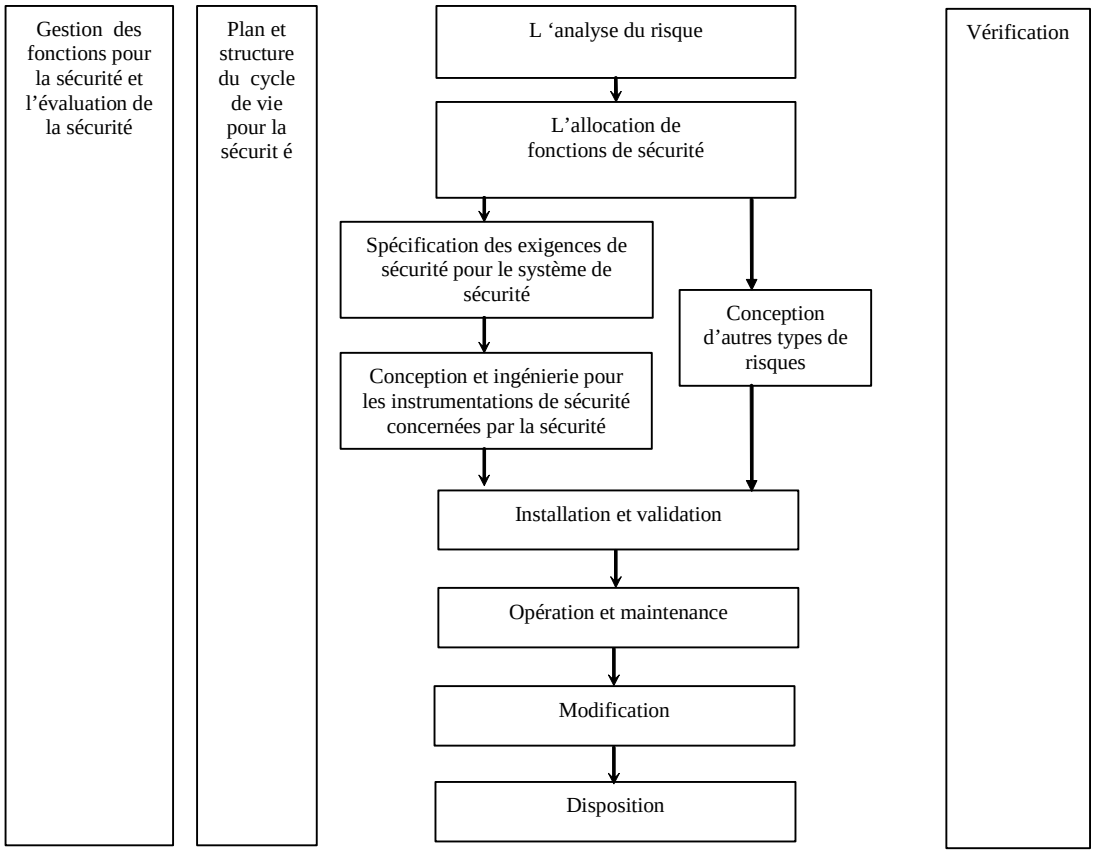

Figure II.8-Cycle de vie pour la sécurité présente dans le standard EIC61511. Ce cycle de vie est divisé en plusieurs phases.

Enfin, la norme doit être utilisée pour démontrer que le système opère en sécurité, que l'équipement est en sécurité et que le système est protégé contre les risques. En effet, la norme internationale IEC61508, vise à Améliorer la sécurité et la performance des systèmes complexes, permettre l'intégration de la sécurité durant le développement du système, et fournir une approche pour déterminer la performance de la sécurité des systèmes complexes (nucléaires, aérospatiales, ferroviaires, etc..) durant leur développement.

Dans la norme, on remarque que les échecs dangereux de la sécurité peuvent surgir :

- D'une spécification incorrecte des exigences,
- D'une omission des exigences de sécurité,
- D'un échec aléatoire des matériels,
- D'un échec systématique des matériels et des logiciels de commandes.
- Des erreurs humaines,

- D'une Influence de l'environnement (électromagnétique, température, etc.…)

7. LA TRACABILITE

La traçabilité est la possibilité de retrouver, pour un système donné, la trace de toutes les étapes de son cycle de vie, de sa fabrication à sa mise hors service, ainsi que la provenance de tous ses composants. En effet, la traçabilité d'un produit permet par exemple, de retrouver les fournisseurs des matières premières, les différents endroits où le produit a été entreposé, les manipulations et équipements utilisés dans sa fabrication. Dans notre étude, on se focalise à chercher la trace d'une exigence. Pour cela, on emploie la traçabilité des exigences qui se rapporte à la capacité de suivre le cycle de vie d'une exigence [SAH-05].

La traçabilité est un processus difficile à réaliser manuellement [STE-03]. Depuis que les systèmes sont devenus de plus en plus complexes, la traçabilité entre les différentes phases du développement du système a pris une importance primordiale. Beaucoup d'avantages existent pour avoir un modèle de traçabilité. De nombreuses techniques existent pour tracer les exigences, parmi ces techniques on cite les réseaux sémantiques, l'hypertexte, les dépendances entre les documents, les matrices, les bases de donnée relationnelles. Actuellement, plusieurs outils fournissent une traçabilité des exigences durant le développement, parmi lesquels DOORS et RTM. Par la suite, nous allons présenter les liens de la traçabilité, et la Pré/Post traçabilité.

7.1. Liens de traçabilité

Une mauvaise élicitation des exigences, est parmi les raisons qui expliquent la mauvaise intégration d'un modèle de traçabilité des exigences au cours du développement du système. Selon [AND-98], la traçabilité est une activité de l'ingénierie des exigences plus particulièrement en matière de gestion des exigences. En fait, la traçabilité des exigences consiste à établir des liens :

- De dépendances entre les exigences,
- De déclinaisons entre les exigences,
- D'historiques entre les exigences.

7.2. Pré/Post traçabilité

Le besoin et l'effet de traçabilité sont divisés par les auteurs. Dans [SPA-02], on conclut que la traçabilité doit être minimale, puisqu'elle emploie des ressources et ralentit les processus du développement du système. Ils pensent que les traces doivent être faites seulement si c'est vraiment nécessaire ; et qu'elles doivent être limitées aux secteurs où elles ont une valeur ajoutée. D'autres, tel que [RAM-95], croient que la traçabilité n'a pas besoin d'être vue comme une charge durant les processus du développement, plutôt comme perfectionnement durant le cycle de vie du système.

En fait, les traces peuvent documenter le cycle de vie d'une exigence [JON-03]. Au début, la traçabilité des exigences a été établie comme un mécanisme afin d'assurer que les objectifs ont été satisfaits après la réalisation du système. Aujourd'hui, les utilisations courantes de la traçabilité des exigences incluent la garantie de la qualité, la réutilisation des composants, l'essai, l'entretien, et l'analyse d'impact du changement.

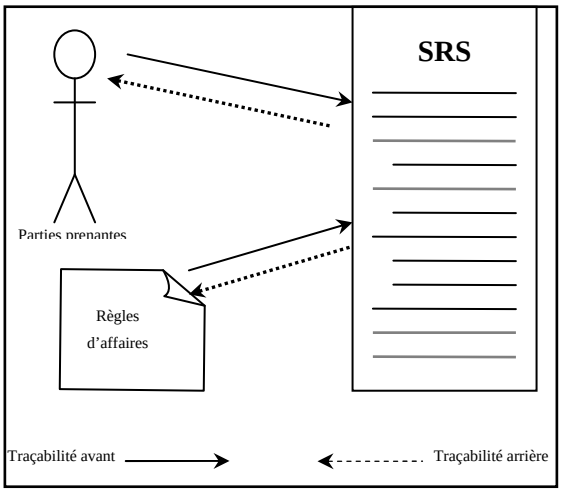

Figure II.9-Pré-traçabilité

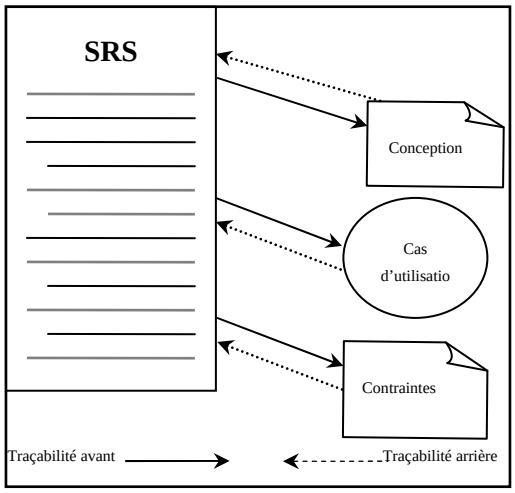

Figure II.10-Post-traçabilité

Dans [GOT-94], la traçabilité des exigences est divisée en : pré-traçabilité et post-traçabilité. La Pré-traçabilité (cf. Figure II.9) lie l'information concernant les sources des exigences et les règles d'affaire avec le SRS. Par contre, la post- traçabilité est l'ensemble des traces joignant la conception des exigences avec le SRS.

La pré-traçabilité (cf. Figure II.9) contient deux types de liens : traçabilité avant et traçabilité arrière. La traçabilité avant de la pré-traçabilité permet le passage des règles d'affaires vers le SRS. Par contre, La traçabilité arrière de la pré-traçabilité permet le passage de SRS vers les règles d'affaires [JAR-98].

La post- traçabilité (cf. Figure II.10) contient deux types de liens : traçabilité avant et traçabilité arrière. La traçabilité avant de la post-traçabilité permet la satisfaction des exigences existantes au niveau du SRS avec une conception. Par contre, La traçabilité arrière de la post-traçabilité est utilisée pour savoir pourquoi et pour quelle condition une conception existe.

Tous les liens fournis dans ce paragraphe, sont employés pour assurer le bon développement du système, et pour vérifier toutes les caractéristiques du système, afin d'accomplir la satisfaction des parties prenantes [JAR-98].

8. SPECIFICATION FORMELLE

Les méthodes formelles sont des techniques permettant de raisonner rigoureusement sur des programmes informatiques, ou aussi des matériels électroniques, afin de démontrer leurs conformités (correctness) en se basant sur des raisonnements de logique mathématique. En pratique, une méthode formelle est basée sur une notation formelle associée à un ensemble d'axiomes, permettant d'atteindre des exigences de qualité élevées du logiciel.

Elles sont généralement coûteuses en ressources (humaines et matérielles) ; vu que leur utilisation ralentit le développement des logiciels. Pour cela, les méthodes formelles sont actuellement réservées aux applications les plus critiques, afin d'appliquer un ensemble de preuves et de tests. En fait, selon [TRA-04] la spécification formelle est reconnue comme fondamentale, mais parfois difficile à utiliser durant le développement des logiciels. L'élaboration de nouvelles techniques pour les méthodes formelles et le développement des outils qui facilitent leurs applications, ont un impact considérable sur l'utilisation de ces méthodes.

Les méthodes formelles sont basées sur les sémantiques de programmes, c'est-à-dire sur des descriptions mathématiques formelles (contrairement à une spécification en langage naturel qui peut donner lieu à différentes interprétations). En fait, les méthodes formelles peuvent être vues comme les mathématiques appliquées pour le développement des systèmes informatiques. Car, elles sont exprimées à l'aide de langages dont la sémantique est définie mathématiquement, en s'appuyant sur des techniques de transformation et de vérification basées sur la preuve mathématique.

Enfin, l'application de méthodes formelles peut guider durant le processus de test. Pratiquement, les méthodes formelles sont appelées à jouer un rôle fondamental par

l'introduction d'une sémantique rigoureuse dans les méthodes à objets. Et, elles peuvent être utilisées pour donner une spécification du système qu'on souhaite développer, au niveau du détail désiré.

Alors, les descriptions formelles du système peuvent être utilisées comme des références pendant le développement. En plus, elles sont utilisées pour vérifier (formellement) que la réalisation finale du système (décrite dans un langage informatique dédié) respecte les attentes initiales (notamment en termes de fonctionnalité) des parties prenantes.

8.1. Les attributs d'une Spécification formelle

Parmi les qualités attendues d'une bonne spécification formelle, nous pouvons retenir ce qui suit: justesse, non- ambiguïté, vérifiabilité, consistance, convivialité, modificabilité.

- *Juste* : Une spécification formelle est juste si et seulement si elle satisfait totalement les exigences des parties prenantes, ce qui revient à s'assurer que chaque élément de la spécification correspond à une exigence bien exprimée dans le SRS,
- *Non- ambiguë* : Une spécification et non- ambiguë si et seulement si chacun de ses éléments a une interprétation unique.
- *Vérifiable* : Une spécification est vérifiable si et seulement s'il existe un ou plusieurs processus finis a l'aide desquels, une personne ou une machine peut s'assurer que le logiciel correspond bien a la spécification formelle. L'ambiguïté est parmi les facteurs qui jouent un rôle pour que la spécification ne soit pas vérifiable.
- *Consistante* : Une spécification est consistante si et seulement si aucun élément n'est en conflit ou en contradiction avec les autres éléments. Les inconsistances peuvent se manifester dans une spécification sous les formes suivantes: Comportement conflictuel, lorsque les deux parties de la spécification spécifient des stimuli différents et en conflits pour gérer une réponse particulière, ou spécifient des réponses différentes à des stimuli identiques.
- *Conviviale* : La convivialité s'adresse surtout aux clients et aux utilisateurs qui sont très souvent experts dans un domaine d'application donné, mais n'ont pas forcement la formation requise pour comprendre des spécifications complexes. Pour cela, un des moyens développés par les chercheurs pour rendre les

spécifications formelles plus accessibles aux utilisateurs, est d'utiliser les notations semi formelles [SAH-01].

- *Modifiable* : Une spécification est modifiable si sa structure et le style utilisé sont tels que tout changement nécessaire peut se dérouler facilement, de manière complète et consistante.

8.2. Méthodes de vérification

Un intérêt d'utiliser les méthodes formelles, est la capacité de spécifier les fonctionnalités du système avec une abstraction mathématique. Les méthodes formelles prennent tout leur intérêt lorsque les preuves elles-mêmes sont garanties correctes formellement. On peut distinguer deux grandes catégories d'outils permettant la preuve de propriétés sur des modèles formels : la preuve automatique et le model-checking:

8.2.1. Preuve automatique

La preuve automatique des théorèmes comme celle utilisée pour la vérification et la validation dans le travail fait dans [TRA-97] pour le passage du statecharts vers VDM et d'UML vers PVS[15] [TRA-04]. La preuve automatique consiste à laisser l'ordinateur prouver les propriétés automatiquement, étant donné une description du système, un ensemble d'axiomes et un ensemble de règles d'inférences. La preuve de théorème par rapport au model-checking, a l'avantage d'être indépendant de la taille de l'espace des états, et peut donc s'appliquer sur des modèles avec un très grand nombre d'états, ou même sur des modèles dont le nombre d'états n'est pas déterminé (modèles génériques).

8.2.2. Model-Checking

Le Model-Checking est une technique de vérification automatique de systèmes informatiques (logiciels, circuits logiques, protocoles de communication). Il s'agit de tester algorithmiquement si un modèle donné du système lui-même, ou une abstraction du système, en satisfait une spécification logique et une propriété, généralement formulée en termes de logique temporelle.

Le model-checking consiste à vérifier des propriétés par une énumération exhaustive et astucieuse (selon les algorithmes) des états accessibles. L'efficacité de cette technique dépend

[15] Proof Verification System

en général de la taille de l'espace des états accessibles et trouve donc ses limites dans les ressources de l'ordinateur pour manipuler l'ensemble d'états accessibles. Des techniques d'abstraction (éventuellement guidées par l'utilisateur) peuvent être utilisées pour améliorer l'efficacité des algorithmes.

8.3. Spécification Semi Formelle/Formelle

Actuellement, UML[16] est l'un des formalismes les plus utilisés pour la représentation des systèmes. La modélisation en UML émerge de plus en plus dans le cycle de développement pour tout type de système (électronique, informatique, ferroviaire, médicale, temps réelles, etc…). Cette représentation est née de la fusion de trois méthodes: OMT, Booch et OOSE. UML est le résultat d'un large consensus. De très nombreux industriels ont adopté UML pour le développement de leur système.

Dans UML, le modèle est une abstraction de la réalité. L'abstraction est un des piliers de l'approche objet. Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une entité en vue d'une utilisation précise.

Le caractère abstrait d'un modèle doit notamment faciliter la compréhension du système étudié. Il réduit la complexité du système étudié, permet de simuler le système, le représenter et le reproduire. La représentation actuelle de UML2.0 (après l'extension du standard UML1.0) est formée actuellement de 13 diagrammes (diagramme de cas d'utilisation, diagramme de classes, diagramme d'états transitions, etc.…) avant tout la modélisation avec UML est un support performant de communication, qui facilite la représentation et la compréhension de solutions objets :

- Sa notation graphique permet d'exprimer visuellement une solution objet, ce qui facilite la comparaison et l'évaluation de solutions.
- Son indépendance par rapport aux langages de programmation, aux domaines d'application et aux processus, en fait un langage universel.

[16] UML : Unified Modeling Language

8.3.1. UML/PVS

La spécification en utilisant les diagrammes UML n'est pas suffisante pour fournir une spécification complète pour un système, et pour la vérification de ses propriétés. C'est dans ce but qu'il faut trouver d'autres moyens pour préciser la sémantique du modèle.

De nos jours la plupart des industriels s'intéressent à développer leur système le plus vite possible dans le but d'augmenter leurs chiffres d'affaire. C'est pour cela, que la plupart des produits sont livrés sans avoir l'assurance de qualité et pour certains produits le développeur demande aux utilisateurs de trouver les erreurs ; mais cela est anormal dans le cas des domaines critiques (le transport, la médecine, les centrales nucléaires).

Bien que l'utilisation des méthodes formelles augmente la confiance durant le développement des systèmes, la majorité des industriels ne préfèrent pas l'utiliser vu la difficulté d'intégration, car elle ralentit le développement des logiciels (car ils préfèrent le passage directe d'UML vers un langage de programmation). Malheureusement, la plupart des programmes sont produits directement à partir des exigences sans aucune caractéristique efficace. La difficulté d'intégration des méthodes formelles durant le cycle de développement est un obstacle devant les projets industriels. Actuellement, la notation OCL[17] complète l'expressivité d'UML, mais sa contribution dans le contexte du raisonnement rigoureux est limitée par le manque de la sémantique formelle.

Dans ce but, beaucoup des chercheurs ont développé différentes approches pour passer d'une spécification semi–formelle (qui n'a pas une sémantique claire et précise, comme UML) vers une autre plus formelle. Parmi ces approches, nous pouvons citer les travaux permettant de passer de RSML vers le Model-checking NuSMv, et d'autres travaux pour passer d'UML vers PVS.

[17] Object Constraints Language

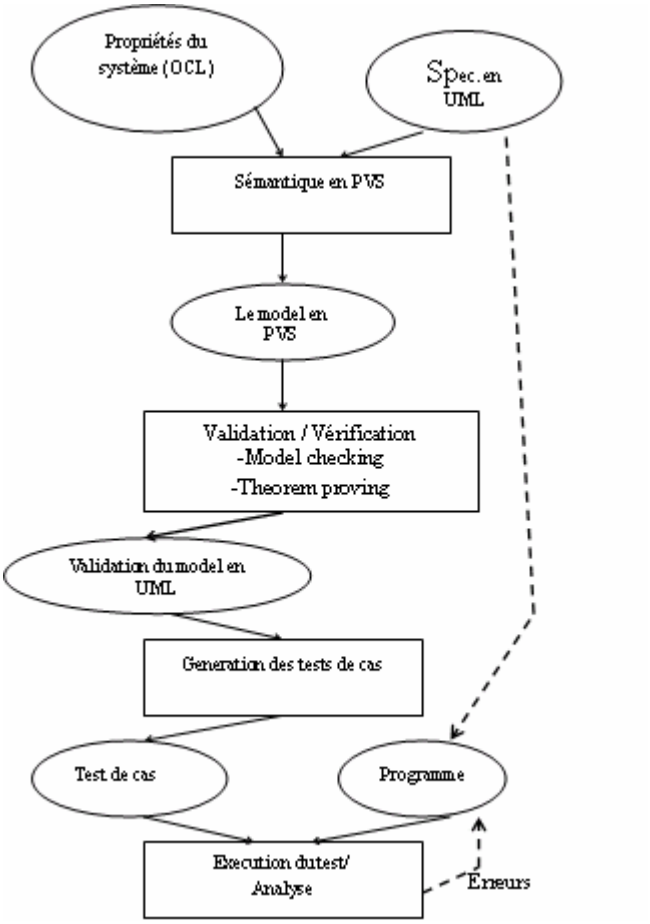

Figure II.11-L'approche PRUDE présente le passage de la représentation semi-formelle d'UML, vers la spécification formelle en PVS

8.4. Spécification formelle avec RAISE

Actuellement, dans le milieu académique existe plusieurs méthodes formelles pour la spécification abstraite des systèmes. Parmi ces méthodes formelles ont peut citer : Z, B, VDM, RAISE. La méthode formelle RAISE (Rigorous Approach to Industrial software Engineering) [ZIE-94], fournit des équipements pour l'usage industriel des méthodes formelles durant le développement des logiciels. Alors, le but de cette méthode est de permettre :

- Le développement de logiciels plus fiables,
- Le développement de logiciels avec peu d'erreurs,
- L'amélioration de la documentation de logiciels,
- La maintenabilité de logiciels.

Comme les autres notations formelles, RAISE est basé sur les notations mathématiques. Actuellement il existe des outils qui supportent ce langage. Le but de RAISE est d'améliorer le processus de développement de logiciel en soutenant une discipline qui peut être projetée et effectuée dans un contexte industriel. C'est pour cela qu'ils ont développé la méthode RAISE, et le langage de spécification RSL (RAISE Specification language)

8.4.1. La méthode RAISE

La méthode RAISE inclut des directives pour la plupart des activités appropriées de développement de logiciel et de systèmes, telles que l'élicitation des exigences et la gestion de projet. La méthode RAISE est basée sur un ensemble d'étapes. Selon ce paradigme, le logiciel est construit par une série d'étapes, où chaque étape représente un raffinement de la précédente. En fait, RAISE soutient un type de raffinement connu sous le nom de la théorie d'extension. Parmi les outils supportés pour RAISE, existe un outil pour l'aspect de vérification.

Evidement une méthode comme RAISE fournit une base solide pour obtenir des logiciels corrects et qui correspondent aux exigences des parties prenantes. La méthode RAISE n'est pas prescriptive, mais elle fournit un cadre et des directives durant le développement du logiciel. La méthode RAISE permet ainsi aux utilisateurs de choisir le niveau de formalisation approprié aux circonstances particulières, et aux normes du projet.

8.4.2. Langage de spécification RAISE : RSL

Le langage de spécification RSL, est développé pour pouvoir utiliser la méthode RAISE. L'avantage de RSL est que les utilisateurs doivent seulement connaître une notation en plus du langage de programmation utilisé pour l'exécution et l'implémentation du logiciel, vu que le développement est entièrement exécuté avec RSL. En fait, la structuration de RSL facilite la décomposition des différents modules des systèmes et facilite leur réutilisation. Bien sûr, RSL doit être considéré comme un modèle mathématique et comme une représentation mathématique des exigences essentielles à développer. La distinction entre les différentes parties de la spécification présentée à plusieurs niveaux, se différencie par leurs niveaux d'abstraction.

8.4.3. D'UML vers RSL

Nous avons précisé dans les paragraphes précédents, comment les chercheurs dans le domaine du génie logiciel s'intéressent au développement des traducteurs pour les

représentations graphiques (qui sont facile à manipuler) vers les représentations formelles pour effectuer les analyses nécessaires. Egalement à l'approche UML/PVS, un outil était développé sous Java (pour qu'il soit un outil portable) au sein de l'IIST-UNU[18] pour générer automatiquement des fichiers en RSL qui correspond à la représentation du diagramme de classes en UML. L'utilisation globale de l'outil UML2RSL peut être simplifiée avec :

- La spécification de la classe diagramme en UML,
- Exportation du model UML en XML,
- L'utilisation du traducteur pour générer la spécification en RSL.

Dans le chapitre IV nous allons détailler beaucoup plus l'utilisation de cet outil lorsqu'on va détailler les fonctionnalités de l'outil qui doit supporter la méthodologie de la recherche d'impact.

8.4.4. De RSL vers SAL

Parmi les outils supportés par RSL, existe le traducteur vers le model-checking SAL. En effet, SAL représente le laboratoire symbolique d'analyse. Il est utilisé pour l'analyse de programme, le calcul de propriétés pour les systèmes de transition. Enfin, on emploie le traducteur pour analyser l'impact du changement sur la sécurité.

9. CONCLUSION

Ce chapitre nous a permit de présenter les approches existantes pour la recherche d'impact lors d'une demande de changement, et il nous a permit d'introduire les notions de traçabilité et des besoins d'analyse formelle. La traçabilité nous aide à connaitre les liens entre les exigences contenues dans le SRS, la conception et la réalisation afin d'analyser l'impact d'une modification sur le coût le délai et la sécurité. Les spécifications formelles permettent d'analyser la cohérence entre les exigences du système. Le chapitre suivant sera divisé en quatre axes majeurs:

- La présentation des traces et liens rencontrés au cours du développement,
- La présentation du système de sécurité,
- La présentation de notre démarche,

18 International Institute of Software Technologies-United Nations University

- Et, la présentation du cycle de vie de la modification.

Enfin, la méthodologie présentée dans le chapitre suivant, nous aide pour appliquer des modifications sur le système en le gardant en sécurité.

CHAPITRE III

METHODOLOGIE

1. INTRODUCTION

La complexité rencontrée durant le développement des systèmes complexes, rend l'analyse d'impact du changement des exigences de plus en plus difficile. Alors, l'élaboration d'une étude qui se focalise sur la recherche d'impact du changement est un grand défi.

Dans les deux premiers chapitres nous avons présenté : la problématique de la volatilité des exigences, les processus d'ingénierie des exigences, les processus d'ingénierie systèmes, et les approches abordées pour analyser l'impact lors d'une demande de changement. Beaucoup d' études ont été élaborées pour analyser l'impact du changement des exigences sur plusieurs aspects [STE-03] [BOH-02]. Mais, il n'y a pas une méthodologie disponible pour analyser l'impact du changement des exigences sur la sécurité. Dans ce but, nous allons présenter notre méthodologie concernée par : l'explication des relations rencontrées durant le cycle de vie des systèmes, la décomposition du système de sécurité, l'explication de notre démarche, et la présentation du cycle de vie pour la demande de changement.

2. RELATIONS ET TRACES

La demande de changement cause des échecs et des perturbations, et dans certains cas une désorganisation durant le cycle de vie du développement de systèmes complexes. Cette désorganisation va apparaître lorsqu'il y aura une application d'une demande de changement dans une phase donnée sur une partie du système, sans connaitre ni les parties, ni les phases impactées par la demande. D'où l'intérêt d'extraire les traces et les relations qui relient les différentes parties du système. Cette extraction nous aide dans la définition de la phase et les parties du système impactées par une telle demande de changement.

Par la suite, nous allons présenter tous les types des relations rencontrées durant le cycle de développement. Puis nous allons détailler les types de relations existantes au cours du développement du système lors d'une décomposition du système selon l'EIA-632.

2.1. Relations durant le développement

Si on considère le cas réel du développement des systèmes complexes représenté par la figure ci-dessous (cf. figure III.1), on peut dire que les trois phases du développement sont trop interdépendantes. Alors, l'échec dans une phase donnée affecte directement les autres phases liées à elle. En fait, le développement des systèmes complexes change dans des circonstances diverses, au cours de leur cycle de vie et après leurs livraisons.

Pour cela, plusieurs outils de développement sont utilisés pour le développement et la conception virtuelle des systèmes. Ces outils supportent différents modèles et diagrammes pour la représentation du comportement du système à développer. Egalement, ils supportent un modèle de traçabilité. Le but d'intégrer le modèle de traçabilité est de chercher les traces de la demande de changement afin d'analyser l'impact.

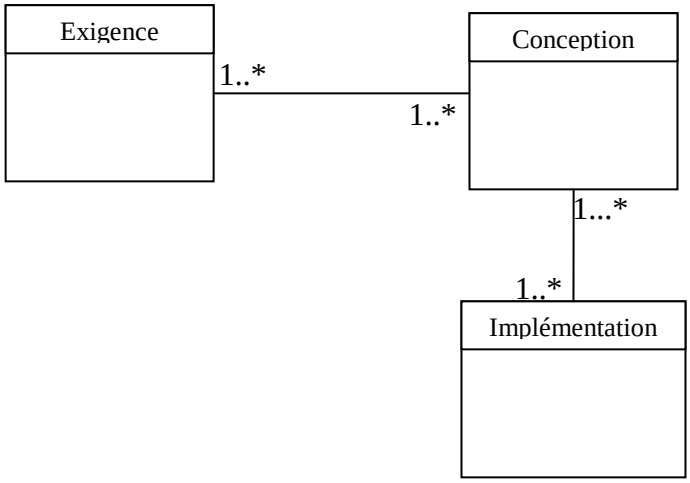

Figure III.1- La relation entre les différentes parties du développement des systèmes est: plusieurs à plusieurs.

Parmi la variété d'outils de développement existants on peut citer : Rational Rose, et Rhapsodie. Ces outils ne supportent pas une analyse automatique d'impact, lors de la mise en application d'une demande de changement donné. Par contre, ils supportent une recherche des traces. Cette recherche est basée sur la recherche de traces d'une telle demande de changement sur toutes les parties et les phases impactées par la demande.

Dans la figure ci-dessous (cf. figure III.2), on présente le cas de développement d'un exemple simple selon le diagramme des classes présenté dans la figure (cf. figure III.1). En effet, la flèche existante entre deux cercles représente une relation entre deux phases différentes (cf. figure III.2). Le cycle de vie de développement de cet exemple est décomposé en trois phases. La première phase est formée de cinq exigences, la deuxième phase est formée de quatre parties de conception et la troisième phase est formée de cinq parties d'implémentation. En fait, entre deux phases différentes existent des relations pour faciliter la recherche des traces d'une telle demande de changement.

Par exemple, la traçabilité arrière de l'implémentation I1 donne : C1 comme conception, et {E1, E2} comme exigence. Alors, dans le cas où on a une demande de changement sur l'implémentation {I1}, on peut facilement conclure que la conception {C1} et les exigences {E1, E2} seront impactées par la demande de changement. Autrement, la traçabilité en avance de E5 donne C4 comme conception et {I3, I4, I5} comme implémentation.

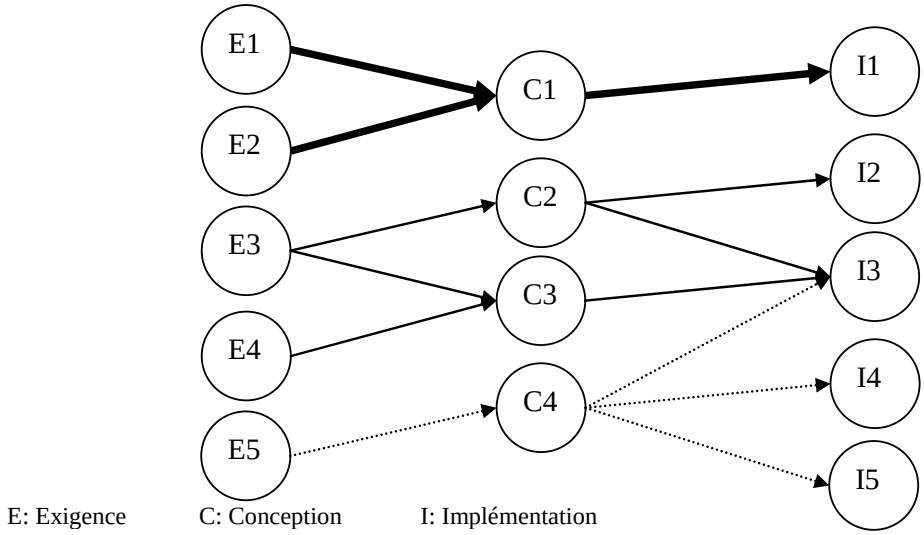

Figure III.2- Relations et traces au cours du cycle de vie du développement des systèmes

Effectivement, dans le même exemple (cf. figure III.2), on remarque qu'une exigence peut être conçue avec deux ou plusieurs parties de la conception différentes. Egalement, une seule partie de la conception répond parfois, à une ou plusieurs exigences existantes dans le SRS[19].

Dans le paragraphe précédent, on a précisé qu'entre deux phases successives existent des liens et des relations qui servent à chercher les traces d'une telle demande de changement. Par

[19] System Requirement Specification

contre, la recherche des traces n'est pas suffisante pour maîtriser l'analyse d'impact (soit sur la sécurité ou autres genres d'impact) lors de la demande de changement. D'où, l'intérêt d'extraire tout les types des relations rencontrées durant le cycle de vie du système.

Plusieurs types des relations sont rencontrés durant le cycle de vie des systèmes. Dans le diagramme présenté dans la figure ci-dessous (cf. figure III.3), on remarque deux types de relations principales: le premier type représente les relations existantes dans une seule entité; le deuxième type représente les relations entre deux entités différentes. Effectivement, le premier type se résume par les relations de dépendance, et de décomposition. Par contre, le deuxième type est divisé en: la traçabilité avant et la traçabilité arrière[20].

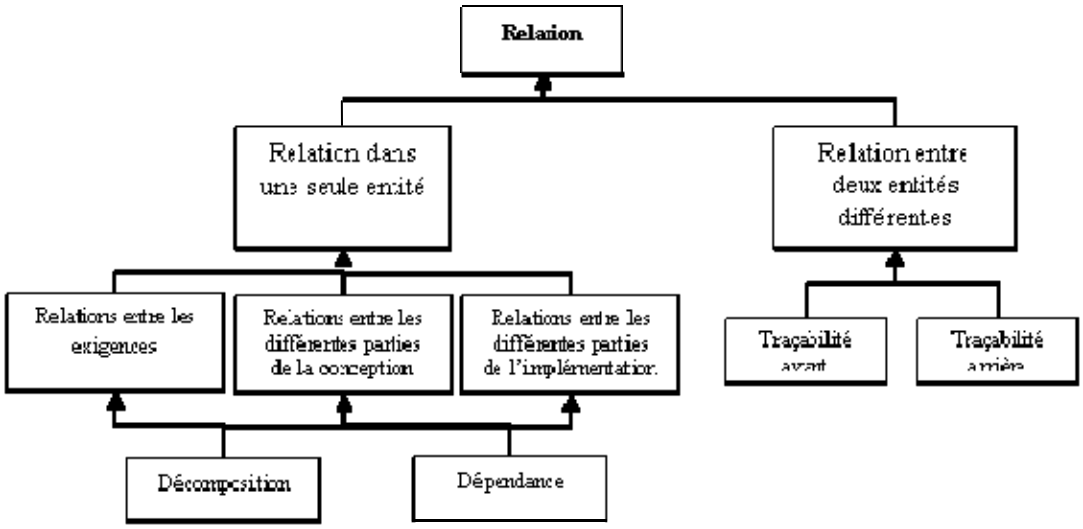

Figure III.3- les différents types des relations.

2.2. Les traces dans un module

Le développement du système selon l'EIA-632, exige une décomposition en plusieurs niveaux (voir chapitre 2). Chaque niveau donné contient un ou plusieurs modules. En effet, le développement d'un tel module, passe par sept étapes principales, et qui sont : des exigences techniques, une procédure de vérification, une solution logique, une solution physique, des exigences techniques dérivées, une conception, et finalement des exigences spécifiées pour le niveau suivant (qui existe dans le niveau plus bas).

Entre chacune des différentes étapes dans un même module, existe au moins une relation. Parmi les relations qui existent durant le développement d'un seul module, on cite les relations

[20] Ces deux types de relations de traçabilité, permettent le passage d'une phase vers une autre dans les deux sens.

: allouées, génère, est basé sur, source de, définie. Ces relations sont toutes des relations de traçabilité.

Lorsqu'il y aura une soumission d'une demande de changement sur un tel module, on garde les sept étapes déjà présentées (entourées par des ellipses). Mais, on aura de plus : des nouvelles étapes et de nouveaux types de relations. Les nouvelles étapes sont caractérisées par : une fiche pour la demande de modification (DDM), une justification, une nouvelle exigence émise par les parties prenantes, et une nouvelle contrainte. Les nouveaux types des relations sont : basée sur, amène à, crée, génère, contient, détermine.

Malgré qu'on a gardé les mêmes étapes précisées par l'EIA-632, l'application de la demande de changement pour un module donné implique : des nouvelles exigences techniques, une nouvelle procédure de vérification, une nouvelle solution logique, une nouvelle solution physique, des nouvelles exigences techniques dérivées, une nouvelle conception, des nouvelles exigences spécifiées.

La relation " génère " (cf. figure III.4), présente la relation de décomposition entre les exigences techniques. Par contre, les autres flèches représentent les relations de traçabilité arrière et de traçabilité avant. Par exemple la relation " source de " qui se trouve entre la nouvelle solution physique et la nouvelle solution de conception est une relation de traçabilité avant pour la nouvelle solution physique. Et, le passage dans le sens inverse sera basé sur la notion de traçabilité arrière.

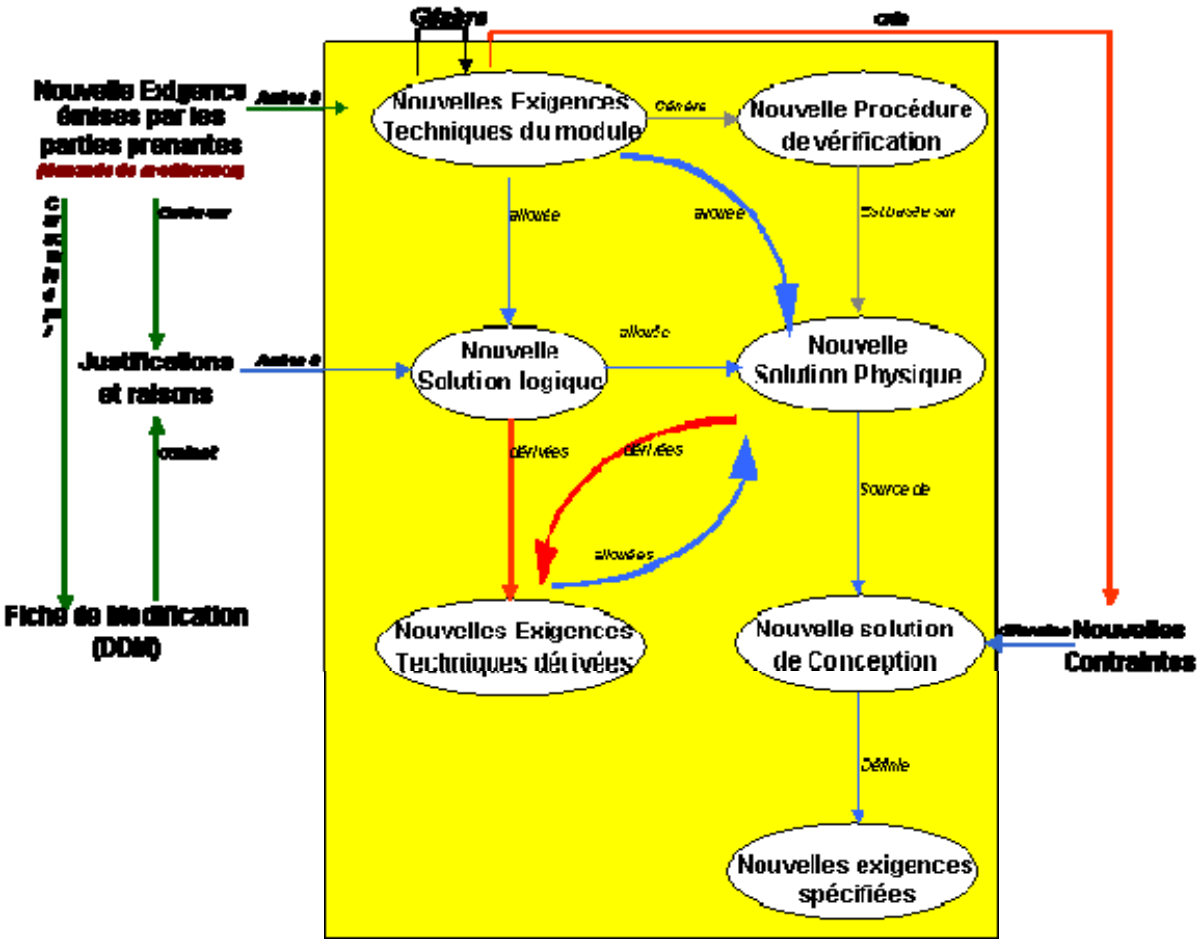

Figure III.4-les traces d'un module, afin de développer le produit final.

Enfin, l'application de la demande de changement ne s'arrête pas au niveau de la demande. Mais, il faut continuer avec la même démarche présentée dans la figure (cf. Figure III.4) pour changer les modules qui se trouvent dans le niveau plus bas du module impacté pas la demande de changement.

3. SYSTEME DE SECURITE

Un système sécurisé, est un système qui n'aura pas d'accident (un accident est un événement non désiré et non planifié) ni une perte inacceptable (une perte est une destruction des propriétés et des accidents fatals) ni une dégradation de la sécurité, au cours de son développement ou de son exploitation. En effet, la dégradation de la sécurité d'un tel système est liée parfois à l'évolution asynchrone. L'évolution asynchrone apparaît lors de l'application d'une demande de changement sur une partie du système sans connaitre les autres parties ou les sous-ensembles impactées par l'application de la demande [Site1].

Les problèmes de sécurité sont considérés comme résultat des échecs des sous-systèmes ou des composants [LEV-03b]. Ils proviennent d'un contrôle insatisfaisant pour le système de sécurité durant la phase d'analyse des exigences, la phase de conception, ou la phase d'implémentation. Si on considère le cas d'un mauvais contrôle durant la phase d'analyse des exigences, on remarque qu'à ce niveau on a une certaine ambiguïté entre la fiabilité et la sécurité du système. Surtout, lorsqu'on pense que le système est sécurisé s'il est fiable. Cependant, dans [LEV-04] [LEV-05] ils indiquent la possibilité d'avoir un système qui n'est pas fiable mais sécurisé, ou bien un système fiable mais n'est pas sécurisé.

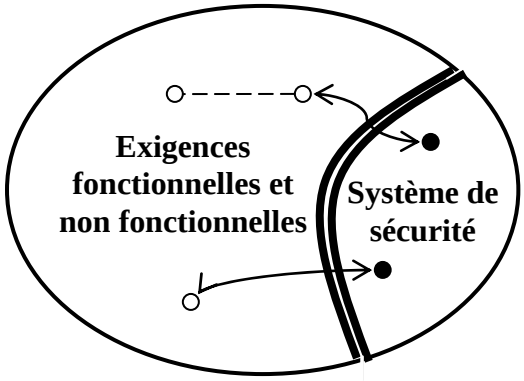

Figure III.5-Le lien avec le système de sécurité

Pour garantir la sécurité du système, on propose d'intégrer le système de sécurité dès la phase des exigences. Dans notre étude, on considère que le système de sécurité est formé par un ensemble des exigences sécuritaires (cf. figure III.6). Pratiquement, on représente le système de sécurité à part des exigences fonctionnelles et non fonctionnelles. Puis, on associe des liens entre les exigences (fonctionnelles et non fonctionnelles) avec le système de sécurité. En effet, les exigences qui sont en liaison avec le système de sécurité, on les appelle : exigences liées à la sécurité. Cette représentation nous aide à étudier si le système de sécurité est impacté par l'application d'une telle demande de changement.

Dans la figure ci-dessus, on remarque que le système de sécurité est présenté à part des exigences fonctionnelles (respectivement non fonctionnelles). Les deux flèches non détachées représentent les relations de dépendance entre les exigences sécuritaires avec les exigences fonctionnelles et non fonctionnelles du système. L'exigence En (respectivement Em) est une exigence liée à la sécurité.

Enfin, la sécurité ne se limite pas que sur les exigences sécuritaires, car elle est concernée encore par le plan de sécurité qui sert à assurer que le système (à développer) est sécurisé vis-

à-vis de son environnement. Par la suite, nous allons présenter les trois différents types des exigences sécuritaires.

3.1. Exigences Sécuritaires

Les exigences fonctionnelles (respectivement non fonctionnelles) spécifient ce que doit faire le système (respectivement les propriétés comportementales que les fonctionnalités doivent avoir). En effet, les exigences sécuritaires qui forment le système de sécurité au niveau des exigences, spécifient ce que le système ne doit pas faire [BER-98], et comment garder le système en sécurité au cours de son développement et lors son exploitation. Alors, une exigence sécuritaire est une exigence concernée par la sécurité.

L'ingénierie de la sécurité contient plusieurs types d'analyses qui servent à construire le système de sécurité. Dans notre méthodologie, nous proposons une décomposition pour le système de sécurité entre : la sécurité fonctionnelle, et la sécurité non fonctionnelle. Cette décomposition est recommandé car la plupart des personnes se focalisent sur la sécurité fonctionnelle (en analysant les types de problèmes si elle n'est pas mise correctement en application) et négligent la sécurité non fonctionnelle.

En effet, la sécurité fonctionnelle contient deux catégories principales des exigences sécuritaires : (1) les exigences pures de sécurité, (2) les exigences de sécurité significative. Par contre, la sécurité non fonctionnelle contient le troisième type des exigences sécuritaires: (3) les contraintes de sécurité. Les trois types des exigences sécuritaires (cf. Figure III.6) spécifient l'aspect de sécurité primaire. Par la suite, nous allons présenter en détail l'explication de chaque type des exigences sécuritaires.

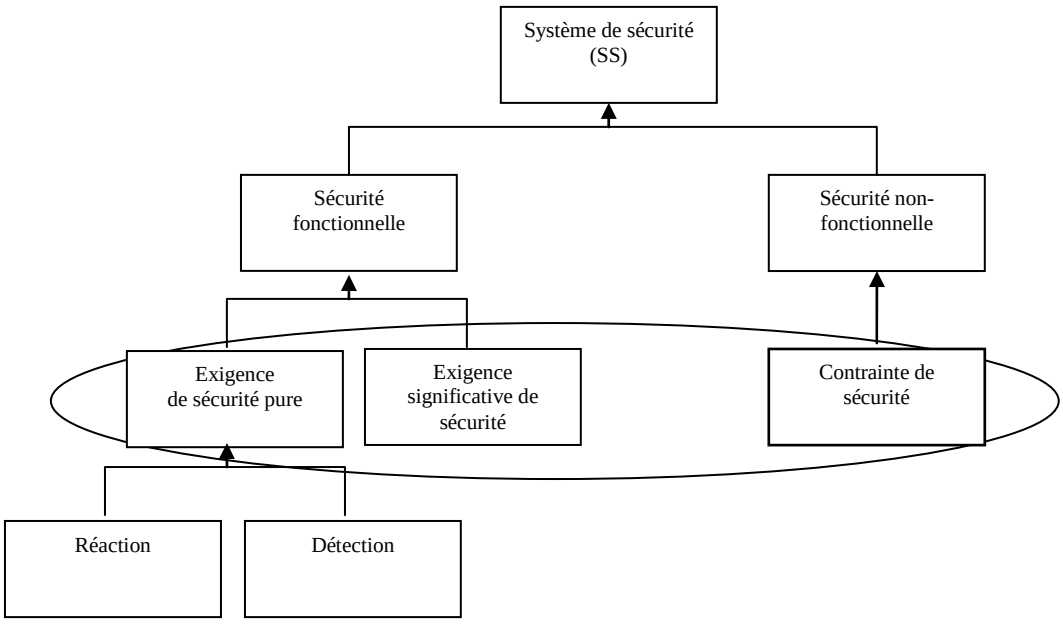

Figure III.6-le système de sécurité se décompose entre sécurité fonctionnelle et une autre non fonctionnelle

3.1.1. Sécurité fonctionnelle

A) Exigence de sécurité pure

Les exigences pures de sécurité décrivent les instrumentations destinées à la sécurité. Ces instrumentations seront utilisées pour surveiller la sécurité du système. En fait, ce type d'exigences contient la description des techniques de prévention pour l'assurance de la sécurité, des exigences de détection des accidents et d'autres pour la réaction lors de l'existence d'un accident. Ci-dessous, nous allons détailler les exigences de sécurité pour la détection et les exigences de sécurité pour la réaction.

Exigence de sécurité pour la détection: L'exigence de sécurité pour la détection est une exigence pure de sécurité (qui est une exigence sécuritaire). Elle décrit comment le système doit détecter l'occurrence d'un incident qui affecte la sécurité. Par exemple, l'exigence qui spécifie le capteur de détection (d'un cas critique dans un système donné) doit être parmi les exigences de sécurité pour la détection.

Exigence de sécurité pour la réaction: L'exigence de sécurité pour la réaction est une exigence pure de sécurité (qui est une exigence sécuritaire). Elle définit comment le système doit réagir lors de l'existence d'un tel accident, dans le but de garder le système en sécurité.

B) Exigence significative de sécurité

Une exigence significative de sécurité est une exigence qui décrit une partie physique du système, tel que la défaillance de cette partie physique entraine un problème de sécurité. Malheureusement, une exigence significative de sécurité n'est pas toujours prise en compte au niveau du système de sécurité.

Une exigence significative de sécurité n'est pas une exigence de sécurité au premier degré. Mais, elle doit être intégrée au sein du système de sécurité vu qu'elle peut affecter un problème de sécurité dans certains cas. Alors, toutes les exigences de ce type sont associées à des risques de sécurité. Parmi les attributs de ce type d'exigences on cite : la gravité de l'accident qui peut varier entre plusieurs niveaux lors d'une telle défaillance ; et la probabilité d'occurrence d'un tel accident.

Gravité	Definition de chaque niveau
Catastrophique	•1 Perte de la vie humaine (membres public, utilisateurs, opérateurs) •2 Perte du système •3 Blessure permanente •4 Dommages environnementaux graves irréversibles qui violent une loi ou des règlements.
Sévère	•1 Rend des personnes handicapées •2 Un dommage qui exige au minimum l'hospitalisation de 3 personnes •3 Perte d'une partie du système ou d'un sous système •4 Impact sur l'environnement qui viole une loi ou un règlement
majeur	•1 Un dommage qui amène à une perte d'une journée de travail •2 Perte d'un composant du système •3 Dommage considérable sur l'environnement d'où le rétablissement peut être accompli sans violer une loi ou un règlement.
mineur	•1 Une blessure qui n'entraîne pas un arrêt du travail d'au moins une journée. •2 Dommage minimal sur l'environnement dont le rétablissement peut être accompli sans violer une loi ou un règlement.
nulle	•1 Pas de perte

Tableau III.1 - Un exemple de la catégorisation de la gravité des accidents

En effet, la gravité de l'accident dépend en premier lieu du niveau d'impact de l'accident sur : l'environnement, les matérielles, et les propriétés. Dans le tableau ci-dessus, on présente cinq niveaux de gravité lors de l'occurrence d'un tel accident : Catastrophique, sévère, majeur, mineur, nulle. Dans notre étude, on ne présente pas la différence entre les différentes approches pour décomposer la gravité des accidents entre plusieurs niveaux [DUL-04]. Mais on indique qu'il faut intégrer les exigences significatives de sécurité au sein du système de

sécurité ; Car une petite lacune d'une simple partie physique du système peut avoir un impact négatif sur la sécurité.

Parallèlement à la décomposition des accidents entre cinq niveaux, il faut spécifier la probabilité d'occurrence de chaque accident. Dans le tableau 2, on présente les cinq niveaux pour la probabilité de l'occurrence des accidents, et qui sont : élevé, modéré, raisonnable, bas, négligeable. Dans le cas où il y aura une demande de changement qui affecte une exigence significative de sécurité, il faut effectuer des analyses spécifiques. Plus le niveau de la gravité sera élevé, plus la nécessité d'utiliser des méthodes de vérification de plus en plus rigoureuses sera obligatoires (dans notre étude on ne présente pas ces méthodes de vérification).

Probabilité	**Définition**
Élevé	L'accident aura lieu fréquemment durant l'opération du système la probabilité d'apparence d'accident durant un cycle d'opération : 10^{-1}<probabilité<=1
Modéré	L'accident aura lieu plusieurs fois durant le cycle d'opération du système la probabilité d'apparence d'accident durant un cycle d'opération : 10^{-2}<probabilité<=10^{-1}
Bas	La possibilité d'obtenir un accident durant le cycle d'opération du système la probabilité d'apparence d'accident durant un cycle d'opération : 10^{-3}<probabilité<=10^{-2}
Réduit	C'est quasiment improbable bien que c'est possible d'avoir un accident durant le cycle d'opération du système la probabilité d'apparence d'accident durant un cycle d'opération : 10^{-6}<probabilité<=10^{-3}
Négligeable	la probabilité d'apparence d'accident durant un cycle d'opération : probabilité<10^{-6}

Tableau III.2- Probabilité pour l'occurrence des accidents

Après la spécification de la sévérité et de la probabilité de l'occurrence d'un accident/risque donné, on peut construire la matrice de sécurité contenant les indices des risques pour la sécurité. Le but de cette matrice est de faciliter la gestion de la sécurité. Dans certaine norme les indices des risques sont appelés les niveaux intégrés de sécurité (Safety Integrity Levels) [STA-IEC]. Pratiquement, les niveaux intégrés de sécurité changent d'un projet à un autre, et d'un domaine d'application à un autre.

		Probabilité				
		Élevé	Modéré	Bas	Réduit	Négligeable
Gravité	catastrophique	Intolerable	Intolerable	Intolerable	Critique	Majeur
	Sévère	Intolerable	Critique	Critique	Majeur	Modéré
	Majeur	Critique	Critique	Majeur	Modéré	Modéré
	Mineur	Critique	Majeur	Modéré	Mineur	Mineur
	Nul	Nul	Nul	Nul	Nul	Nul

Tableau III.3- matrice de sécurité

Enfin, Si on considère comme cas d'étude le système automatique du métro qui relie les deux extrémités de l'aéroport, on remarque que lors de la description des exigences de ce système on aura des : exigences fonctionnelles, des exigences non fonctionnelles, des exigences sécuritaires forment le système de sécurité. Parmi les exigences sécuritaires, on trouve les exigences significatives de sécurité qui décrivent les parties physiques pour : l'ouverture, la fermeture et l'arrêt du métro. La présence de ces exigences au niveau du système de sécurité est due au risque de défaillance avec ces parties physiques qui causent des accidents graves lors de l'opération du métro.

3.1.2. Sécurité non fonctionnelle

A) Contrainte de sécurité

Le dernier type des exigences sécuritaires est la contrainte de sécurité. Ce type d'exigences sécuritaire est utilisé pour exprimer une contrainte sur le comportement du système, pour qu'il ne cause pas un problème de sécurité. On rencontre souvent les contraintes de sécurité pour ne pas avoir un blocage : du matériel, sur un système en mouvement, sur la manipulation des procédures pour les matériels toxiques.

Les contraintes de sécurité sont exigées par une politique de sécurité. Car, par la volatilité d'une contrainte de sécurité simple, la sécurité sera affectée. Par exemple, si on considère le même exemple du métro déjà présenté, parmi les contraintes de sécurité on cite les suivantes : il ne faut pas que les portes s'ouvrent jusqu'à ce (1) qu'il n'y aura aucun mouvement détecté, (2) que le train s'arrête sur le point prévu, (3) que le train ne reviendra jamais vers l'état de mouvement.

4. DEMARCHE

Le développement des systèmes complexes est traité en tant qu'un développement dynamique. Puisqu'il est composé d'un ensemble des processus, qui prennent en compte les demandes des changements proposées par les parties prenantes. En effet, les demandes de changement (ou les feedback) apparaissent tout au long du cycle de vie. Malheureusement, un problème de sécurité résulte : soit des processus défectueux exécutés durant le cycle de vie, soit d'une omission de quelques aspects de vérification lors de la soumission d'une demande de changement.

Par la suite, nous allons aborder successivement notre démarche proposée pour l'analyse d'impact lors d'une demande de changement. Cette démarche contient une présentation: des catégories de la demande de changement, des règles à respecter lors de l'application de la demande de changement, de l'approche générale[21], et du cycle de vie de la demande de changement.

4.1. Les catégories de la demande de changement

Dans notre étude on précise qu'il ne faut pas traiter toutes les demandes de changement de la même façon. Pour cela on décompose les demandes de changement selon deux catégories : la première catégorie concerne les demandes qu'il faut appliquer sans exécution des procédures de vérification, car ils n'ont aucun lien avec le système de sécurité; la deuxième catégorie concerne les demandes qui peuvent être appliquées après une analyse d'impact, car elles affectent le système de sécurité.

4.2. Règles à suivre

Dans ce paragraphe, nous proposons les règles qui nous paraissent importantes à suivre lors de l'application d'une demande de changement. En fait, toute demande de changement doit être appliquée en respectant l'ensemble des règles présentées ci-dessous :

- Aucune demande de changement ne doit être prise sans l'accord des parties prenantes intéressées,
- Les parties prenantes doivent prendre en considération que chaque demande de changement doit être suivi par cinq rôles différents: le rôle du demandeur, le rôle

21 Dans le chapitre suivant nous allons détailler le modèle de traçabilité.

de l'évaluateur, le rôle du responsable, le rôle du modificateur, et le rôle du validateur[22],

- Toute demande de changement doit tenir compte d'une étude d'impact (coût, sécurité, délais, techniques),
- Toute demande de changement doit obligatoirement avoir un cycle de vie,
- Il ne faut pas appliquer une modification sans connaitre ses traces.
- La demande de changement d'une exigence/conception/implémentation ne peut se faire qu'à partir de la dernière version du système (on propose que la dernière version du système soit une version sécurisée).

Enfin, les règles présentées ci-dessus doivent être respectées par notre outil, qu'on le présente au niveau du quatrième chapitre.

4.3. L'Approche Générale

La difficulté d'une demande de changement n'est pas limitée juste à son application, mais également à l'analyse de son impact sur la sécurité. Car, elle n'est pas restreinte à un modèle donné, et non plus à une phase donnée au cours du développement du système désiré.

La navigation entre les différentes phases durant le cycle de vie d'un système exige l'intégration d'un modèle de traçabilité. Plusieurs outils utilisés pour le développement des systèmes, intègrent un modèle de traçabilité. Parmi ces outils on rencontre DOORS. Par exemple, cet outil facilite la recherche de la trace d'une demande de changement tout en indiquant : les exigences impactées avec la demande, les diagrammes UML impactés avec la demande. Malheureusement, cet outil ne supporte pas l'aspect pour analyser l'impact sur la sécurité.

22 Une partie prenante peut jouer plusieurs rôles.

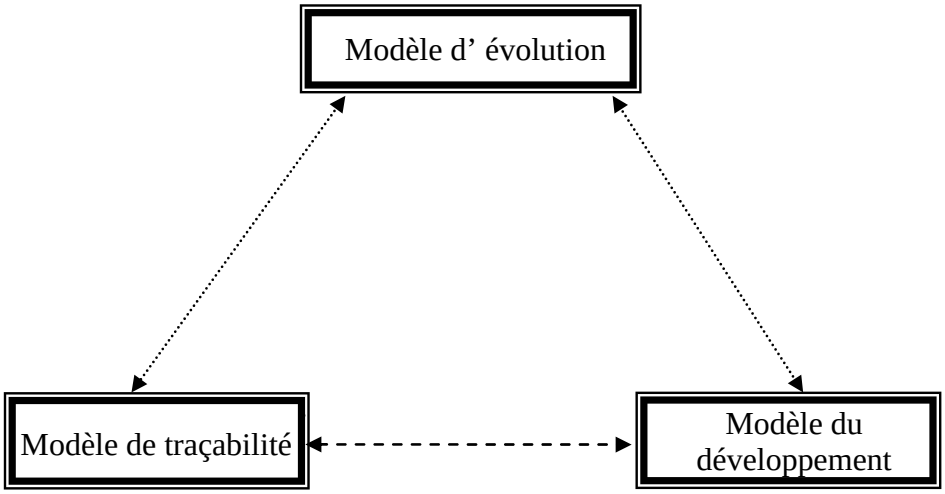

Figure III.7-L'approche générale

L'approche présentée ci-dessus, est notre infrastructure pour l'analyse d'impact lors d'une demande de changement. Dans cette approche on indique que l'analyse d'impact est basée sur l'interaction entre 3 modèles: le modèle de l'évolution ou de la demande de changement, un modèle de traçabilité, et le modèle du développement. Les flèches présentées dans la figure ci-dessus (cf. Figure III.7) représentent les liens d'interaction entre les différents modèles.

En effet, le modèle de traçabilité [ELJ-05a] est appliqué pour fournir les relations existantes au cours du développement du système complexe. Les relations fournies par le modèle de traçabilité permettent aux parties prenantes : de vérifier la satisfaction des exigences après la réalisation du système désiré et de chercher la trace de la demande de changement au niveau du système de sécurité. Et d'autre part, si le système de sécurité est impacté par la demande de changement. Effectivement, l'analyse d'impact de la demande sur la sécurité, doit être assurée au niveau du modèle du développement présenté dans la figure ci-dessus (voir § 4.3.2).

Comme notre approche est basée sur le contexte de l'ingénierie systèmes, on peut dire que : le modèle du développement du système, sera une décomposition selon plusieurs niveaux et plusieurs modules. Enfin, lors de la demande de changement l'interaction entre les trois modèles doit répondre à plusieurs questions comme :

- Quels sont les modules qui sont affectés par la demande de changement ?
- Combien des modules sont affectés par une demande de changement ?
- Quelle est la relation entre la demande de changement et le système de sécurité ?

4.3.1. Gestion de la modification

La demande de changement au cours du développement d'un tel système implique : une nouvelle configuration du système, un nouvel ensemble des processus, et une difficulté pour la gestion de la demande[23]. En effet, une mauvaise gestion de la demande amène à des mauvais résultats et joue un facteur énorme pour l'échec du projet. Car, l'application des demandes de changement désordonne les processus du développement du point de vue de la variation de la demande de changement n'est pas restrictive juste dans la phase où on applique la demande, mais elle passe de la phase des exigences à celle de l'implémentation. D'où la nécessité de bien gérer les demandes de changement. Une bonne gestion de la demande concerne une : recherche pour les parties impactées par la demande, et une analyse d'impact.

Dans la figure présentée ci-dessous, on remarque que la gestion de la demande de changement au début du cycle de vie est moins difficile que, la gestion de la demande durant les phases avancées du développement. En fait, pour limiter la complexité de la gestion on exige que chaque demande de changement soit suivie selon un cycle de vie, comme nous allons le détailler vers la fin de ce chapitre

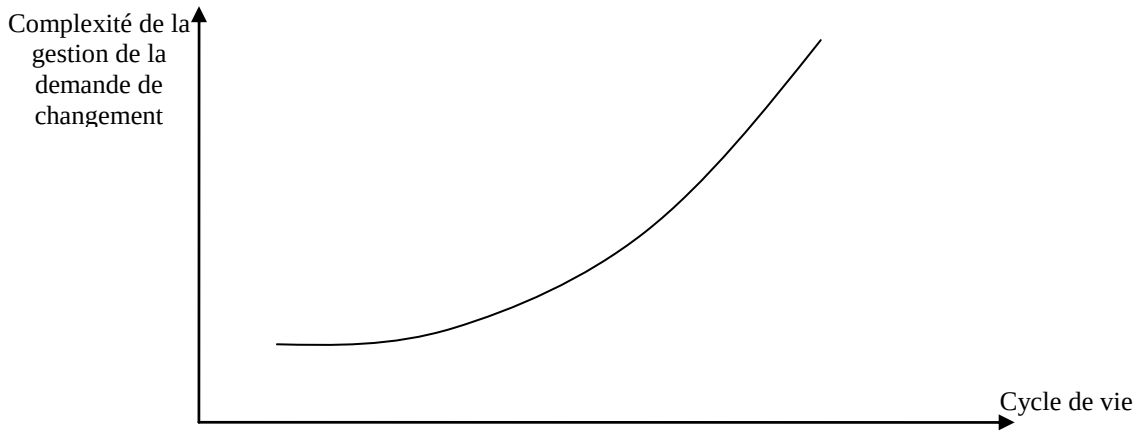

Figure III.8 -Plus le système est avancé durant son cycle de vie, plus la gestion de la modification est difficile.

[23] La complexité de la gestion des modifications durant le cycle de vie de développement du système, oblige parfois les industriels à décider d'intégrer les demandes de changements optionnels au sein de la nouvelle version du système.

4.3.2. Analyse d'impact sur la sécurité

Dans notre approche, nous avons montré que l'analyse d'impact lors d'une demande de changement, doit être basée sur l'interaction des trois modèles. Dans ce paragraphe, nous allons montrer comment on doit analyser l'impact du changement sur la sécurité lors d'une demande de changement au niveau du système de développement (modèle de développement). Le modèle du développement est représenté par : la phase des exigences du système, la phase de la conception du système et la phase de l'implémentation du système. Le modèle de traçabilité est représenté par les types des relations employées entre deux phases différentes (traçabilité en avance et traçabilité en arrière) et par les relations présentes à l'intérieur d'une phase donnée. Et, le modèle de d'évolution, est représenté par le type de la demande de changement, L'application du modèle de traçabilité, doit nous permettre de connaitre les liens entre :

- Exigence(s) <->Conception(s)
- Conception (s) <->Implémentation(s)
- Exigence (s) <->Implémentation(s)

	Changement d'une	**Phase du changement**	**Type du changement**	**Analyse à effectuer**
DCE	Exigence	Exigences	DCEE	Analyse préliminaire
	Exigence	Conception	DCEC	Analyse préliminaire+ Analyse d'impact formelle
	Exigence	Implémentation	DCEI	Analyse préliminaire+ Analyse d'impact formelle
DCC	Conception	Conception	DCCC	Analyse préliminaire+ Analyse d'impact formelle
	Conception	Implémentation	DCCI	Analyse préliminaire+ Analyse d'impact formelle
DCI	Implémentation	Implémentation	DCII	Analyse préliminaire+ Analyse d'impact formelle

Tableau4 - Les différents types de la demande de changement.

Au niveau de la phase des exigences on rencontre : les exigences fonctionnelles, les exigences non fonctionnelles et le système de sécurité. Les relations présentées au niveau des exigences sont les relations de dépendances et de décomposition. En fait, aucune demande de changement ne doit être appliquée sans avoir les données nécessaires. C'est dans ce but que notre analyse d'impact est basée sur l'utilisation d'un modèle de traçabilité. Alors, la traçabilité est utilisée :

- Premièrement, pour trouver la trace d'une demande de changement : au niveau des exigences du système (particulièrement au niveau du système de sécurité) au niveau de la conception, et au niveau de l'implémentation.
- Deuxièmement, pour garder la consistance du système durant son développement, c.à.d. il faut que les exigences correspondent à la conception du système, et que la conception corresponde à l'implémentation du système, et que l'implémentation valide les exigences du système.

La majorité des demandes de changement apparaissent durant la phase des exigences. Dans certains cas, les parties prenantes exigent l'application d'une évolution du système durant la phase de conception/implémentation tout en ajoutant une nouvelle fonctionnalité pour le système. Pour cela, on divise les demandes de changement selon trois types différents : une demande de changement pour une exigence (DCE), une demande de changement pour une conception (DCC) et finalement une demande de changement pour une implémentation (DCI). Les trois types des demandes de changement (DCE, DCC, DCI)[24] ont un seul format fixe que nous allons le présenter dans le quatrième chapitre. Ces trois types de demande de changement, représentent le modèle de l'évolution.

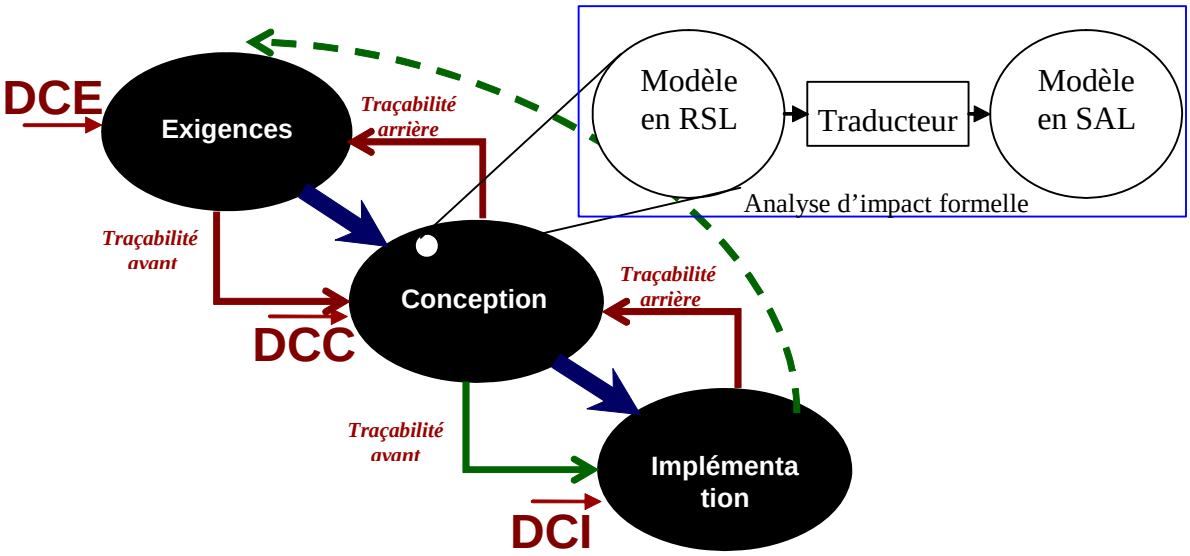

Figure III.9 –Analyse d'impact sur la sécurité.

24 Parfois les demandes de changement apparaissent en employant de nouvelles technologies

Si on considère que le développement du système a lieu durant la phase des exigences, alors l'analyse de la demande de changement d'une exigence (DCEE) est limitée à une recherche de la trace de la demande au niveau du système de sécurité (en utilisant le modèle de traçabilité au niveau des exigences). Par contre, l'analyse d'impact sur la sécurité pour tous les autres types de demande de changement, se divise en :

- *Analyse préliminaire* : Cette analyse sert à chercher la trace de la demande de changement au niveau du système de sécurité (comme dans le cas de DCEE). Cette analyse est établie juste avec un retour de la phase conception /implémentation vers la phase des exigences en utilisant les relations de traçabilité en arrière.
- *Analyse d'impact formelle:* Cette analyse doit être exécutée dans le cas où la demande de changement affecte le système de sécurité et précisément une contrainte de sécurité, car pour les deux autres types d'exigences sécuritaires il faut appliquer d'autres techniques d'analyses. En fait, cette analyse est basée sur des techniques bien précises. Pour cette analyse, on considère que les exigences fonctionnelles du système critique seront spécifiées formellement au niveau de la conception (modèle en RSL). Alors, avant l'application de la demande on précise l'obligation d'avoir une version provisoire pour le modèle spécifié formellement. Afin d'effectuer l'analyse nécessaire pour prouver que les contraintes de sécurité ne sera jamais affecté dans le cas où la demande sera intégré dans le système.

Enfin, si on projette le modèle de développement présenté au niveau de ce paragraphe sur le cas du développement d'un module dans le cadre de l'EIA-632(voir §2.2), on dit que :

- La phase des exigences correspond:
  - Aux exigences techniques du module,
  - A la procédure de vérification.
- La phase de la conception correspond:
  - A la solution logique<=> modèle formel en RSL
  - A la solution physique
  - A la conception du module

A) Analyse d'impact formelle

Une spécification formelle est une représentation abstraite pour le comportement du système. La représentation abstraite est préférée car la preuve des propriétés à un niveau abstrait est beaucoup plus facile, que la preuve de propriétés directement sur la description concrète du système. Certaines méthodes formelles comme la Méthode RAISE (également la méthode B) s'appuient sur une spécification selon plusieurs niveaux d'abstractions, en partant du plus abstrait vers le plus concret (ce processus est appelé processus de raffinement).

En effet, la partie Analyse d'impact formelle présentée dans la figure ci-dessus est composée de deux modèles et un traducteur. Le premier modèle représente la spécification formelle des exigences fonctionnelles du système critique avec RSL en intégrant les contraintes de sécurité, le deuxième modèle représente la transformation de la spécification formelle vers le model-checking SAL (Symbolic Analysis Laboratory) qui supporte l'aspect de la validité des contraintes de sécurité, et finalement un traducteur qui nous permet d'avoir le modèle en SAL après avoir spécifié le système formellement avec RSL.

A.1) Modèle formel

Pratiquement, avec RSL on ne peut pas spécifier que les exigences fonctionnelles et un seul type des exigences sécuritaires, qui sont les contraintes de sécurité. En fait, les contraintes de sécurité présentées au sein de système de sécurité seront vues comme des propriétés au niveau de la spécification formelle. Parmi les types des exigences qu'on ne peut pas spécifier avec RSL on peut citer par exemple les exigences d'espace [GEO-98] [HAX-00].

```
Object exemple:
    Class
        Type
            /*Définition des types explicitement*/
        Value
            /*Spécification explicite des fonctions du système*/
        Transition-System
            /*le système de transition spécifiant les différentes états du système*/
        Ltl-assertion
            /*Spécification des propriétés avec la logique temporelle pour vérifier leur validité*/
End
```

Figure III.10-Spécification générale en RSL

La spécification des contraintes de sécurité au sein de la spécification formelle du système critique nous permet de vérifier que les contraintes de sécurité sont toujours validées avant et après l'application de la demande de changement. En fait, lorsque la demande de changement affecte une contrainte de sécurité. Il faut avoir un nouveau cycle de vie pour la demande de changement. Le cycle de vie de la demande de changement implique la nécessité d'avoir un nouveau modèle formel du système en spécifiant les nouvelles fonctionnalités et en intégrant les contraintes de sécurité. Cette nouvelle spécification sera une spécification provisoire jusqu'à ce qu'on puisse prouver la validité des contraintes de sécurité. Tout cela, pour vérifier que la sécurité ne sera jamais impactée après l'application de la demande de changement.

A.2) Vérification avec SAL

Le passage du modèle formel en RSL vers le modèle représenté avec SAL en utilisant le traducteur RSL-SAL (développé au sein de l'IIST-UNU) exige une spécification applicative concrète pour les exigences fonctionnelles du système. La spécification applicative concrète signifie une spécification explicite pour toutes les fonctions du système, en répondant aux exigences fonctionnelles existantes dans le SRS. Le passage de la spécification abstraite des fonctionnalités du système ne permet pas une utilisation pour le model-checking SAL.

Après la spécification de toutes les fonctionnalités de système (cf. figure III.10), on ajoute le système de transition en précisant qu'elles sont les entrées et les sorties du système. Puis, on spécifie les contraintes de sécurité qui doivent être validées durant le fonctionnement du système.

Enfin, le model-checking SAL nous aide à valider les contraintes de sécurité existantes parmi les exigences sécuritaires. Mais, l'explosion des états est parmi les problèmes rencontrés lors de l'utilisation du model-checking SAL.

5. CYCLE DE VIE D'UNE MODIFICATION

Une demande de changement implique : des nouvelles démonstrations, des nouvelles preuves, des nouvelles procédures de vérification, et une nouvelle configuration du système. Quel que soit le type de la demande de changement, il faut qu'on exécute un ensemble des phases pour : initialiser, évaluer, appliquer et valider la demande.

Les différentes phases liées à la demande de changement sont simplifiées par : le cycle de vie de la demande de changement. Chaque cycle de vie pour la demande sera instruit à l'intérieur du cycle de vie du développement du système désiré. Pour cela, la demande de

changement exige des techniques spécifiques liées à chaque phase. Parmi les différentes techniques utilisées on peut citer par exemple: l'utilisation des scénarios, l'utilisation d'un système d'information, les méthodes de vérification (test des cas, Model-checking), etc….

Le cycle de vie pour la demande de changement doit être divisé en quatre phases : l'initialisation de la demande de changement, l'évaluation de la demande de changement, l'application de la demande de changement et finalement la validation de la demande. Dans le diagramme ci-dessous, on présente le détail de ces 4 phases, qui peuvent être appliquées dans le contexte de l'EIA-632.

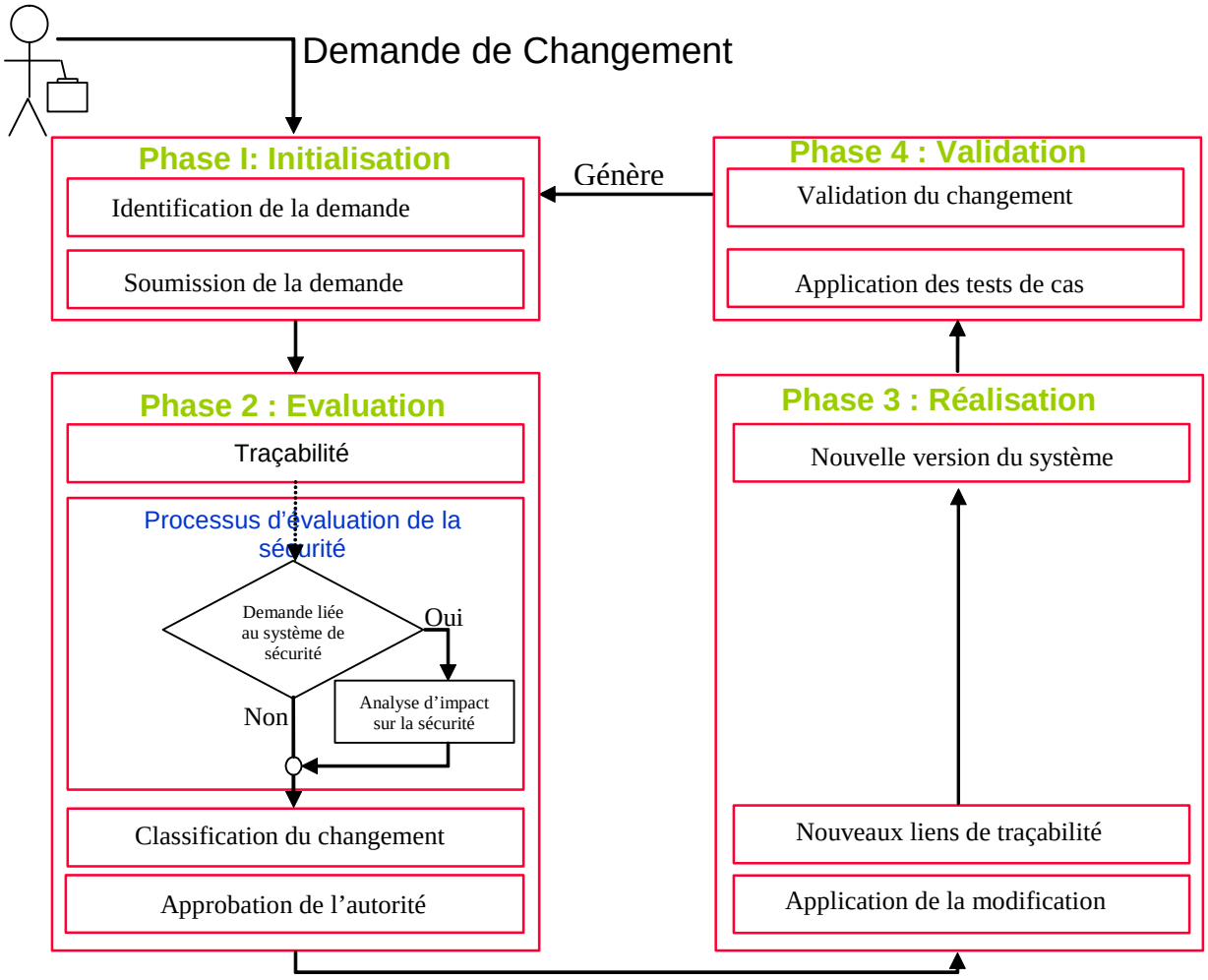

Figure III.11 –Cycle de vie pour une modification.

En effet, chaque phase contient plusieurs étapes qui peuvent être vues comme des processus internes. Par exemple, la phase d'initialisation contient deux processus qui sont : le processus d'identification de la demande de changement, et le processus de la soumission. Par contre, la phase de l'évaluation est formée de quatre processus internes. Effectivement, cette

phase est la plus intéressante dans notre démarche, car elle contient le processus d'analyse qu'on a présenté dans le paragraphe précédent. Ce processus, sert à analyser l'impact de changement sur la sécurité. Nous allons maintenant présenter chaque phase en détaillant ses propres processus.

5.1. Phase d'initialisation

La première phase de toute demande de changement est la phase d'initialisation. Comme la figure ci-dessus le montre, cette phase est divisée en deux processus successifs qui sont : le processus d'identification de la demande de changement, et le processus de la soumission de la demande de changement.

5.1.1. Processus d'identification

Le processus d'identification de la demande est le premier processus rencontré durant le cycle de vie de la demande de changement. Cette phase implique plusieurs activités principales qui incluent : l'attribution d'un numéro à la demande pour faciliter sa poursuite tout le long de son cycle de vie, le type de changement, la vérification si la demande de changement est une nouvelle demande ou non, l'origine du changement, la nature du changement (améliorer une fonctionnalité, diminuer le coût total, etc.), et la justification de la demande de changement.

5.1.2. Processus de Soumission

Après l'exécution du processus de l'identification de la demande, les parties prenantes doivent passer vers le deuxième processus présenté dans la première phase du cycle de vie d'une demande de changement. Ce processus signifie que toutes les informations concernant la demande de changement sont prises en compte, et que la demande doit passer vers la deuxième phase pour analyser son impact.

5.2. Phases d'évaluation

Dans le paragraphe précédent, on a présenté la phase d'initialisation de la demande. En effet, la phase précédente peut être utilisée pour toute demande de changement. Mais, le processus d'analyse de l'impact du changement sur un aspect bien précis comme la sécurité, délai, ou le coût doit être instruite dans cette phase. Par la suite, nous allons présenter le détail de cette phase en présentant : le processus de traçabilité, l'évaluation de la demande de

changement, le processus de classification pour la demande de changement, et finalement le processus d'approbation de changement.

5.2.1. Processus de Traçabilité

Tous les processus utilisés auparavant sont employés pour donner des informations nécessaires afin d'évaluer l'impact de la demande de changement sur la sécurité. Mais, toutes les informations déjà données ne sont pas suffisantes pour pouvoir analyser l'impact sur la sécurité, car elles ne donnent pas la trace de la demande de changement. Alors, le processus de traçabilité est utilisé pour connaitre le déploiement de ce changement au niveau du système de sécurité, afin d'effectuer l'analyse nécessaire sur la sécurité.

5.2.2. Processus d'évaluation d'impact

Pratiquement, l'évaluation d'impact sur la sécurité est divisée en deux types d'analyse : au début la recherche des liens entre la demande de changement et le système de sécurité, puis une analyse d'impact formelle comme on a déjà présenté (voir § 4.3.2). En effet, le premier type d'analyse de traçabilité peut être considéré comme une analyse préliminaire (analyse à priori). Le deuxième type d'analyse, peut être considéré comme une analyse plus formelle, vu l'utilisation des techniques formelles des vérifications pour les contraintes de sécurité.

Enfin, le processus d'évaluation d'impact n'est pas suffisant, car il est limité pour vérifier la validité des contraintes de sécurité, et pas toutes les exigences sécuritaires. En effet, dans le cas où les exigences pures de sécurité ou les exigences significatives de sécurité sont affectées par la demande de changement, il faut utiliser d'autres techniques qu'on n'a pas précisé au niveau de ce processus.

5.2.3. Processus de Classification

Le but de ce processus est de classifier les demandes de changement. En effet, toutes les demandes de changement ne sont pas forcement des nouvelles demandes. Parfois, les parties prenantes soumettent des demandes de changement qui sont identiques. Alors, pour ne pas répéter l'analyse de la demande plusieurs fois, le but de ce processus est de classifier les demandes de changement qui se ressemblent.

5.2.4. Processus d'approbation de la demande

Après une analyse d'impact de la demande de changement, la sortie de ce processus doit être la décision d'appliquer ou pas la demande de changement. En fait, à ce niveau il faut faire

attention à ce que l'autorisation d'appliquer une demande de changement est donnée en s'assurant que la demande n'a pas d'impact sur la sécurité. Mais, c'est possible que la demande de changement ait un impact sur d'autres attributs comme le coût et le délai du développement. Enfin, au cours de notre problématique la recherche d'impact sur la sécurité est une clé pour accepter ou refuser l'application des demandes de changement.

5.3. Phase de réalisation

L'exécution de cette phase aura lieu quand la demande de changement proposée a été acceptée. Par la suite nous allons détailler les deux processus présentés dans cette phase et qui sont : le processus pour l'application de la demande, et le processus pour établir des nouveaux liens de traçabilité.

5.3.1. Application de la modification

L'application de la demande de changement dépend de plusieurs facteurs. Parmi ces facteurs, on peut citer : la phase de la soumission de la demande de changement, et de la structure utilisée pour le développement du système. Par exemple, la politique de l'application de la demande de changement à la phase des exigences même s'il n'y a pas un impact sur la sécurité, n'est pas la même politique lorsque la demande de changement a lieu dans la phase de conception ou bien à la phase de l'implémentation.

Comme notre étude est située dans le contexte de l'EIA-632, nous avons divisé l'application de la demande de changement en trois étapes.

Premièrement, on effectue une recherche pour connaître le module de plus haut niveau impacté par la demande. Après cette recherche, il faut changer : les exigences techniques présentées à ce niveau et impactées par la demande de changement, la solution logique, la solution physique, la conception du module…. La troisième étape, est de continuer avec la même démarche pour les autres modules qui se trouvent dans le niveau juste en bas, du module impacté. Enfin, de l'application du changement s'arrête exactement dans la phase où on a eu l'apparition de la demande de changement.

5.3.2. Nouveaux liens de traçabilité

Pratiquement, après l'application de la demande de changement aux systèmes en cas de développement, il y aura des nouveaux liens de traçabilité. Car, une application de

modification implique la suppression ou l'addition des liens, lors de la suppression ou l'addition d'une exigence ou respectivement d'un module.

En effet, ce processus n'a aucune influence sur l'analyse d'impact avec la demande actuelle. Mais, il joue un rôle majeur pour les nouvelles demandes de changement. Alors, le but de ce processus est de mettre à jour les liens au sein du système de développement. Car sinon, la nouvelle application d'une demande de changement sera quasiment impossible, vue l'impossibilité de connaitre les vrai parties du système impactées par une telle demande de changement.

5.4. Phase de validation

Après la réalisation et l'intégration de la demande de changement, au cours du système de développement. Il faut passer par la validation de la demande de changement. Alors, la phase finale du cycle de vie de la demande de changement est la phase de validation. Cette phase a pour but, de valider la bonne réalisation de la demande de changement, et de vérifier que ce qui a été présenté et demandé au niveau de la phase de l'initialisation de la demande corresponde bien a ce qui a été réalisé. Par la suite, on présente les deux processus de cette phase et qui sont : le processus de test du changement et le processus de validation.

5.4.1. Processus de test

Plusieurs techniques existent pour la phase du test [BAC-99]. En effet, lors de la phase de l'identification, il faut que les parties prenantes précisent la nature du test de la demande de changement. Ce processus sert à appliquer un ensemble de tests de cas, puis à envoyer les résultats des tests avec des fichiers vers les parties prenantes concernées par la phase de validation, pour qu'ils puissent exécuter le processus de validation.

5.4.2. Validation de la modification

Une fois l'ensemble des tests effectués, les validateurs peuvent décider si la réalisation de la modification corresponde bien à ce qui a été présenté dans la demande de changement ou non. Si oui, la demande de changement sera validée. Autrement, il sera possible de générer d'autres demandes de changement et de commencer avec un autre cycle pour l'application et l'intégration de la nouvelle demande de changement au sein du développement du système.

6. SYSTEME D'INFORMATION

Dans le paragraphe ci-dessus, on a présenté les principales phases du cycle de vie d'une demande de changement. En effet, le cycle de vie d'une modification est formé de quatre phases. Cependant, le diagramme ci-dessous (cf. Figure III.12), montre qu'une modification passe par six états, qui sont comme suit : une modification soumise, une modification évaluée, une modification rejetée, une modification approuvée, une modification réalisée, une modification validée.

Rôle	**Description et responsabilité**
Demandeur	Personne qui soumet la demande d'évolution.
Evaluateur	La personne commanditée par le gestionnaire de projet pour analyser l'impact de l'évolution proposée.
Responsable	Le groupe qui décide d'approuver ou de rejeter les évolutions proposées sur un projet spécifique.
Modificateur	Personne responsable des évolutions sur le produit en cours, en réponse à une demande d'évolution approuvée, met à jour le statut de la demande après coup.
validateur	La personne qui détermine si les modifications ont été correctement réalisées.

Tableau 4-les parties concernées avec une demande de changement.

Dans le tableau ci-dessus, on présente les rôles des personnages liées à une telle modification, tout au long son cycle de vie. En effet, lors de la demande de changement, on a déjà précisé qu'il faut exécuter au début le processus de l'initialisation, après cette exécution l'état de la demande de modification devient **Soumise**. Puis, après la soumission il y aura une intervention de la part du vérificateur pour qu'il vérifie tout type d'impact. Cependant, ce qui nous intéresse est de vérifier l'impact sur la sécurité, en passant vers le modèle formel et fonctionnel du système dans le but de vérifier la validité des contraintes de sécurité. Après cette étape, l'état de la modification devient **Evaluée**.

Après la vérification, les parties prenantes de la demande de modification (Responsable) vont décider si la demande sera **Rejetée** ou **Approuvée**. Enfin, dans le cas où la demande sera approuvée, la modification sera intégrée au cours du développement du système pour qu'elle soit **réalisée** par un réalisateur puis **validée** par un validateur.

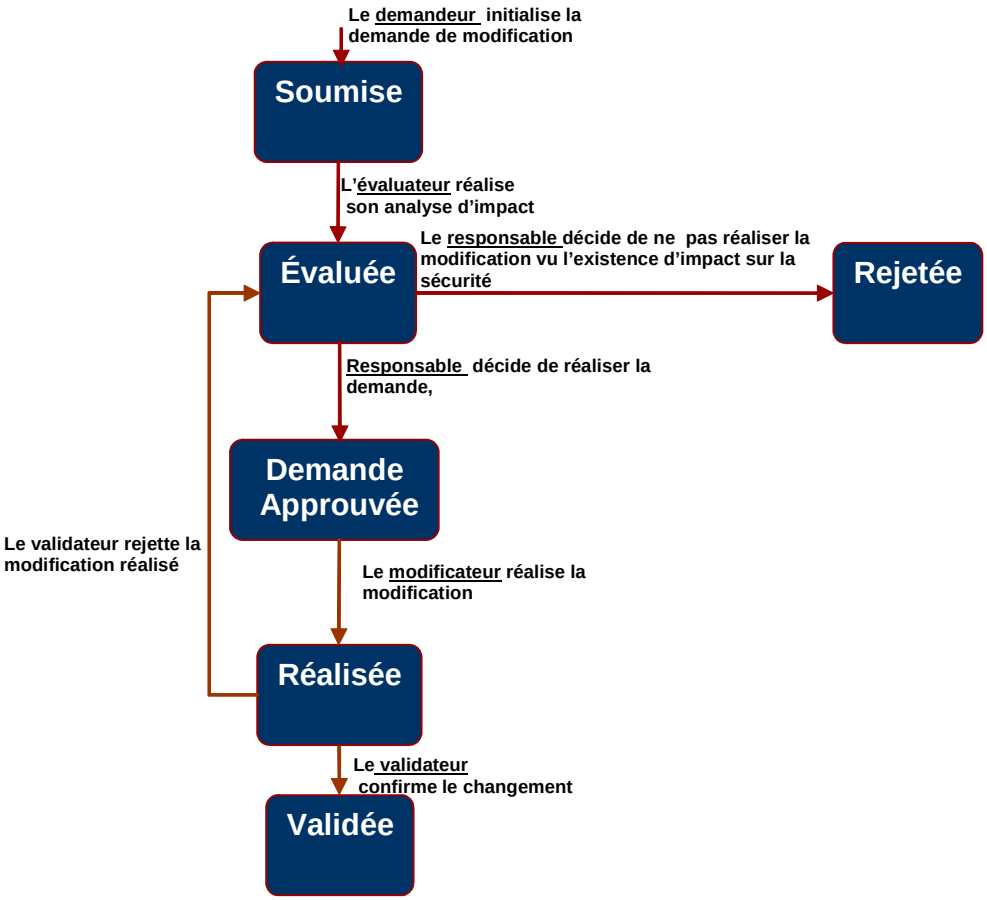

Figure III.12-Le diagramme états transitions pour la demande de changement.

7. CONCLUSION

Dans ce chapitre, nous avons présenté la méthodologie de la recherche d'impact de la modification au cours du développement. Un échec de vérification lors de la demande de changement simple peut avoir comme conséquences : des pertes économiques significatives, des dommages physiques et des menaces pour la vie humaine.

La maîtrise du changement tout au long du cycle de vie de développement du système, dépend de plusieurs facteurs. Parmi ces facteurs, on peut citer: la bonne maîtrise de la gestion de la modification, la bonne maîtrise de l'analyse de l'impact, une très bonne maîtrise de l'analyse de la sécurité, cela exige une mise en place d'un modèle de traçabilité.

Le chapitre suivant contient le détail de la conception de l'outil que nous avons développé au sein de l'IIST. Cet outil a été conçu pour faciliter la recherche d'impact sur la sécurité lors de la demande du changement. En fait, cette conception présente en détail les 3 modèles discutés dans ce chapitre. Premièrement, elle présente le modèle de développement du

système complexe selon l'EIA-632, deuxièmement elle contient le modèle de traçabilité entre les différentes parties du système de développement (en présentant quelques types de traces supportées par l'outil), et finalement le modèle du changement qui interagit avec les deux autres modèles.

Au début du chapitre suivant nous allons présenter la conception de l'outil avec les notations d'UML, en utilisant : le diagramme du cas d'utilisation, et le diagramme de classe. Puis, nous allons passer à la spécification formelle de l'outil en utilisant RSL pour : éliminer l'ambiguïté présentée au niveau des diagrammes UML, et spécifier formellement les fonctionnalités principales pour la recherche d'impact sur la sécurité lors d'une demande de changement.

CHAPITRE IV

SPECIFICATION ET OUTILLAGE

1. INTRODUCTION

Actuellement, l'augmentation du taux de changement et des technologies implique beaucoup de mises à jour du système durant son développement. Plusieurs aspects facilitent la mise à jour des systèmes, parmi lesquels la traçabilité. Malheureusement, une mauvaise recherche de trace et une mauvaise structuration du système à développer rendent la recherche d'impact quasiment impossible. Dans notre outil, la traçabilité a un rôle important pour la mise à jour du changement, et pour l'analyse d'impact. Car, la traçabilité nous aide à connaître les parties impactées par une telle demande de changement et pour valider que la demande a bien été établie [ELJ-05b] [ELJ-05c] [ELJ-05d].

Dans le chapitre précédent, nous avons présenté la méthodologie de recherche d'impact sur la sécurité. Effectivement, le troisième chapitre était divisé en plusieurs parties comme : les traces et les liens d'un module, la gestion de la modification, l'analyse d'impact avec la nécessité de procéder à une vérification formelle, et le cycle de vie de la modification.

Dans ce chapitre, la formalisation avec RSL est utilisée pour spécifier les fonctionnalités de notre outil. En effet, l'outil supporte la méthodologie de recherche d'impact sur la sécurité lors d'une demande de changement [ELJ-04a]. Malgré que notre outil n'intègre pas la capacité à réaliser la vérification formelle des contraintes de sécurité ; mais il doit indiquer si le système de sécurité est impacté par une telle demande de changement lors de sa soumission.

Un des buts principaux de ce chapitre, est de montrer explicitement tous les types des relations présents au cours du développement de système (au niveau du diagramme de classes). Cette représentation explicite nous permet de savoir si le système de sécurité est impacté par la demande, afin d'effectuer les analyses nécessaires, soit en appliquant des vérifications formelles ou bien en appliquant d'autres techniques. Par la suite, nous allons

présenter une vue globale du développement de l'outil en utilisant les diagrammes d'UML, puis nous allons passer vers la spécification pour les fonctionnalités de l'outil, et finalement nous terminons le chapitre par un cas d'étude.

2. ARCHITECTURE DE L'OUTIL

Une architecture consiste à créer une représentation simplifiée pour les fonctionnalités d'un système désiré. En effet, l'architecture du système facilite la compréhension de sa mission.

Par exemple, l'utilisation de la spécification semi formelle comme UML, nous permet de définir et de visualiser un modèle à l'aide de plusieurs diagrammes. En effet, un diagramme UML est une représentation graphique, qui s'intéresse à tel aspect. Chaque type de diagramme présent dans UML, possède une structure bien précise. Ci-dessous, nous allons présenter: le diagramme de classe, et le diagramme de cas d'utilisation, afin de montrer les fonctionnalités primaires de notre outil.

2.1. Diagramme de cas d'utilisation

Les diagrammes de cas d'utilisation sont utilisés pour donner une vision globale du comportement fonctionnel d'un système. Ainsi, ils permettent d'identifier les fonctionnalités principales et critiques du système. Ci-dessous, nous allons présenter les deux composants principaux des diagrammes de cas d'utilisation qui sont : les acteurs et les cas d'utilisation.

Les acteurs sont des entités externes, qui interagissent avec le système (comme une personne humaine). Une personne physique peut jouer le rôle de plusieurs acteurs d'un système. C'est pourquoi les acteurs doivent surtout être décrits par leur rôle ; ce rôle décrit les besoins et les capacités de l'acteur. En effet, les acteurs sont représentés par un pictogramme humanoïde sous-titré par le nom de l'acteur (ou bien un rôle).

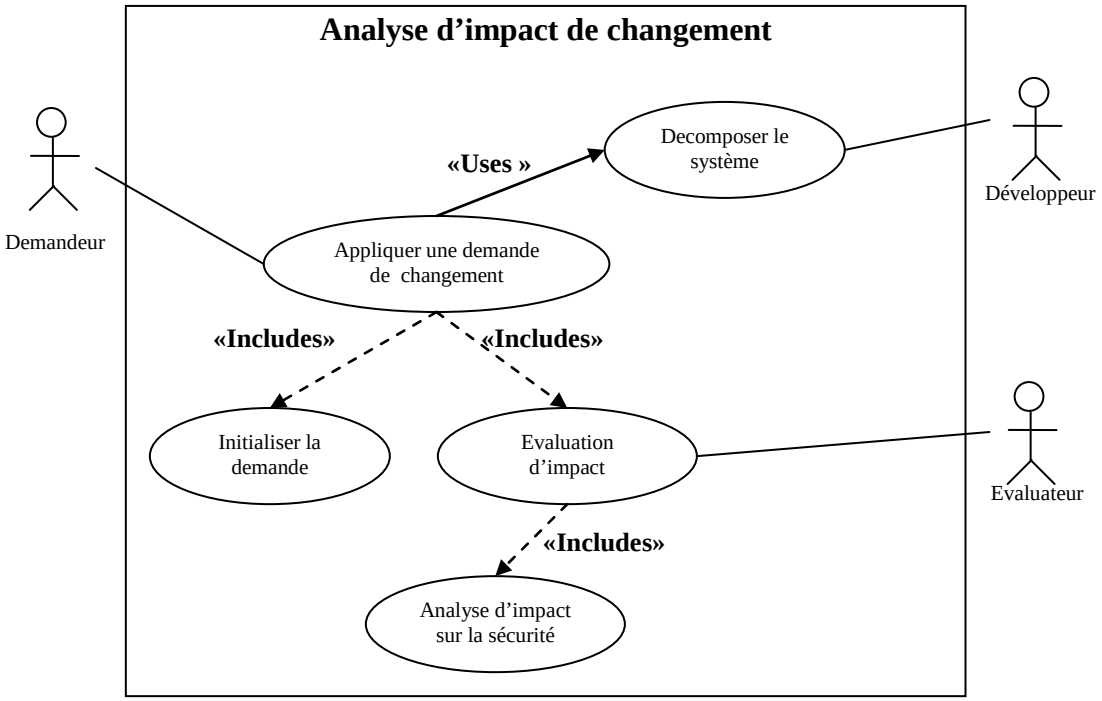

Figure IV.1- le diagramme de cas d'utilisation

Le cas d'utilisation (use case), permet de décrire l'interaction entre l'acteur et le système. Un cas d'utilisation ne décrit pas une solution d'implémentation, mais il est une description des interactions qui vont permettre à l'acteur d'atteindre son objectif en utilisant le système. En effet, il est représenté par une ellipse sous-titrée par un nom (éventuellement le nom est placé dans l'ellipse). Un acteur et un cas d'utilisation sont mis en relation par une association représentée par une ligne.

Dans la figure ci-dessus (cf. Figure IV.1), on présente le diagramme de cas d'utilisation de notre outil. En fait, l'outil supporte deux cas d'utilisation principaux :

- Décomposition du système désiré, selon la norme de l'EIA-632,
- L'application d'une demande de changement.

La fonctionnalité de décomposition du système selon l'EIA-632 joue un rôle majeur pour l'utilisateur de notre outil. Car, cette première fonctionnalité doit permettre aux parties prenantes, de développer le système selon la décomposition présentée par la norme de l'EIA-632. Cette décomposition est une décomposition du système désiré avec : des projets, des modules, produits capacitants, produits finaux et des technologies capacitants. Durant la présentation du diagramme de classes, nous allons expliquer le détail de ce premier cas d'utilisation.

Le deuxième cas d'utilisation de l'outil est de permettre à une analyse d'impact lors de la demande de changement. Dans notre outil, l'analyse d'impact d'une modification sur la sécurité est basée sur deux types d'analyses différentes (révise chapitre 3, § 4.3.2) :

- La première analyse est une analyse préliminaire pour les constituants impactés par la modification Ces constituants peuvent être des exigences sécuritaires, des exigences techniques, des modules, des fichiers, des produits capacitants, des technologies capacitants.
- La deuxième analyse est une analyse d'impact formelle. Cette analyse sera établie, en analysant les fichiers impactés (qui correspondent à la solution logique d'un tel module et qui contiennent la spécification formelle) par la modification, afin de vérifier la validité des contraintes de sécurité.

Réellement, notre outil ne doit pas supporter une analyse formelle sur la sécurité. Mais, il doit s'interfacer avec un autre outil comme RSL, afin d'analyser l'impact formellement.

2.2. Diagramme de classes

Dans ce paragraphe, nous allons présenter la spécification de l'outil[25] en utilisant le diagramme de classes. En effet, le diagramme des classes est un schéma utilisé en génie logiciel pour présenter les classes d'un système, ainsi que les différentes relations entre celles-ci. Il fait partie de la représentation statique d'UML car il fait une abstraction des aspects temporels et dynamiques. Précisément, une classe décrit les responsabilités, le comportement et le type d'un ensemble d'objets. Les éléments de cet ensemble sont les instances de la classe. Une classe est définit avec des fonctions (méthodes) et des données (attributs) qui sont liées ensembles par un champ sémantique.

Effectivement, les classes permettent de modulariser un programme et ainsi de découper une tâche complexe en plusieurs petits travaux simples. Elles peuvent être liées entre elles, avec des associations. Chaque association de ces relations est représentée par un arc spécifique dans le diagramme de classes. Dans notre cas d'étude, on n'a pas présenté les attributs d'une classe, vu que nous voulons les présenter formellement avec RSL.

[25] Spécifié au sein de l'IIST-UNU

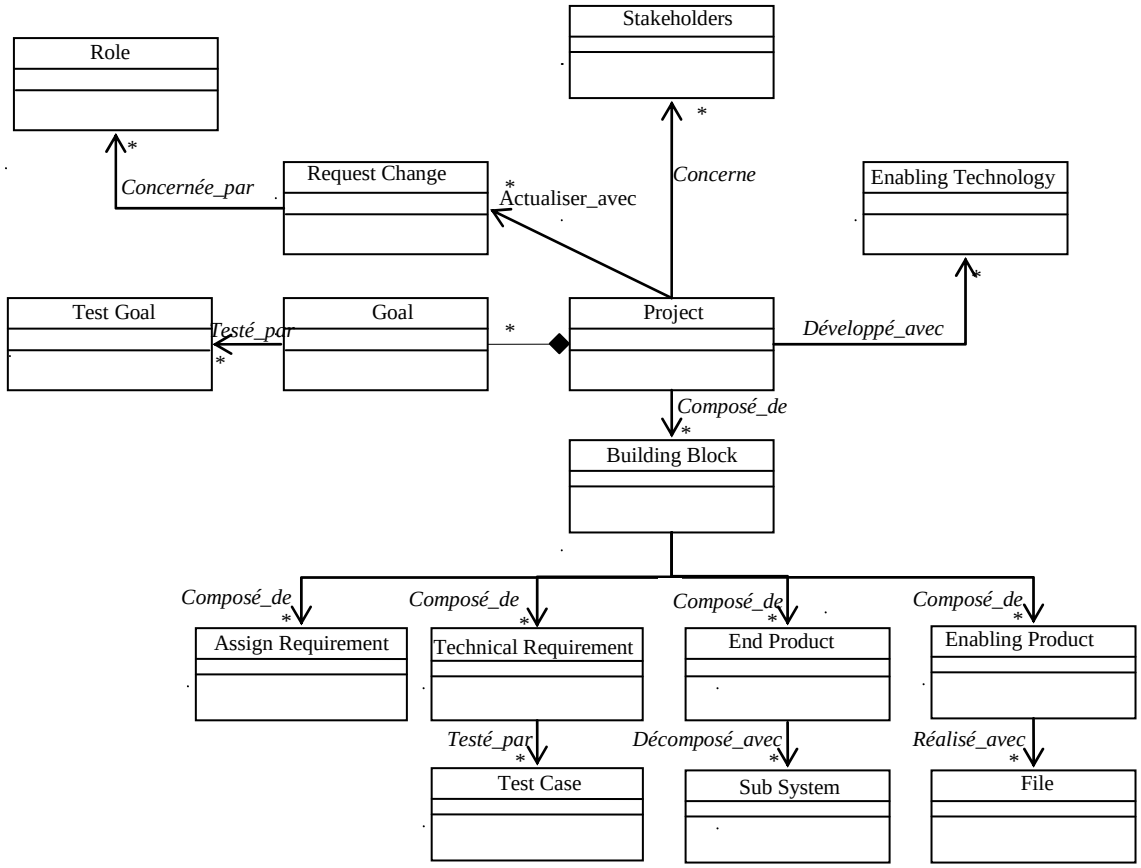

Figure IV.2-Le diagramme de classes

En réalité, les sources d'information durant le développement du système seront présentées dans différents types de documents. Ces documents, contiennent les informations techniques ou non pour le développement du système. Par exemple, des documents contiennent des informations sur les parties prenantes liées au développement du système, leurs adresses, les numéros des téléphones ; les modules ; les produits finaux; les produits capacitants; ainsi que les fichiers qui correspondent au système de développement d'un module,

Dans le diagramme ci-dessus, on remarque que la traçabilité est représentée par l'ensemble des relations qui relient les classes entre eux (*concernée_par*, *concerne*, *développé_avec*, *testé_par*, *décomposé_avec*, etc…). Ces relations représentées explicitement, nous permettent par exemple de connaître les exigences techniques, les produits finaux, les produits capacitants d'un module donné.

3. UML VERS RSL

L'utilisation des diagrammes UML nous a aidé, d'une part pour donner une vision assez simplifiée de notre outil, et d'autre part pour générer la spécification de l'outil avec RSL en utilisant le traducteur UML2RSL[26]. En effet, la spécification formelle générée automatiquement par le traducteur est une spécification primaire, car le traducteur génère le squelette de la spécification [GEO-99]. Pour cela, nous avons spécifié manuellement le contenu de chaque fonction de l'outil, afin d'obtenir une spécification concrète. Le choix de spécifier les fonctionnalités de notre outil explicitement avec RSL est basé sur deux raisons: premièrement pour éliminer l'ambiguïté présentée lors de l'utilisation des diagrammes UML. Deuxièmement, pour appliquer un ensemble de tests sur la spécification concrète, surtout avant le codage de l'outil avec un tel langage de programmation. L'utilisation du traducteur UML2RSL est composée de trois étapes principales:

- Une saisie graphique du diagramme de classes.
- Une génération automatique du diagramme de classes, dans un format XML.
- Un parsing pour le fichier obtenu en XML (en utiulisant le traducteur UML2RSL), afin d'obtenir le squelette de la spécification de l'outil avec RSL. En effet, le traducteur vérifie la syntaxe du diagramme de classes avant la génération automatique de la spécification primaire, sinon il génère des messages d'erreurs.

Le résultat de la génération du diagramme UML en RSL est modulaire. Cette génération consiste en plusieurs fichiers. L'un des fichiers générés par le traducteur est nommé SYSTEM.rsl. Le module System.rsl correspond au module du plus haut niveau de structuration de la spécification en RSL. Il contient la spécification de toutes les fonctionnalités présentées dans les autres modules auxiliaires. Lors de la génération de la spécification vers RSL, chaque classe est traduite avec deux modules. Le premier module (ex. PROJECT) représente la classe avec ses attributs et il prend le même nom que la classe. Le deuxième module (ex. PROJECTS) représente l'ensemble des objets existants d'une même classe.

[26] Le traducteur UML2RSL est développé au sein de l'IIST-UNU

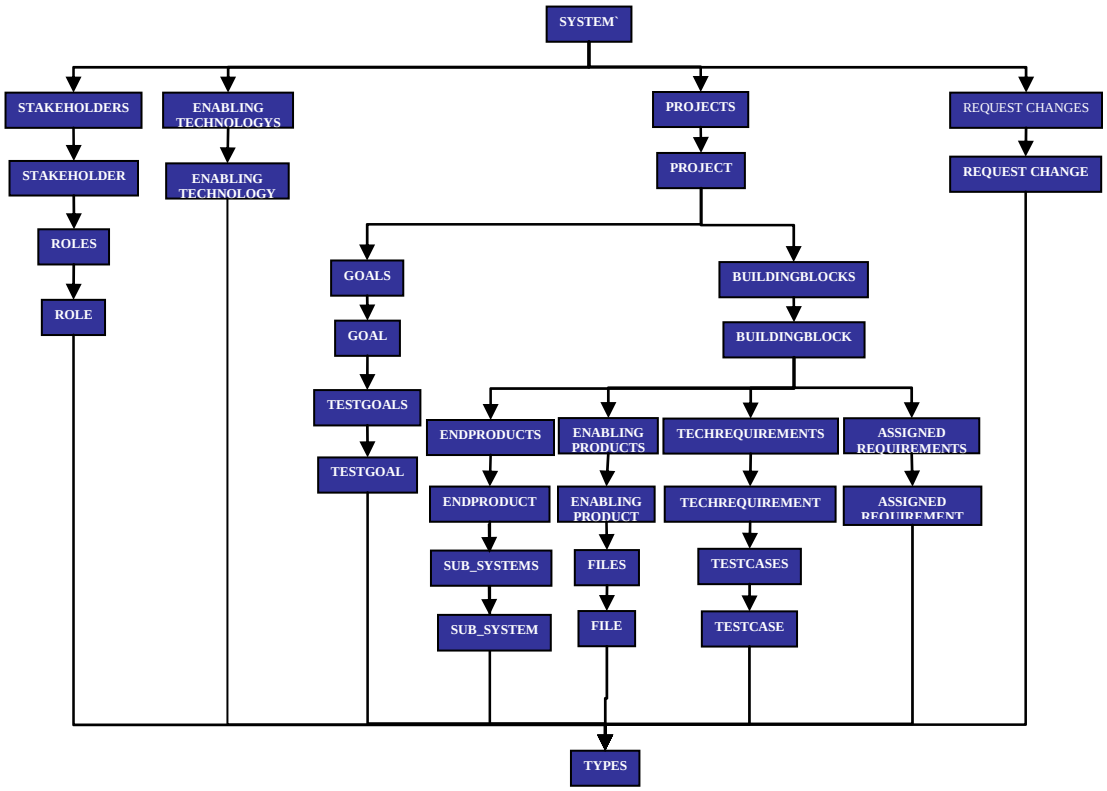

Figure IV.3-La structuration de la spécification de l'outil avec RSL

Enfin, dans le niveau le plus bas de la spécification avec RSL on peut remarquer l'existence du module «TYPES». Dans ce module, on a défini tous les types utilisés au cours de la spécification. Par la suite, nous allons présenter quelques modules de la spécification formelle comme le module: types, Système, Projet, produits capacitants, et technologie capacitante.

3.1. Les types

Le pas initial lors de la spécification formelle avec RSL pour n'importe quel type de système est la définition des types. D'après [GEO-99], on peut dire qu'un type est la collection des valeurs logiquement reliées. Certains types sont prédéfinis au niveau de RSL comme le Bool, Real, etc…D'autres doivent être définis par le spécificateur de l'application. Effectivement, plusieurs manières existent pour la définition des types. Parfois, au niveau d'une spécification en RSL, on trouve l'existence des types abstraits et d'autres types déclarés ou concrets. Par exemple, touts les types définis dans le module types (TYPES.rsl) sont des types concrets (cf. annexe E).

3.2. Système

Le module de plus haut niveau dans l'architecture de l'outil est le module SYSTEM.rsl. Au sein de ce module, on a spécifié toutes fonctionnalités de l'outil comme les fonctionnalités : de la recherche d'impact, de l'ajout/suppression d'une exigence technique, de l'ajout/la suppression d'un projet, de l'ajout/la suppression d'un module, etc…

Sys ::
stakeholders: STAKEHOLDERS_.Stakeholders <-> *replace_stakeholders*
projects: PROJECTS_.Projects <-> *replace_projects*
requestchanges: REQUESTCHANGES_.RequestChanges <-> *replace_requestchanges*
enablingtechnologys :ENABLINGTECHNOLOGYS_.EnabTechs <-> *replace_enablingtechnologys*

Figure IV.4-Le développement d'un système est formé d'un ensemble des parties prenantes, des projets, des demandes de changements, et des technologies capacitants

Dans la figure ci-dessous (cf. Figure IV.4), on présente les attributs de la classe SYSTEM.rsl spécifié formellement. A ce niveau, on peut facilement remarquer qu'un système est formé par un ensemble de projets, un ensemble de parties prenantes, un ensemble de demandes de changement pour la mise à jour du système, et un ensemble de technologies utilisées pour le développement du système.

3.3. Projet

Selon l'EIA-632, le développement du système doit est divisé en plusieurs projets. Parmi les fonctionnalités qui doivent être supportées par tout outil de gestion des exigences, on trouve les fonctionnalités d'ajouter, supprimer et afficher un objet. Dans la figure ci-dessous (cf. Figure IV.5), on présente la spécification formelle de la classe Project. La spécification formelle de la classe Project, est formée de sept attributs. Les cinq premiers seront employés pour la description du projet. Par contre, les deux derniers attributs, sont là pour aider à identifier les modules et les buts d'un projet donné.

Project::
project_name : Project_Id <-> *replace_projname*
project_manager_id : Stakeholder_Id <-> *replace_projmanager*
project_date : Date <-> *replace_projdate*
project_description : Description <-> *replace_projdesc*
project_validation : Validation <-> *replace_projvalidation*
goals: GOALS_.Goals <-> *replace_projgoals*
buildingblocks : BUILDINGBLOCKS_.BBlocks <-> *replace_buildingblocks*

Figure IV.5-spécification formelle de la classe project

L'attribut *project_validation*, joue un rôle majeur lors d'une demande de changement. Dans le cas où le projet n'est pas validé par les parties prenantes, l'utilisateur ne sera pas obligé de passer via la soumission d'une demande de changement, pour la mise à jour du système. Autrement, et si le projet est validé, les parties prenantes doivent suivre un cycle de vie pour la demande de changement, afin d'effectuer les analyses nécessaires sur la sécurité.

```
add_project :
      Project_Id >< Stakeholder_Id >< Date ><
      Description >< Validation >< GOALS_.Goals ><
      BUILDINGBLOCKS_.BBlocks >< Sys -> Sys
    add_project(
      id, manager, date, description, valid, goals, bbs,
      system)
          is
      update_projects(
        PROJECTS_.add_project(
          id, manager, date, description, valid, goals,
          bbs, projects(system)), system)
    pre
      can_add_project(
        id, manager, date, description, valid, goals,
        bbs, system),
```

Ci-dessous, nous avons détaillé la spécification de la fonction *add_project* au sein du module SYSTEM.rsl. En RSL, avant le passage vers la spécification formelle du contenu d'une telle fonction, on doit définir sa signature. Cette signature doit correspondre aux attributs déjà définis dans la figure qui présente la classe Project. Par la suite on présente un simple exemple de la spécification formelle de la fonction *add_project* qui est définit au plus haut niveau de la structure de la spécification (SYSTEM.rsl). En effet, pour cette fonction il y aura l'existence d'une pré-condition qui sert à tester la possibilité d'ajouter un projet ou non durant le développement de système.

La signature de la fonction *function* spécifiée en RSL, et la signature de la fonction *can_function* qui joue le rôle d'une pré-condition, doivent être pareilles. Pour cela, dans l'architecture de notre outil on a toujours suivi la même règle citée précédemment. Alors, la signature de la fonction *can_add_project* a la même signature que la fonction *add_project*. En effet, la fonction *can_add_project* est spécifiée au niveau du module SYSTEM.rsl, et elle est divisée en deux parties spécifiées formellement. La première partie, concerne l'appel de la fonction de *can_add_project* au niveau du module PROJECTS. Par contre, la deuxième partie de la fonction *can_add_project* sert à tester la possibilité de la mise à jour de l'ensemble des projets utilisés pour le développement du système.

```
can_add_project :
    Project_Id >< Stakeholder_Id >< Date ><
    Description >< Validation >< GOALS_.Goals ><
    BUILDINGBLOCKS_.BBlocks >< Sys ->
      Bool
  can_add_project(
    id, manager, date, description, valid, goals, bbs,
    system) is
    PROJECTS_.can_add_project(
      id, manager, date, description, valid, goals,
      bbs, projects(system)) /\
    let
      new_projects =
        PROJECTS_.add_project(
          id, manager, date, description, valid, goals,
          bbs, projects(system))
    in
      preupdate_projects(new_projects, system)
    end,
```

3.4. Produit Capacitants

Suivant notre architecture proposée au début de ce chapitre, on peut remarquer que chaque projet est formé d'un ensemble de modules, et à son tour chaque module sera divisé en plusieurs produits capacitants. Les produits capacitants ont pour but de réaliser le produit final, à un niveau donné de la décomposition du système. Par ailleurs, les explications des fonctionnalités de l'outil présenté dans ce chapitre, ne suffisent pas pour comprendre complètement toutes les fonctionnalités attendues l'outil. Mais, elles sont indispensables pour donner une vision globale de l'outil qu'on a spécifié.

En effet, plusieurs types de produits capacitants existent au cours du développement d'un module (cf. Figure IV.6). Parmi les types de produits capcacitants, on rencontre : le développement du produit final, le test du produit final, la production du produit final, la formation pour l'utilisation du produit final, le déploiement du produit final, le support du produit final, et la mise hors service du produit final. La spécification formelle des différents types (cf. Figure VI.7) d'un produit capacitant sera spécifiée au niveau du module TYPES.rsl.

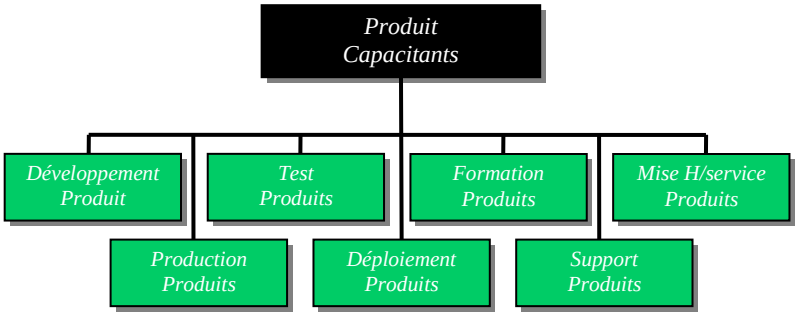

Figure IV.6- Les différents types des produits capacitants

```
EnablingProduct_type = =Product_Developpement |
                        Test_Product |
                        Production_Product |
                        Training_Product |
                        Deployement_product |
                        Support_Product |
                       Operation_Product
```

Figure IV.7-Les différents types des produits capacitants spécifiés avec RSL

Ci-dessous, nous allons présenter les attributs de la classe produit capacitant (EnabpPro). En effet, la classe est formée de quatre attributs. Les trois premiers attributs seront utilisés pour donner un identificateur, une description et un type pour le produit capacitant. Le type du produit capacitant sert à savoir sa mission, car il peut être utilisé pour le développement du produit final, la production du final, etc. Enfin, le dernier attribut de la classe produit capacitant, sert à connaitre l'ensemble des fichiers utilisés en cours de développement du système.

```
EnabPro ::
    enabPro_id : EnablingProduct_Id <-> replace_Id
    enabPro_description : Description <-> replace_description
    enabPro_enabtype : Enabling_Type <-> replace_enabtype
    enabPro_files :FILES_.Files <-> replace_files
```

Figure IV.8-Spécification formelle de la classe produit capacitant

3.5. Les fichiers

Au cours du développement des systèmes complexes, les parties prenantes utilisent de plus en plus la notion de fichiers. Les fichiers contiennent toutes les informations, car tous types de

sources d'informations sont généralement existants dans des fichiers [MES-05] [MES-06]. Dans notre cas d'étude, on considère que : tout ce qui concerne le développement d'un tel module, doit être présent au sein de fichiers différents. D'où l'importance primordiale, d'utiliser la notion de fichier au cours du développement des systèmes.

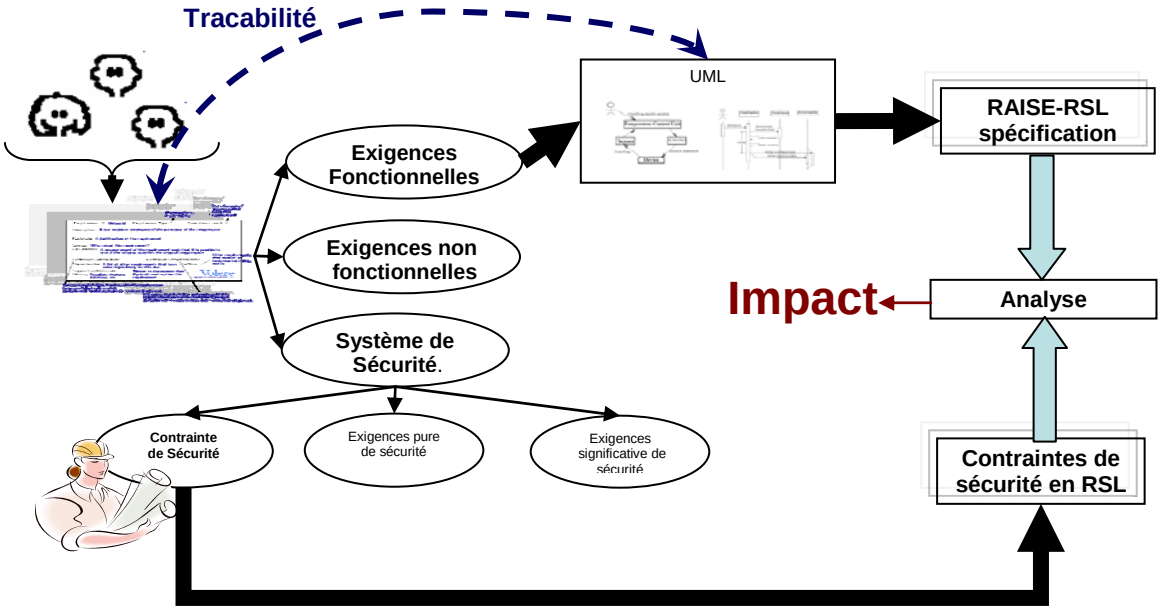

Figure IV.9- L'analyse d'impact formelle, sur la sécurité pour la solution logique d'un tel module impacté par la demande de changement.

Effectivement, on considère que la solution logique d'un produit final à l'intérieure d'un tel module, sera réalisée au sein d'un fichier. Et, que cette solution spécifie formellement le comportement du produit final. En effet, lors d'une demande de changement, l'analyse établie sur le fichier contenant la spécification formelle du produit final, concerne l'analyse d'impact formelle.

3.6. Exigences Techniques

Plusieurs formats existent pour représenter une exigence, parmi lesquels VOLERE [TEM-Vol]. En effet, durant la spécification de notre outil nous avons proposé un format pour la représentation des exigences techniques. L'ensemble des exigences techniques d'un tel module, forme le SRS du module.

Ci-dessous, on présente la spécification formelle du format des exigences techniques. Notre format est inspiré du format VOLERE, en ajoutant d'autres attributs qui nous aident à

chercher l'impact sur la sécurité. Parmi les nouveaux attributs on peut citer l'attribut : *Version, hasparent, updated_by.*

L'attribut Version, nous aide à connaître l'historique de l'exigence. Par contre, l'attribut *hasparent*, nous aide à savoir si cette exigence est descendante d'une autre exigence. L'attribut *updated_by* nous aide à savoir les identificateurs des demandes de changement, qui ont été soumises pour changer cette exigence technique. Finalement, l'attribut *testcases* qui nous aide à savoir l'ensemble des tests qu'il faut appliquer sur le système avant sa livraison, afin de valider la bonne réalisation de cette exigence.

TechRequirement ::
id : TechRequirement_Id <-> replace_Identificator
fit_Cretirion : Fit_Cretirion <-> replace_Fit_Cretirion
description : Description <-> replace_Description
requirement_Type : Requirement_Type <-> replace_Requirement_Type
date : Date <-> replace_Date
satisfaction : Satisfaction <-> replace_Satisfaction
dissatisfaction : Dissatisfaction <-> replace_Dissatisfaction
dependency : Dependecy <-> replace_Dependecy
rational : Rational <-> replace_Rational
version : Version <-> replace_Version
source : Stakeholder_Id <-> replace_source
updated_by : RequestChange_Id-set <-> replace_updated_by
hasparent : Parent_Id <-> replace_parent
testcases : TESTCASES_.TestCases <-> replace_testcases

Figure IV.10-La spécification d'une exigence technique avec RSL

Enfin, parmi les autres attributs déjà existants dans le format VOLERE on peut citer : l'attribut rational qui est une justification pour pouvoir intégrer une telle exigence technique parmi le SRS du module, l'attribut *satisfaction* et l'attribut *dissatisfaction* sont utilisés pour indiquer le degré de la satisfaction et la non satisfaction du client après la réalisation du système, et l'attribut *fit_Creterion* nous aide à savoir comment le système après sa réalisation satisfait cette exigence technique.

3.7. Enabling Technology

La définition anglaise de la technologie est : « the practical application of science to commerce or industry ». La traduction de la définition en français est : l'application pratique de la science pour le commerce ou l'industrie. En fait, le développement des systèmes complexes doit être basé sur un ensemble de technologies, pour cela on a intégré une classe juste pour présenter l'ensemble des technologies utilisées durant le développement. Cet ensemble des technologies est, soit des technologies de test et de vérification, soit des

technologies de conception physique, ou bien d'autres technologies à la base du développement de système complexe.

3.8. Demande de changement

Comme présenté dans le chapitre 3, le cycle de vie de la demande de changement est composé de 4 phases : la phase d'initialisation, la phase d'évaluation, la phase de réalisation, et finalement la phase de validation. Dans la figure ci-dessous (cf. Figure IV.11), on présente la classe de la demande de changement avec ces attributs. En fait, dans notre étude on s'intéresse juste à intégrer les deux premières phases du cycle de vie de la demande de changement qui sont : l'initialisation de la demande de changement, et la phase d'évaluation de la demande de changement. Concernant les deux dernières phases de la demande de changement (réalisation et validation), on présente leurs attributs au sein de la classe de la demande de changement, sans se focaliser sur eux.

Effectivement, la classe de la demande de changement RequestChange, doit contenir toutes les informations concernant l'application de la demande au sein du système de développement au cours de son cycle de vie, en passant par la phase d'initialisation jusqu'à celle de validation (cf. chapitre3). En effet, les sept premiers attributs de la classe RequestChange présentée ci-dessus correspondent à la phase d'initialisation de la demande de changement.

RequestChange::
request_Id : RequestChange_Id <-> replace_Request_Id
request_date : Request_Date <-> replace_Request_Date
kind : Change_Kind <-> replace_Kind
change_Type : Change_Type <-> replace_Change_Type
description : Description <-> replace_Description
rationale : Rationale <-> replace_Rationale
impact : Impact <-> replace_Impact
evaluation_date : Evaluation_Date <-> replace_Evaluation_Date
change_decision : Change_Decision <-> replace_Change_Decision
decision_Date : Decision_Date <-> replace_Decision_Date
request_application : Reuqest_Application <-> replace_Request_Application
application_date : Application_Date <-> replace_Application_Date
change_confirmation : Change_Confirmation <-> replace_Change_Confirmation
validation_date : Validation_Date <-> replace_Validation_Date
roles : Role_Id-set <-> replace_roles

Figure IV.11-les attributs de la demande de changement

Par exemple, les sept premiers attributs concernés par la phase d'initialisation de la demande de changement sont comme nous allons détailler par la suite: l'identificateur de la

demande de changement pour permettre aux utilisateurs de suivre son cycle de vie ; la date de la demande de changement ; le type de la demande de changement sur une exigence technique ou bien sur un produit final ou finalement sur un module ; la description de la demande de changement ; la justification de la demande de changement (pour savoir dans quel but les parties prenantes veulent appliquer le changement) ; et finalement l'étude d'impact. Dans la classe RequestChange, et précisément dans le septième attribut, on précise la possibilité d'appliquer plusieurs analyses d'impact : sur le coût, le délai, et sur la sécurité. Mais, ce qu'on essaye de présenter est juste l'analyse de l'impact sur la sécurité. En fait, le détail des sept premiers attributs, nous permet juste d'avoir une idée sur l'importance de cette phase, car elle aide les évaluateurs à analyser l'impact du changement.

3.8.1. Différents types de demande

Dans le paragraphe ci-dessus nous avons présenté la spécification de la classe RequestChange avec RSL. Précisément, le quatrième attribut de la classe RequestChange précise qu'on a différents types de demande de changement. Dans notre cas d'étude on présente trois types de demande de changement : changement d'une exigence technique, exigence d'un produit final, ou bien changement d'un module. Effectivement, d'autres types de demande de changement peuvent être ajoutés, comme la demande de changement d'un produit capacitant, la demande de changement d'une technologie capacitante, etc…

```
Change_Type = =
      mk_Change_TechRequirement(
        req_in_pid : Project_Id,
        req_in_bb : Building_Block_Id ,
        tr_id : TechRequirement_Id) |

      mk_Change_EndProduct(
        endp_in_pid : Project_Id,
        endp_in_bblock : Building_Block_Id,
       endp_id : EndProduct_Id) |

    mk_Change_BBblock(
         endp_in_pid : Project_Id,
        bblock : Building_Block_Id)
```

Figure IV.12-Les différents types de la demande de changement

Pratiquement, d'après la figure ci-dessus (cf. Figure IV.12), on peut remarquer que la demande de changement d'une exigence technique exige la connaissance l'identificateur du

projet, l'identificateur du module (Building Block), et l'identificateur de l'exigence technique. Par contre, dans certains outils d'ingénierie des exigences, les utilisateurs ont juste besoin de l'identificateur de l'exigence pour appliquer la demande de changement, ce qui n'est pas notre cas, car dans la spécification de notre outil on considère que deux différents modules (Building Block) peuvent avoir le même identificateur d'une exigence technique, mais ces deux exigences techniques doivent être différentes vu qu'elles sont dans deux modules différents. Ci-dessous, nous allons présenter les deux fonctions intéressantes qui doivent être utilisées tout au long du cycle de vie de la demande de changement.

3.8.2. Fonction d'initialisation

Les deux phases que nous avons spécifiées pour aider les développeurs tout au long du cycle de vie de la demande de changement pour la recherche d'impact sont : la phase d'initialisation, et la phase d'évaluation. Ci-dessous, on présente la fonction *initialization_process* qui est représentée au plus haut niveau de la spécification, et plus précisément au sein du module SYSTEM.rsl (cf. Figure IV.3).

```
/*P1:initialization process*/
  initialization_process :
    RequestChange_Id >< Sys -> Bool
  initialization_process(rc_id, system) is
    REQUESTCHANGES_.initialization_process(
      rc_id, requestchanges(system)),
```

Figure IV.13-La spécification formelle de la fonction initialisation de la demande

La fonction d'*initialization_process* (cf. Figure VI.13), ne représente pas la phase d'initialisation de la demande de changement, mais plutôt elle sert à tester si les parties prenantes ont complété toutes les informations nécessaires pour pouvoir évaluer l'impact de la demande. Ce qui veut dire explicitement, que les parties de la demande de changement ne peuvent pas continuer le cycle de vie de la demande de changement, si la sortie de la fonction d'*initialization_process* est fausse.

3.8.3. Fonction d'évaluation

Le deuxième processus que nous allons présenter est la phase d'évaluation. L'évaluation de la demande de changement doit être basée sur la notion de traçabilité pour analyser les parties impactées et concernées par une telle demande de changement. Egalement, la fonction d'*evaluation_process* est spécifiée formellement en RSL comme présentée ci-dessous (cf.

Figure VI.14). En effet, la recherche d'impact et le concept de traçabilité auront lieu juste en appelant d'autre fonctionnalité comme la fonction *safety-impact* spécifiée vers la fin de ce chapitre.

Effectivement, la sortie de la fonction *evaluation_process* est de type Impact. La fonction *evaluation_process* est spécifiée formellement en utilisant deux instructions If…then….else. En effet, au sein de la fonction *evaluation_process* il y aura un appel à la fonction *safety_impact* présentée dans le paragraphe ci-dessous pour connaître le lien avec la sécurité, et comme dans notre cas on n'indique pas l'impact de la demande de changement sur le coût ni sur le délai, la sortie de la fonction doit être No_Impact.

```
/*P2: evaluation process*/
    evaluation_process : RequestChange_Id >< Sys -> Impact
    evaluation_process(rc_id, system) is
     if
      safety_impact(rc_id, system) =
       No_Safety_related_to_System /\
      ~ delay_impact(rc_id, system) /\
      ~ cost_impact(rc_id, system)
     then No_Impact
     else
      if
       safety_impact(rc_id, system) =
         RC_Impact_on_Safety    /*RC:RequestChange*/
      then Safety
      else No_Impact
      end
     end,
```

Figure IV.14-La spécification formelle de la fonction evaluation_process

4. SPECIFICATION DE LA SECURITE

Dans les paragraphes ci-dessus, nous avons présenté la spécification en RSL de cinq classes avec leurs attributs. Par la suite, nous allons présenter les fonctionnalités concernées par la recherche d'impact sur la sécurité lors de la demande de changement. Par contre, la recherche d'impact sur la sécurité est divisée en plusieurs niveaux. Par la suite, nous allons présenter cinq différentes fonctions qui ont des sorties de types booléens, ces fonctions seront

la base de la recherche du lien entre la demande de changement soumise par les parties prenantes avec le système de sécurité.

4.1. Lien avec la sécurité

Plusieurs types d'exigences techniques peuvent être présents durant le développement des systèmes complexes. Dans la structuration de l'outil (cf. Figure VI.3), nous avons présenté la structuration de la spécification formelle des fonctionnalités de l'outil, mais nous n'avons pas présenté le concept de lien des exigences techniques à la sécurité.

```
tech_related_safety : TechRequirement -> Bool
    tech_related_safety(tr) is
      safety_builtin_techrequirement(tr),

safety_builtin_techrequirement :
      TechRequirement -> Bool
safety_builtin_techrequirement(tr) is
      ~ ((requirement_Type(tr) = Functional) \/
        (requirement_Type(tr) = Non_Functional)),
```

Figure IV.15-La fonction ci-dessus sert à tester si une exigence est liée à la sécurité. Cette fonction est spécifiée au niveau du module TechRequirement.rsl

Cependant, l'analyse du type des exigences techniques nous aide à savoir si telle exigence technique est liée à la sécurité ou non. Cette analyse, est présentée par l'appel de la fonction *safety_builtin_techrequirement*. Précisément, cette fonction a comme entrée un objet de type TechRequirement et comme sortie Bool. Afin, d'indiquer si une exigence technique est liée à la sécurité. Pourtant, la fonction *tech_related_safety* est comme la fonction *safety_builtin_techrequirement*, car à l'intérieure de la fonction *tech_related_safety* on fait appel à la function *safety_builtin_techrequirement*. La fonction *tech_related_safety* doit être spécifiée au niveau du module TechRequirement.rsl, et elle sera appelée au plus haut niveau de la spécification formelle de l'outil, au sein du module SYSTEM.rsl

Malheureusement, la spécification de la fonction *tech_related_safety* au niveau du module TECHREQUIREMENT.rsl n'est pas suffisante en tant que telle pour connaître le lien avec la sécurité, mais elle sera appelée au niveau du module SYSTEM.rsl par une autre fonction pour effectuer les tests demandés afin de savoir si une telle exigence technique est liée à la demande du changement et à la sécurité.

Dans la figure ci-dessous (cf. Figure VI.16), on peut remarquer que la signature de la fonction *safety_dependency_on_techrequirement* exige la connaissance de l'identificateur du projet, du module et de l'exigence technique, pour tester la dépendance avec la sécurité. Par contre, la spécification de la fonction est divisée en trois parties liées par le connecteur ou (\/). En effet, la première partie de la spécification sert à tester l'existence d'une autre exigence qui a comme identificateur tr1, et qui se trouve dans le champ de dépendance de l'exigence qu'on est entrain de tester sa dépendance avec la sécurité et qui a comme identificateur tr_id et que tr1 est liée à la sécurité. La deuxième partie de la fonction, signifie que l'exigence est liée directement à la sécurité, et la fonction *safety_dependency_on_techrequirement* signifie au moins l'existence d'une exigence fille qui sera liée à la sécurité.

```
safety_dependency_on_techrequirement:
      TechRequirement_Id >< Building_Block_Id >< Project_Id >< Sys ->
       Bool
safety_dependency_on_techrequirement (
      tr_id, bid, pid, system) is
      (exists tr1 : TechRequirement_Id :-
        tr1 isin
         TECHREQUIREMENT_.dependency(
                        get_techrequirement(tr_id, bid, pid, system)) /\
                  TECHREQUIREMENT_.tech_related_safety(
                    get_techrequirement(tr1, bid, pid, system))
            \/
                  TECHREQUIREMENT_.tech_related_safety(
                    get_techrequirement(tr_id, bid, pid, system))),
            \/
                    techreq_sons_related_to_safety(
                     tr_id, bid, pid, system)
```

Figure IV.16 -la spécification de la dépendance de la sécurité avec les exigences techniques au niveau du module System.rsl

```
techreq_sons_related_to_safety :
            TechRequirement_Id >< Building_Block_Id >< Project_Id >< Sys ->
                Bool
techreq_sons_related_to_safety(tr_id, bid, pid, system) is
                (exists id_son : TechRequirement_Id :-
                  id_son isin
                    dom techrequirements(bid, pid, system)
            /\
                  tr_id isin
                    get_parents_of_techreq(id_son, bid, pid, system)
            /\
                        TECHREQUIREMENT_.tech_related_safety(
                    get_techrequirement(tr_id, bid, pid, system))
```

Figure IV.17-Cette fonction a pour but de chercher si les descendantes d'une exigence sont liées à la sécurité.

Effectivement, la fonction *techreq_sons_related_to_safety* en RSL est divisée en trois parties (cf. Figure VI.17). Le but de cette fonction, est de chercher s'il existence au moins une exigence descendante liée à la sécurité. En effet, les trois parties de la fonction sont liées au connecteur and (/\).

Les trois fonctions présentées ci-dessus, concernent la présentation du lien entre les exigences techniques et la sécurité. Précisément, elles sont développées pour aider les parties prenantes à évaluer l'impact de la demande de changement sur la sécurité. Ainsi, les deux fonctions formelles présentées ci-dessous, ont le même but que les autres fonctions cités ci-dessus, afin d'aider les parties prenantes au cours du cycle de vie de la demande de changement. Mais, les deux fonctions ci-dessous seront utilisées dans un niveau beaucoup plus haut pour la recherche du lien de changement avec la sécurité.

```
buildingblock_related_safety :
        Building_Block_Id >< Project_Id >< Sys -> Bool
buildingblock_related_safety(bid, pid, system) is
        (exists endpro_id : EndProduct_Id :-
          endproduct_related_safety(
            endpro_id, bid, pid, system)),
```

Figure IV.18-cette fonction sert à vérifier si un module est lié à la sécurité.

```
system_related_safety: Sys -> Bool
      system_related_safety(system) is
        (exists pid : Project_Id, bid : Building_Block_Id:-
          buildingblock_related_safety (bid, pid, system)),
```

Figure IV.19-cette fonction sert à vérifier si un système durant/après son développement est lié à la sécurité.

4.2. Analyse d'impact

Dans ce paragraphe, nous allons présenter les trois fonctionnalités concernées par la recherche d'impact lors de la demande de changement. Au cours de ce travail de thèse, nous avons essayé de chercher juste l'impact sur la sécurité lors de la demande du changement. En fait, les deux fonctions présentées dans la figure ci-dessous (cf. Figure VI.20): *delay_impact* et *cost_impact* spécifiées avec RSL, renvoient toujours comme résultat : qu'il y a pas d'impact ni sur le délai ni sur le coût lors de la demande du changement, du point de vue recherche d'impact sur le coût ou le délai exigent des attributs particuliers, qui doivent être présents au sein du système de développement. Ces attributs permettent l'analyse d'impact sur le coût ou bien sur le délai, et on ne les a pas ajoutés car on effectue seulement une analyse d'impact du changement sur la sécurité.

```
/*Another Impact Analysis*/

    delay_impact : RequestChange_Id >< Sys -> Bool
    delay_impact(rc_id, system) is false,

    cost_impact : RequestChange_Id >< Sys -> Bool
    cost_impact(rc_id, system) is false,
```

Figure IV.20- La spécification formelle des deux fonctions delay_impact et cost_impact.

En fait, dans notre cas d'étude on considère qu'avec la demande de changement il y aura pas ni un impact sur la le coût ni un impact sur le délai (qui est quasiment impossible).

Ci-dessous (cf. Figure IV.21), nous allons présenter la spécification de la fonction utilisée pour rechercher l'impact sur la sécurité qui est *safety_impact*. Précisément, cette fonction concerne l'analyse de l'impact sur la sécurité. En fait, cette fonctionnalité doit être spécifiée au plus haut niveau au sein du module System.rsl.

```
/*safety impact*/
    safety_impact : RequestChange_Id >< Sys -> Message
    safety_impact(rc_id, system) is
      if ~ initialization_process(rc_id, system)
      then Initialization_Process_not_Finish
      else
        if ~ system_related_safety(system)
        then No_Safety_related_to_System
        else
          if safety_caused_by_RequestChange(rc_id, system)
          then RC_Impact_on_Safety
          else No_Safety_Impact_with_RC
          end
        end
      end,
```

Figure IV.21-spécification formelle de la fonction safety_impact

Particulièrement, la fonction *safety_impact* aura comme entrée l'identificateur de la demande de changement «*RequestChange_Id*», et le système «Sys» avec ses attributs. En effet, la sortie de cette fonction sera du type «Message» (cf. Annexe TYPES). Cette sortie joue le rôle d'émetteur de message pour l'évaluateur de la demande de changement, afin de permettre l'analyse de l'impact sur la sécurité.

5. IMPLEMENTATION DE L'OUTIL

5.1. Développement avec JAVA

Après l'explication de quelques fonctionnalités de la spécification de l'outil en RSL, on est passé vers l'implémentation de l'outil avec un langage de programmation. Parmi les langages proposés pour le développement des outils informatiques, on a choisi le langage de programmation Java. En fait, le but de Java à l'époque était de constituer un langage de programmation pouvant être intégré dans les appareils électroménagers, afin de pouvoir les contrôler, de les rendre interactifs, et surtout de permettre une communication entre les appareils. Etant donné que le langage C++ comportait trop de difficultés, un des acteurs du projet (considéré désormais comme le père de Java) décida de créer un langage orienté objet reprenant les caractéristiques principales du C++, en éliminant ses points difficiles, et en le rendant moins encombrant et plus portable.

5.1.1. L'aspect orienté objet

La première caractéristique du JAVA est l'orienté objet ("OO"). Ce caractère fait référence à une méthode de programmation et de conception du langage. Bien qu'il existe plusieurs interprétations de l'expression orientée objet, une idée phare dans ce type de développement est que les différents types de données doivent être directement associés avec les différentes opérations qu'on peut effectuer sur ces données. En conséquence, les données et le code sont combinés dans une même entité appelée objet. Un objet peut être vu comme une entité unique regroupant un comportement, le code, avec un certain état, les données. Le principe est de séparer les choses qui changent de celles qui ne changent pas; souvent un changement au niveau d'une structure de données va impliquer un changement dans le code servant à manipuler ces données et réciproquement. Ce découpage en entités cohérentes appelés objets permet d'avoir des fondations plus solides pour bâtir une architecture logicielle de qualité.

L'objectif est de pouvoir développer des projets plus simples à gérer et de réduire le nombre de projets aboutissant à un échec. Un autre objectif majeur de la programmation orientée objet est de développer des objets génériques de sorte que le code réutilisable entre différents projets. Un objet "client" générique par exemple doit avoir globalement le même comportement dans les différents projets, en particulier si ces différents projets se recoupent comme c'est souvent le cas dans les grandes organisations. Dans ce sens, un objet peut être vu comme un composant logiciel enfichable, permettant à l'industrie du logiciel de bâtir des projets d'envergure à partir d'éléments de base réutilisables et à la stabilité éprouvée tout en diminuant de manière drastique le temps de développement.

La réutilisation du code, lorsqu'elle est soumise à l'épreuve de la pratique, rencontre deux difficultés majeures: la création d'objets génériques réellement réutilisables est une notion très mal comprise et une méthodologie pour diffuser de l'information nécessaire à la réutilisation de code manque cruellement. Certaines communautés du monde "Open Source" veulent contribuer à résoudre ce problème en fournissant aux programmeurs la possibilité de diffuser largement de l'information sur les objets réutilisables et les bibliothèques objet.

5.1.2. Indépendance vis-à-vis de la plateforme

La deuxième caractéristique présentée en utilisant le langage de programmation JAVA, est l'indépendance vis à vis de la plateforme. Cette indépendance, signifie que les programmes écrits en Java fonctionnent de manière parfaitement similaire sur différentes architectures matérielles. On peut effectuer le développement sur une architecture donnée et faire tourner

l'application sur toutes les autres. Ce résultat est obtenu par les compilateurs Java qui compilent le code source "à moitié" afin d'obtenir un bytecode (plus précisément le bytecode Java, un langage machine spécifique à la plateforme Java). Le code est ensuite interprété sur une machine virtuelle Java (JVM en anglais), un programme écrit spécifiquement pour la machine cible qui interprète et exécute le bytecode Java.

De plus, des bibliothèques standard sont fournies pour pouvoir accéder à certains éléments de la machine hôte (le graphisme, le multithreading, la programmation réseau,...) exactement de la même manière sur toutes les architectures. Notons que même s'il y a explicitement une première phase précoce de compilation, le bytecode Java est interprété ou alors converti à la volée en code natif par un compilateur "juste à temps" (Just in time). Il existe également des compilateurs Java qui compilent directement le Java en code objet natif pour la machine cible, comme par exemple GCJ, supprimant la phase intermédiaire du bytecode mais le code final produit par ces compilateurs ne peut alors être exécuté que sur une seule architecture.

Les premières implémentations du langage utilisaient une machine virtuelle interprétée pour obtenir la portabilité. Ces implémentations produisaient des programmes qui s'exécutaient plus lentement que ceux écrits en C ou en C++, si bien que le langage souffrit d'une réputation de faibles performances. Des implémentations plus récentes de la machine virtuelle Java (JVM) produisent des programmes beaucoup plus rapides qu'auparavant, en utilisant différentes techniques.

5.1.3. Le Swing

Le Swing constitue une innovation apparue dans JDK1.1 en tant qu'extension de ce dernier et en tant que partie intégrante de Java 2 SDK. Dans notre outil, le Swing était une facilité pour le développement de notre interface. Swing est la seconde bibliothèque de classes (la première étant AWT) permettant de créer et gérer des interfaces graphiques. Les méthodes utilisées pour construire une interface Swing sont sensiblement les même que celles de AWT soit :

- créer un cadre ou contenant,
- placer des composants dans ce contenant,
- effectuer la mise en page de ces composants dans le contenant,
- gérer les événements et actions posées par l'utilisateur.

5.2. L'environnement Eclipse

Eclipse est un environnement de développement intégré dont le but, est de fournir une plate-forme modulaire pour permettre de réaliser des développements informatiques. En effet, I.B.M. est à l'origine du développement d'Eclipse qui est d'ailleurs toujours le cœur de son outil Websphere Studio Workbench (WSW), lui même à la base de la famille des derniers outils de développement en Java d'I.B.M. Tout le code d'Eclipse a été donné à la communauté par I.B.M afin de poursuivre son développement.

Cependant, les principaux modules fournis en standard avec Eclipse concernent Java mais des modules sont en cours de développement pour d'autres langages notamment C++, Cobol, mais aussi pour d'autres aspects du développement (base de données, conception avec UML, ...). Ils sont tous développés en Java soit par le projet Eclipse soit par des tiers commerciaux ou en open source. Les modules agissent sur des fichiers qui sont inclus dans l'espace de travail (Workspace). L'espace de travail regroupe les projets qui contiennent une arborescence de fichiers.

Bien que développé en Java, les performances à l'exécution d'Eclipse sont très bonnes car il n'utilise pas Swing pour l'interface homme-machine mais un toolkit particulier nommé SWT associé à la bibliothèque JFace. SWT (Standard Widget Toolkit) est développé en Java par IBM en utilisant au maximum les composants natifs fournis par le système d'exploitation sous-jacent. JFace utilise SWT et propose une API pour faciliter le développement d'interfaces graphiques.

Eclipse ne peut donc fonctionner que sur les plate-formes pour lesquelles SWT a été porté. Ainsi, Eclipse 1.0 fonctionne uniquement sur les plate-formes Windows 98 /NT /2000 /XP et Linux. En fait, SWT et JFace sont utilisés par Eclipse pour développer le plan de travail (Workbench) qui organise la structure de la plate-forme et les interactions entre les outils et l'utilisateur. Cette structure repose sur trois concepts : la perspective, la vue et l'éditeur. La perspective regroupe des vues et des éditeurs pour offrir une vision particulière des développements. En standard, Eclipse propose huit perspectives. Les vues permettent de visualiser et de sélectionner des éléments. Les éditeurs permettent de visualiser et de modifier le contenu d'un élément de l'espace de travail.

Enfin, le plugin ECLIPSE VE (Visual Editor) est un framework dont le but est de faciliter le développement permettant la réalisation d'interfaces graphiques. Même si VE est proposé avec une implémentation de référence proposant la conception d'interfaces graphiques

reposant sur SWT, Swing ou AWT, le but du framework est de pouvoir proposer à terme la conception d'interfaces graphiques reposant sur d'autres bibliothèques, pas nécessairement développées en ou pour Java.

6. ETUDE DE CAS

6.1. Présentation

Dans le troisième chapitre nous avons montré que chaque demande de changement doit avoir un cycle de vie composé de quatre phases différentes : la phase d'initialisation, la phase d'évaluation, la phase de réalisation et celle de validation. Au cours de la thèse nous avons traité trois exemples (feux de croisement, la chaudière et l'exemple du port). Par la suite, nous allons présenter l'exemple de feux de croisement avec sa décomposition selon l'EIA-632, ensuite le cas d'après l'évolution.

6.2. Feux de croisement (S1)

Le réseau routier, est l'ensemble des voies de circulation terrestres permettant le transport par véhicules routiers. L'utilisation du réseau routier est toujours liée à la prévention routière. La prévention routière est utilisée pour assurer la sécurité des utilisateurs de son réseau. En fait, la prévention routière [Site 3] est défini par: l'ensemble des mesures visant à éviter les accidents de la circulation (prévention du risque), ou à atténuer leurs conséquences (prévision).

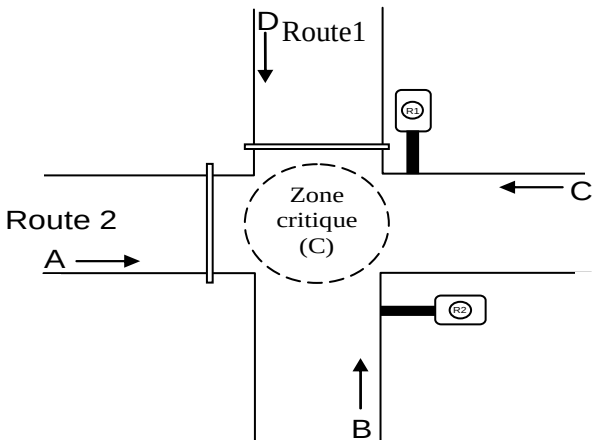

Figure IV.22- la zone critique de (S1), est piloté selon la présence des voitures ; avec une priorité pour la Route1.

Dans la figure ci-dessus (cf. figure IV.22), on présente l'exemple des feux de croisement (S1). Pour cet exemple, on considère que la zone critique (C) qui se trouve sur le croisement des deux routes, est gérée en fonction de la présence des voitures ; avec une priorité pour la Route1 tout en supposant que sur chaque route on a un feu permettant l'utilisation de la zone critique(C) ou non.

Enfin, la présentation de cet exemple sera divisée en deux parties : la première partie sera la décomposition de l'exemple selon l'EIA-632. La deuxième partie sera pour la présentation du système de sécurité.

6.2.1. La décomposition selon l'EIA-632

Dans la figure ci-dessous, nous décomposons le système des feux de croisement, selon la structuration présenté au sein de l'EIA-632. En fait, durant la décomposition de cet exemple, on a proposé une décomposition de 3 niveaux (resp. dans l'exemple présenté dans le deuxième chapitre on a présenté un exemple avec cinq niveaux de décomposition). Le premier niveau (1) présenté dans la figure, contient le module : « feux de croisement ». Ce niveau contient la racine de la hiérarchie. À ce niveau, on aura les buts du client pour le développement du système désiré. Pratiquement, pour cet exemple, on propose une décomposition du système selon trois projets (Projet A, Projet B, Projet C) contenant sept modules (cf. figure V.23) :

- Feux de croisement (niveau1)
    - Infrastructure (niveau2)
        - Route (niveau3)
        - Lampes (niveau3)
        - Feux des routes (niveau3)
        - Capteurs (niveau3)
    - Contrôle (niveau2)

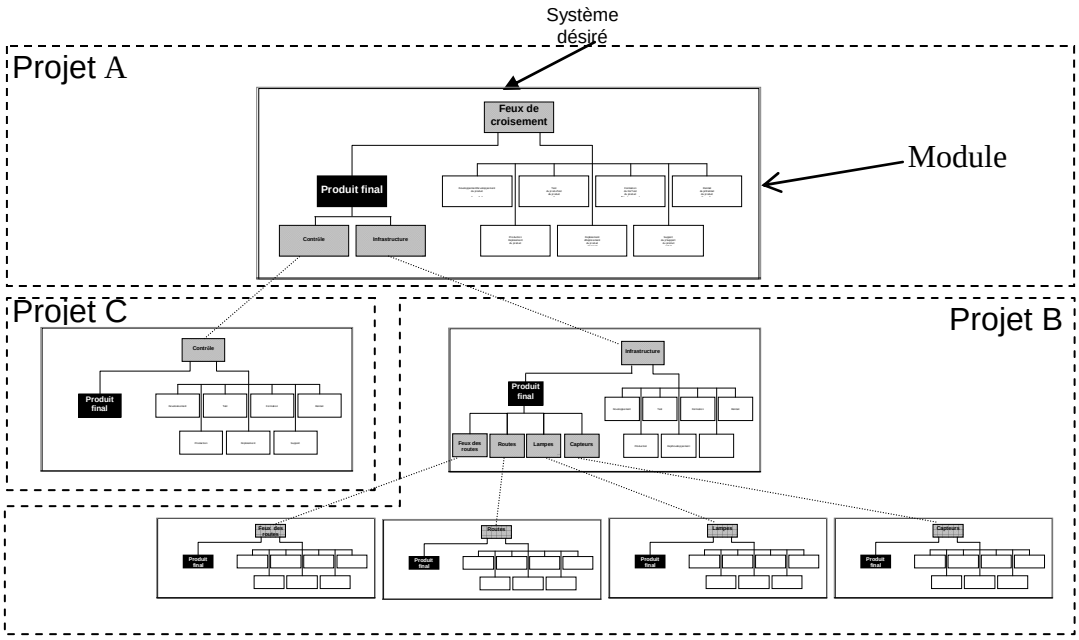

Figure IV.23- On décompose le développement du système des feux de croisement avec sept modules.

D'après la figure *IV.2.4*, on remarque que le développement du produit final du module de plus haut niveau (présenté dans le projet A) sera divisé en deux sous-systèmes: Infrastructure, et Contrôle. Les deux nouveaux sous-systèmes seront développés dans un autre niveau, qui sera le deuxième niveau dans notre décomposition. En fait, le développement du module Contrôle (présent dans le Projet C) n'exige pas une décomposition avec un nouvel ensemble des sous-systèmes, vu que son développement est possible à ce niveau. Par contre, le module Infrastructure (Projet B) exige une décomposition en 4 sous-systèmes. Cette décomposition fournit un troisième niveau contenant quatre modules : Route, Lampes, Feux de routes, Contrôle.

Enfin, le développement de chaque produit final présenté dans chaque module doit avoir : un ensemble d'exigences techniques, une solution logique, une solution physique et une conception. En fait, la solution logique nous permet de valider les contraintes de sécurité pour les modules liés à la sécurité, tout en utilisant le langage formel RSL.

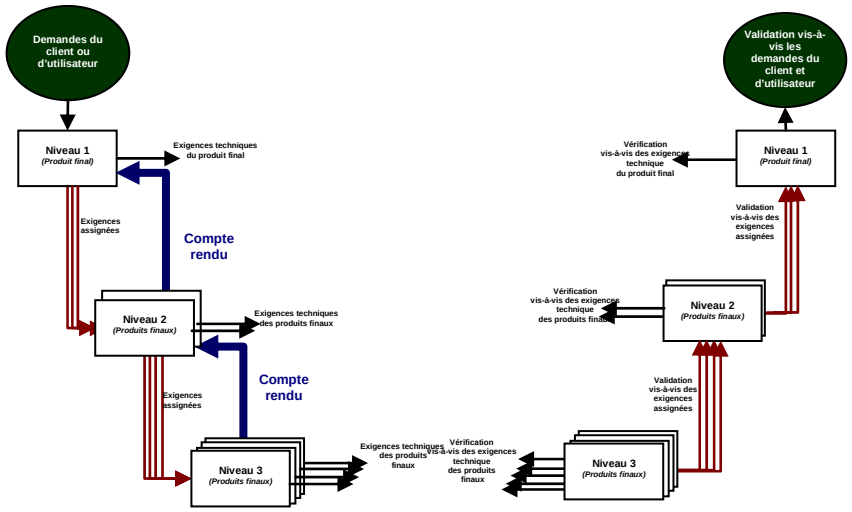

Figure IV.24- développement descendant et réalisation ascendante des produits finaux pour l'exemple de feux de croisement (S1).

6.2.2. Le système de sécurité

Durant le développement du système (S1) selon le cadre de l'EIA-632 nous constatons que le système de sécurité contient deux types d'exigences sécuritaires : les exigences significatives de sécurité et les contraintes de sécurité. En fait, les exigences significatives de sécurité existent au niveau du module «Feux de croisement » et du module « Capteurs ». Par contre, les contraintes de sécurité sont présentées au niveau du module «feux de route » et du module « Contrôle». Par la suite, nous allons présenter la décomposition du système de sécurité au niveau des différents modules :

- Parmi les exigences techniques du système de sécurité pour le module feux de croisement présenté au niveau 1, on trouve la contrainte de sécurité CS1 (le système S1, doit contrôler la zone critique entre les deux routes. Il ne faut pas que la zone critique soit accessible par des véhicules venant des deux routes).
- Parmi les exigences techniques du système de sécurité pour le module Contrôle présenté au niveau2, on trouve la contrainte de sécurité CS2 (Il ne faut pas que les deux lampes qui contrôlent la zone critique soient allumées simultanément).
- Parmi les exigences techniques du système de sécurité pour le module feux de routes présenté au niveau3, on trouve l'exigence significative de sécurité ESS1 (il faut toujours maintenir les feux de routes, car dans le cas d'une telle défaillance la gravité du problème de la sécurité qu'on peut avoir est sévère).

- Finalement, parmi les exigences techniques du système de sécurité pour le module Capteurs au niveau3, on trouve l'exigence significative de sécurité ESS2 (il faut toujours maintenir les feux de routes, car dans le cas d'une telle défaillance la gravité du problème de la sécurité qu'on peut avoir est catastrophique).

Après cette décomposition du système de sécurité nous obtiendrons quatre modules liés à la sécurité:

- Feux de croisement (niveau1) ->***Building-Block lié à la sécurité***
  - Infrastructure (niveau2)
    - Route (niveau3)
    - Lampes (niveau3)
    - Feux de routes (niveau3) ->***Building-Block lié à la sécurité***
    - Capteurs (niveau3) ->***Building-Block lié à la sécurité***
  - Contrôle (niveau2) ->Building-Block lié à la sécurité

6.3. Feux de croisement après évolution (S2)

Dans la figure ci-dessous, on présente le cas d'une évolution d'un ancien système (tout en respectant les exigences de sécurité), afin d'obtenir un nouveau système avec des nouvelles fonctionnalités. Dans cette figure, on considère que toutes les exigences sécuritaires présentées au niveau du système de sécurité doivent être satisfaites avant et après l'évolution, c.à.d. il faut que les exigences sécuritaires soient satisfaites au niveau de l'ancien système et au niveau du nouveau système.

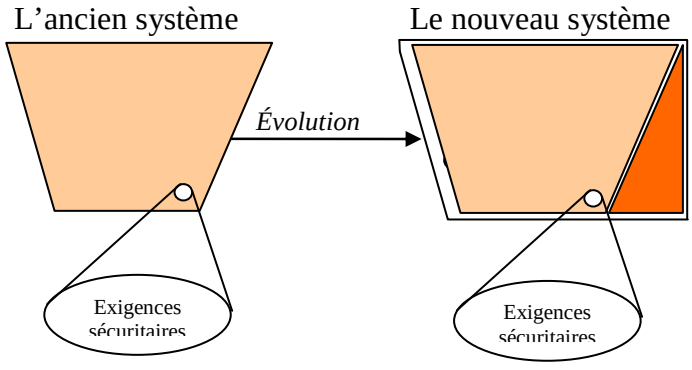

Figure IV.25-Les exigences sécuritaires doivent être respectées avant et après l'évolution.

Dans la figure ci-dessous, on présente l'exemple de feux de croisement (S2) mais après l'évolution du système (S1) présenté dans la figure (cf. figure IV.22). Le système S2 doit toujours conserver les exigences sécuritaires présentées au niveau du système de sécurité. Cette évolution nous oblige à avoir un cycle de vie pour la demande de changement ; afin de l'initialiser, l'évaluer, la réaliser et la valider.

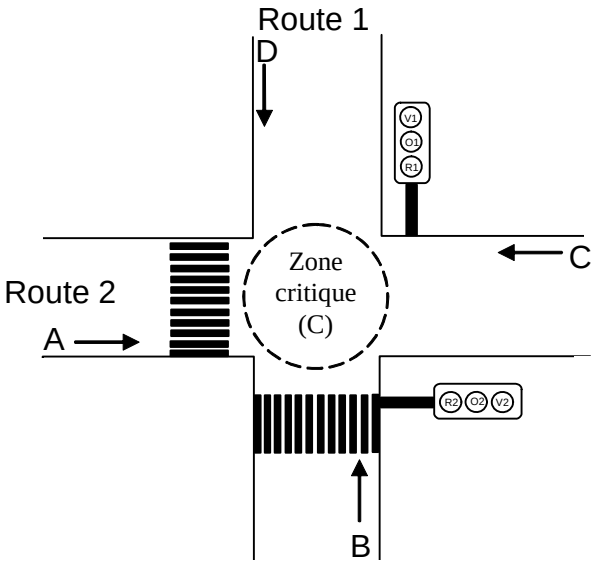

Figure IV.26- la zone critique de (S2), est pilotée par une gestion du passage.

6.3.1. Le cycle de vie de la demande

Le cycle de vie pour une demande de changement est composé de quatre phases. Mais, dans notre étude on se limite aux deux premières phases. Par la suite, nous allons présenter les deux premières phases : l'initialisation de la demande, l'évaluation de la demande.

A) Phase d'initialisation

Le processus d'identification de la demande est le premier processus de cette phase. Le but de ce processus est de connaitre le type de la demande de changement, le numéro de la demande de changement, la phase de la demande de changement, la description et la justification de la demande. Dans ce paragraphe, nous n'allons pas détailler tous les attributs d'une demande de changement, mais nous allons présenter la phase de la demande de changement et la justification.

Type de la demande : dans cet exemple on considère que la demande apparaît après l'implémentation du système (DCI : demande de changement sur une implémentation).

Justification : Il faut intégrer un passage piéton au niveau de S1.

Enfin, après l'exécution de ce processus d'identification de la demande. On exécute le processus de la soumission de la demande, afin d'intégrer la demande au niveau du système et d'évaluer son impact sur la sécurité.

B) Phase d'évaluation

La phase d'évaluation de la demande est composée de quatre processus différents : processus de traçabilité, le processus d'évaluation, le processus de classification de la demande et le processus d'approbation. Par la suite, nous allons expliquer les différentes activités présentées au niveau de chaque processus.

B.1) Processus de traçabilité

Le but de ces processus est de chercher la trace de la demande de changement. En effet, les activités au niveau de ce processus concernent la recherche de tous les constituants du système, impactés par la demande de changement. L'utilisation nous a permit de savoir l'ensemble des modules impactés par la demande de changement, et particulièrement les exigences techniques d'un tel module.

B.2) Processus d'évaluation de la sécurité

Après l'exécution du processus de traçabilité et la recherche de la trace de la demande de changement. L'évaluation de la demande doit être guidée par deux types d'analyses par une analyse de la trace au niveau du système de sécurité, et une analyse d'impact formelle.

Parmi les modules impactés par la demande de changement sur S1, on peut citer le module « feux de route ». En fait, ce module est un module lié à la sécurité. Dans ce cas, l'application de la demande de changement peut affecter un problème de sécurité, vu que la trace de la demande affecte le système de sécurité. D'où la nécessité d'exécuter une analyse d'impact formelle.

La deuxième analyse d'impact sur la sécurité nous oblige à passer vers la spécification formelle faite avec RSL, afin de valider les contraintes de sécurité. En fait, la contrainte de sécurité (CS1) présentée au niveau du module « feux de croisement » est satisfaite, si la contrainte de sécurité (CS2) présentée au niveau du module « Contrôle » est satisfaite. Alors, il faut qu'on passe vers la solution logique (spécification formelle avec RSL) présentée au niveau du module « Contrôle », afin d'établir l'analyse d'impact formelle en intégrant la

modification au niveau de la spécification. En effet, avant l'approbation de la demande de changement la spécification reste une version provisoire

Enfin, notre outil supporte le premier type d'analyse ; car le deuxième type d'analyse (analyse formelle) doit être effectué en interfaçant avec l'outil qui supporte le langage RSL.

6.4. Outillage

Dans la figure ci-dessous (cf. figure IV.27), on présente l'interface de notre outil implémenté et programmé avec JAVA. En fait, notre interface est divisée en trois fenêtres : dans la première fenêtre on présente les champs des objets (dans la figure ci-dessous, on présente le format d'une demande de changement) ; la deuxième fenêtre est utilisée pour envoyer des messages, et finalement une fenêtre pour la gestion. La décomposition de l'exemple de feux de croisement est affichée dans la partie gestion. Dans cette fenêtre, on peut remarquer la décomposition de l'exemple avec trois projets.

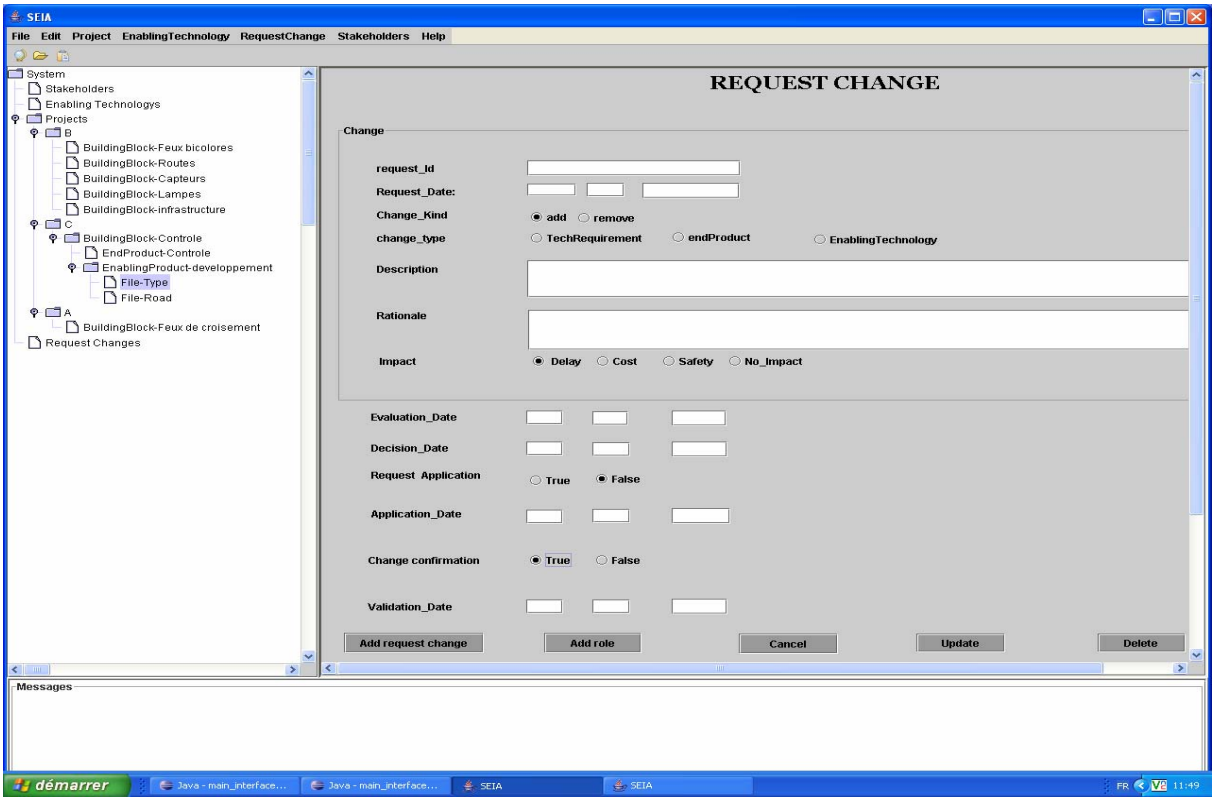

Figure IV.27- L'interface de notre outil

Enfin, dans le cas où il y aura une soumission d'une demande de changement, l'outil permet à l'utilisateur de connaitre l'impact de l'évolution sur la sécurité en cherchant la trace de la modification. Cette analyse sera établie : en cherchant le lien entre la demande et le

système de sécurité et en envoyant le résultat de la recherche d'impact dans la fenêtre des Messages.

7. CONCLUSION

Dans ce chapitre, nous avons présenté le modèle de la recherche d'impact lors de la demande de changement. En fait, la description du modèle était basée sur une spécification en UML, puis nous avons présenté la spécification formelle de modèle en RSL, dans le but de spécifier toutes les fonctionnalités nécessaires pour la recherche d'impact et l'élimination de l'ambiguïté présenté au niveau de la spécification avec UML.

Effectivement, la spécification du modèle en RSL est formée par plus de 8400 lignes entre spécification détaillée et le test de cas «*test_case*». Pratiquement, la conception détaillée de l'ensemble des fonctionnalités à pour but de faciliter le passage vers un langage de programmation comme le langage Java orienté objet. En fait, les tests de cas supportés par le langage de programmation RSL ont pour but d'appliquer un ensemble des tests sur la spécification formelle de l'outil. Le but des tests de cas est de découvrir les erreurs le plutôt possible, avant l'implémentation de l'outil avec un tel langage de programmation. Par contre, dans ce chapitre nous n'avons pas présenté comment peut on appliquer un ensemble de test de cas avec RSL en générant des fichiers SML [BUR-99].

CONCLUSION GENERALE

Plusieurs compagnies utilisent leurs propres méthodes pour analyser l'impact d'évolution des systèmes complexes, d'autres appliquent des méthodes développées dans différents domaines : militaire, électronique ou le génie logiciel. Dans ce manuscrit nous avons présenté une nouvelle méthode pour l'analyse d'impact du changement sur la sécurité. Le but de cette méthode est d'illustrer la manière d'analyser l'impact d'évolution du système sur la sécurité, en considérant que son développement est dans le contexte de l'ingénierie systèmes et de l'ingénierie des exigences.

Dans les deux premiers chapitres nous avons montré les effets de la volatilité des exigences, le contexte de l'étude, les trois différentes approches développées pour la recherche d'impact, les avantages de la traçabilité et les méthodes formelles. En effet, l'analyse d'impact sur la sécurité nous a imposé l'utilisation des méthodes formelles, afin de vérifier la non volatilité des contraintes de sécurité avant et après l'application des évolutions.

Dans les deux derniers chapitres nous avons montré notre méthodologie concernée par : l'approche générale de l'étude, la décomposition du système de sécurité, le cycle de vie d'une évolution, et finalement la spécification formelle de l'outil qui nous aide à analyser l'impact d'évolution sur la sécurité. En effet, dans notre étude nous avons distingué plusieurs types d'évolution: évolution d'une exigence, évolution d'une conception et évolution d'une implémentation. Cependant, au fil de ce manuscrit nous nous sommes focalisé sur l'analyse d'impact de l'évolution d'une exigence ou d'une conception ; beaucoup plus que l'analyse de l'évolution d'une implémentation, vu que son évolution dépend du type du système (mécanique, logiciel, etc…).

Dans notre méthodologie, l'analyse d'impact a été divisée en deux types d'analyses : le premier type d'analyse est l'analyse préliminaire basée sur la recherche de la trace de l'évolution au niveau du système de sécurité. Le deuxième type d'analyse est l'analyse formelle d'impact basée sur la validation des contraintes de sécurité spécifié formellement avec les fonctionnalités du système en RSL.

Notamment, cette étude n'est pas suffisante pour confirmer qu'il n y aura jamais d'impact non désiré sur la sécurité lors de l'évolution, que lorsque le changement affecte une contrainte de sécurité. Car, nous n'avons pas précisé des méthodes à appliquer lorsque le changement affecte une exigence pure de sécurité ou bien une exigence significative de sécurité.

Comme perspective, nous voulons intégrer notre démarche dans un atelier d'ingénierie système permettant d'autoriser l'évolution des exigences, en préservant leurs propriétés comme celle de la sécurité; deuxièmement l'atelier doit disposer d'un puissant modèle de traçabilité entre exigence, conception et implémentation qui nous permet à son tour de faire évoluer le système sans affecter la sécurité.

BIBLIOGRAPHIES

[A]

[ADD-00] R. J. Adduci; W. T. Hathaway, and L. J. Meadow: *"Hazard Analysis Guildlines For Transit Projects"*. U.S. Department of Transportation Research and Special Programs Cambridge, MA 02142-1093, January 2000.

[AND-98] S. Andriole: *"The politics of requirements management"*. IEEE Software, pages 82-84, November/December 1998.

[ARN-93] R. S. Arnold; S. A. Bohner: "*Impact analysis–Towards a framework for comparison*". Proceedings of the International Conference on Software Maintenance, pages 292-301, 1993.

[ARN-96] R. S. Arnold and S. A. Bohner: *"An Introduction to Software Change Impact Analysis"*. IEEE Computer Society Press, 1996.

[B]

[BER-98] Daniel M. Berry, "The Safety Requirements Engineering Dilemma" 9th International Workshop on Software Specification & Design, p. 147, 1998.

[BAC-99] J. Bach:*"Risk and requirements-based testing"*. Computer, pages 113-114, June 1999.

[BOH-91] S. A. Bohner: *" Software change impact analysis for design evolution"*. Proceedings of the International Conference on Software Maintenance and Reengineering, 8:292-301, 1991.

[BOH-96a] S. A. Bohner; R. S. Arnold: *"Software Change Impact Analysis"*. IEEE Computer Society Press, Los Alamitos, 1996.

[BUR-98] J.V Buren; D. cook: *"Experiences in the Adoption of Requirements Engineering Technologies"*, Journal of Defence Software Engineering, December, pp.3-10, 1998.

[BUR-99] I. Burdonov; A.; Kossatchev; A.; Petrenko and D. Galter: *"Automated Generation of Test Suites from Formal Specifications"*. Proceedings of FM'99 - Formal Methods, *volume 1708 of* Lecture Notes in Computer Science, Springer-Verlag, 1999.

[BOH-02] S. A. Bohner: *"Software change impacts - an evolving perspective"*. Proceedings of the International Conference on Software Maintenance, pages 263-272, October 2002.

[C]

[CAR-04] R.S.Carson; E. Aslaksen; G. Caple; P. Davies; R. Gonzales; R.Kohl; A.E.K. Sahraoui: *"Requirements completeness"*.14th Annual International Symposium INCOSE 2004, 15p, Toulouse (France), 20-25 June 2004.

[COU-05a] C.Coulin; A.E.K.Sahraou: *"A meta-model based guided approach to collaborative requirements elicitation for software systems development"*, 18th International Conference on Software & Systems Engineering and their Applications (ICSSEA'2005), 6p, Paris (France), 29 November – 1 December 2005.

[COU-05b] C.Coulin; D.ZowghiI; and A.E.K. Sahraoui: *"A lightweight workshop-centric situational approach for the early stages of requirements elicitation in software systems development"*, 13th IEEE International Requirements Engineering Conference (RE'2005), 10p, Paris (France), 29 Août - 2 September 2005.

[COU-05c] C.Coulin; D.ZowghiI; and A.E.K. Sahraoui:*"Towards a collaborative and combinational approach to requirements elicitation within systems engineering framework "*,18th International Conference on Systems Engineering (ICEng'05), pp.456-461, Las Vegas (USA), 16-18 Août 2005.

[CYS-03] L.M. Cysneiros; E. Yu: *"Non-Functional Requirements Elicitation,"* Perspective in Software Requirements, Kluwer Academics, 2003.

[D]

[DUL-04] N. Dulac; N. Leveson : *"An Approach to Design for Safety in Complex Systems"* International Conference on System Engineering (INCOSE '04), Toulouse, June 2004.

[E]

[ELJ-04a] M.H.Eljamal :*"Requirements evolution and impacts on safety"*, IFIP 18th World Computer Congress, pages 123-132, Student Forum, Toulouse (France), August 2004.

[ELJ-04b] M.H.Eljamal : "Problématique du changement d'exigences et impact sur la sécurité", Groupement d'Echange Thématique Sûreté de Fonctionnement (GET SDF). Atelier Sûreté de Fonctionnement et Maîtrise des Risques, 13 Mai 2004.

[ELJ-05a] M.H.Eljamal : "*Dynamic design and invariants properties*". 3rd Annual Conference on Systems Engineering Research (CSER'2005), New Jersey (USA), 23-25 Mars 2005.

[ELJ-05b] M.H.Eljamal: "*Impact de l'évolution des exigences sur la sécurité* ". 6ème Congrès des Doctorats de l'Ecole Doctorat Systèmes (EDSYS), Toulouse (France), 19-20 Mai 2005.

[ELJ-05c] M.H.Eljamal; A.E.K.Sahraoui : "*Towards a methodology for requirements evolution and impact on safety requirements* ", 6ème Congrès International Pluridisciplinaire "Qualité et Sûreté de Fonctionnement" (QUALITA 2005), pages.117-124, Bordeaux (France), 16-18 Mars 2005.

[ELJ-05d] M.H.Eljamal; A.E.K.Sahraoui: "*Customizing systems engineering concepts: case study on concurrent engineering*", 12th Annual European Concurrent Engineering Conference (ECEC'2005), pages.57-62, Toulouse (France), 11-13 Avril 2005.

[G]

[GEO-98] C. George; D. T. Dung: "*Combining and Distributing Hierarchical Systems* ". *R*equirements Targeting Software and Systems Engineering, LNCS 1526, 1998.

[GEO-99] C. George, D. Bjoerner and S. Prehn: "*Scheduling and Rescheduling of Trains* ". In Industrial-Strength Formal Methods in Practice, Springer-Verlag, 1999.

[GHA-02] A. Ghariani, A.K.A Toguyeni.; P. Craye: " A functional graph approach for alarm filtering and fault recovery for automated production systems". Discrete Event Systems. Proceedings. Sixth International Workshop on 2-4 Oct. 2002 Page(s):289 – 294, 2002.

[GHA-04] A. Ghariani, A.K.A Toguyeni.; P. Craye : "Towards an approach to automate reconfiguration procedure in automated production systems". Systems, Man and Cybernetics, 2004 IEEE International Conference, Volume 5, 10-13 Oct. Page(s):4272 - 4277 vol.5, 2004.

[GIB-04] Evolutionary Project Management & Product Development, Kai Gilb, 2004.

[GIO-05] P. Giorgini; F. Massacci; J. Mylopoulos; and N. Zannone : *"ST-TOOL: A Case Tool For Security Requirements Engineering"*. RE-2005, p451-452, La Sorbonne-Paris, 2005.

[GOT-94] O. Gotel ; A. Finkelstein: *"An analysis of the requirements traceability problem"*. Proceedings of the First International Conference on Requirements Engineering, pages 94-101, 1994.

[H]

[HAX-00] A.E. Haxthausen; J. Peleska: *"Formal Development and Verification of a Distributed Railway Control System"*. IEEE Transactions on Software Engineering, Volume 26, Issue 8 (August 2000) pp. 687 - 7012000.

[HEL-01] L. Hellouin; J.L.Beaugrand; and A.E.K. Sahraoui: *"Requirements process and traceability issues"*. *11th Annual INCOSE Symposium*, Melbourne, 2001.

[HEL-02] L. Hellouin, 2002 : *"Contribution à l'ingénierie des exigences et à la traçabilité"*. PhD Thesis, LAAS-CNRS and INPT, 2002.

[J]

[JAR-98] M. Jarke: *"Requirements tracing"*. Communications of the ACM, 41(12):32—36, December 1998.

[JAC-99] I. Jacobson; G. Booch; and J. Rumbaugh: *"The Unified Software, Development Process"*. Addison-Wesley Longman, 1999.

[JON-03] E. Jonas: *"Analysis of Intent Specification and System Upgrade Traceability"*. Master's thesis, openings Universitet, Sweeden, December 2003.

[JUR-02] N. Juristo; A. M. Moreno; and A. Silva.: *"Is the European industry moving toward solving requirements engineering problems? "*. IEEE Software, pp. 70-77, November/December, 2002.

[K]

[KOT-98]G. Kotonya; I. Somerville: *"Requirements Engineering: Processes and Techniques"*. John Wiley & Sons Ltd, Chichester, West Sussex, England, 1998.

[KRA-89] H. Krasner: *"Requirements Dynamics in Large Software Projects"*. 11th World Computer Congress, IFIP89, Amsterdam, the Netherlands, 1989.

[L]

[LAN-98] M. Lane; A.L.M. Cavaye: *"Management of Requirements Volatility Enhances Software Development Productivity"*. 3th Australian Conference on Requirements Engineering (ACRE 98), Geelong, Australia,1998.

[LEV-00] N. Leveson: *"Intent Specifications: An Approach to Building Human-Centered Specifications"***.** IEEE Trans. on Software Engineering, January 2000.

[LEV-03a] N. Leveson : *"A New Approach to Hazard Analysis for Complex Systems"*. International Conference of the System Safety Society, Ottawa, August 2003.

[LEV-03b] N. Leveson; M. Daouk; N. Dulac, and K. Marais: *"Applying STAMP in Accident Analysis "*. Workshop on Investigation and Reporting of Incidents and Accidents (IRIA), September 2003.

[LEV-04] N. Leveson; J. Cutcher-Gershenfeld: *"What System Safety Engineering can Learn from the Columbia Accident"*. International Conference of the System Safety Society, Providence Rhode Island, August 2004.

[LEV-05] N. Leveson**:** *"A Systems-Theoretic Approach to Safety in Software-Intensive Systems"*. IEEE Trans. on Dependable and Secure Computing, January 2005.

[M]

[MAL-98] Y. Malayia; J. Denton: *"Requirements Volatility and Defect Density"*. *The 10th International Symposuim on Software Reliability Engineering*, Fort Collins, 1998.

[MAC-01] L. A. Maciaszek: *"Requirements Analysis and System Design*".Person Education Limited, Harlow, England, 2001.

[MES-05] M.Messadia, M.H.Eljamal, A.E.K.Sahraoui: *"Systems engineering processes deployment for PLM"*. International Conference on Product Lifecycle Management (PLM'05), pp.282-291, Lyon, France, 11-13 July 2005.

[MES-06] M.Messadia, M. Djeghaba , A.E.K.Sahraoui: *"System engineering framework for manufacturing systems "*. International Conference on Control, Modelling and Diagnosis (ICCMD'06), Annaba (Algérie), 22 - 24 May 2006.

[N]

[NAV-01a] I. Navarro; N. Leveson; K. Lundqvist: *"Reducing the Effects of Requirements Changes through System Design"*. MIT University, 2001.

[NAV-01b] I. Navarro; K. Lundqvist; and N.Leveson: *"An Intent Specification Model for a Robotic Software Control System"*. DASC '01, 2001.

[NUR-04] N. Nurmuliani; D. Zowghi; and S. Fowell.: *"Analysis of Requirements Volatility during Software Development Life Cycle"* .Proceedings of the Australian Software Engineering Conference (ASWEC), April 13-16, Melbourne, Australia,2004.

[R]

[RAM-95] B.Ramesh, C.Stubbs, T.Powers, M.Edwards: "*Lessons Learned from implementing Requirements Traceability "*. Crosstalk Journal of Defense Engineering8, 4 April 1995, pp11-15, 1995.

[ROB-04] S. Robertson; J. Robertson: *" Requirements-Led Project Management: Discovering David's Slingshot "***.** Wesley, 2004.

[RUS-04] Michael S. Russell: *" Assessing the impact of requirements volatility on the SE Process using Bayesian"*. INCOSE, Toulouse, 2004.

[S]

[SAH-01] A.E.K.Sahraoui: "*Experience in requirements elicitation with formal and semi-formal methods"*. Rapport LAAS N°01348, Juillet 2001.

[SEK-02] L. Sekhri; A.K.A Toguyeni.; P. Craye: "A relational based approach for analysing functional graphs of automated production systems". Systems, Man and Cybernetics, 2002 IEEE International Conference, Volume 1, Page(s):55 – 59,6-9 Oct. 2002.

[SAH-02a] A.E.K.Sahraoui: "Requirement elicitation and specification of temporal properties: text, logics, automata and Java assertions". Rapport LAAS N°02454, 13p, Octobre 2002.

[SAH-02b] A.E.K.Sahraoui: "*Requirements engineering: the automata based elicitation and related issues "*. International Conference on Software Engineering Research and Practice (SERP'02), pp.108-114, Las Vegas (USA), 24-27 Juin 2002.

[SAH-04] A.E.K.Sahraoui; D.M.Buede; and A.P.Sage: *"Issues for systems engineering research"*.14th Annual International Symposium INCOSE 2004, Toulouse (France), 11p, 20-25 June 2004.

[SAH-05] A.E.K.Sahraoui: *"Requirements traceability issues methodology and formal basis"*, International Journal of Information Technology & Decision Making (ITDM), Vol.4, N°1, pp.59-80, Mars 2005.

[SCH-91] J. J. Schwarz; J. J. Skubich; R. Aubry : "Lacatre: The basis for a Real Time Software Engineering Workshop". Informatik-Fachberichte; Vol. 295, Workshop über Realzeitsysteme, 12. Fachtagung des PEARL-Vereins e.V. ,Pages: 20 - 40, 1991, ISBN:3-540-54909-9, Springer-Verlag, London, UK

[SPA-02] G. Spanoudakis: *"Plausible and adaptive requirement traceability* Engineering, pages 135-142, 2002.

[STA-Dod] Departement of Defence, Standard Practice for system Safety. MIL-STD-882D,10 February, 2000.

[STE-03] J. O. Steven: *"Analysing the Impact of Changing Software Requirements-Based Methodology"*. Faculty of the Louisiana State University and Agricultural and Mechanical College, Thesis,December 2003.

[T]

[TRA-97] I.Traoré, A.E.K. Sahraoui: *"A Multiformalism Specification Framework with Statecharts and VDM"*. 22nd IFAC/IFIP Workshop on Real Time Systems, Lyon, France, 1997.

[TRA-04] I.Traoré; M.H.Eljamal, Y.M.Lui, A.E.K.Sahraoui: "UML-PVS requirement specification and verification". 14th Annual International Symposium INCOSE 2004, Toulouse (France), 20-25 Juin 2004.

[Z]

[ZIE-94] E. Zierau : *"Use of the Formal Method RAISE in Practice"*. In *Proceedings of SAFECOMP '94*. 1994.

[ZOW-02] D. Zowghi; N. Nurmuliani: *"A Study of the Impact of Requirements Volatility on Software Project Performance"*. 9th Asia-Pacific Software Engineering Conference, Gold Coast, Australia, 2002.

Sites Internet:

[Site1]http://www.capcomespace.net/dossiers/espace_europeen/ariane/ariane5/AR501/AR501_rapport_denquete.htm

[Site 2] http://www.mtq.gouv.qc.ca/fr/securite/ferroviaire/index.asp

[Site 3] http://www.mtq.gouv.qc.ca/fr/securite/routiere/index.asp

[STA-IEC] http://www.iec.ch/zone/fsafety/fsafety_entry.htm

[STA-EIA] http://www.afis.fr/doc/normes/normes3.htm

[STA-IEEE] http://www.afis.fr/doc/normes/normes2.htm

[GRO-Foc] http://www.npd-solutions.com/focusgroups.html

[TEM-Vol] http://www.volere.co.uk/

ANNEXE

ANNEXE A

1. ELICITATION DE L'OUTIL

Dans cette annexe, nous allons présenter l'élicitation des exigences de l'outil qu'on l'a développé (en utilisant MUSTER). Le pas initial lors de l'éliçitation des exigences de l'outil, était l'éliçitation des buts pour le développement de l'outil, deuxièmement les contraintes du développement de l'outil, troisièmement l'élicitation des cas d'utilisations, puis les dispositifs, et finalement nous avons terminé avec une élicitation des exigences fonctionnelles (non fonctionnelles) de l'outil. Ci-dessous, nous allons présenter les buts primaires, les cas d'utilisation de notre outil.

1.1. Les buts primaires

LES BUTS PRIMAIRE	
ID	**Description des Buts**
111	Le développement d'un outil qui supporte la méthodologie de la recherche d'impact d'une évolution sur la sécurité.
113	L'outil doit supporter un modèle de traçabilité basé sur une base de données relationnelle.
114	L'outil indique l'impact de l'application d'un changement d'une exigence sur le système (de point de vue sécurité).
115	L'outil doit être un moyen pour faciliter l'application de la modification au cours du développement des systèmes complexes.
118	L'outil doit supporter le système de sécurité selon les différents types des exigences de

	sécurité (*ou les exigences sécuritaires*).
119	L'outil doit être implémenté d'une façon que, si quelqu'un voudra ajouter un autre processus pour la recherche d impact lors d'une demande de changement sera possible (c.à.d. l'application d'une évolution de l'outil sera possible).
126	L'outil doit supporter la gestion des exigences dans le cadre de l ingénierie des systèmes (basé sur la norme : EIA-632) et l'ingénierie des exigences.
123	L'outil supporte un format fixe pour le changement des exigences, tel que le numéro de la demande, la description de la demande de changement, et les parties concernées avec le changement.
125	Le but du développement de l outil est pour acquérir des expériences pratiques.
127	lors d'un impact sur la sécurité, l'outil doit indiquer le (ou les) fichier(s) impacté par le changement, afin d'effectuer l'analyse d'impact.sur la sécurité

1.2. Les cas d'utilisation

LES CAS D'UTILISATION		
ID	**Nom**	**Acteur**
161	Analyser l'impact lors d'une demande de modification/évolution.	cher/Indus
162	Gérer les tests de cas pour les exigences.	développeur
163	Gérer les projets durant le développement des systèmes complexes.	Industriel
168	La possibilité d'utiliser l'outil dans une étude théorique.	Chercheur

165	Gérer la vérification formelle de la recherche d'impact lors d'une demande de modification sur la sécurité.	Analyseur
166	Faciliter la gestion des exigences techniques du produit final.	Manager
167	Décomposer le système selon l'EIA-632	développeur

ANNEXE B

1. L'ALGORITHME DE CHANGEMENT

/*Processus d'initialisation*/

Initialisation de la demande

/*Processus d'Evaluation*/

Chercher les parties impactées par la demande de changement ;

Si (existe un lien entre la demande de changement et le système de sécurité?)

Alors

Créer une version provisoire pour le système ;

Spécifier et analyser l'impact du changement ;

Répète

Vérifié la validité des exigences sécuritaires ;

Jusqu'à ce que toutes les exigences liées à la sécurité seront validées;

Donner l'autorité pour appliquer le changement **;**

/* Processus de Réalisation*/

Si (l'autorité d'appliquer le changement est approuvée ?)

Alors

Répète

Appliquer le changement sur les parties impactées;

Mise à jour pour les liens de traçabilité

Jusqu'à ce que toutes les parties impactées seront changées ;

/*Processus de Validation*/

Répète

Générer les tests de cas pour tester le changement

Jusqu'à ce que tous les tests seront appliqués ;

Si (tous les tests sont approuvés ?)

Alors

Nouvelle configuration du système est approuvée;

Le changement est validé

Sinon générer d'autres demandes de changement

Sinon le changement est rejeté

Sinon changement est accepté

ANNEXE C

1. BUT

Goal ::
goal_id : Goal_Id <-> *replace_goalid*
goal_description : Description <-> *replace_goaldesc*
goal_date : Date <-> *replace_goaldate*
goal_precondition : PreCondition <-> *replace_goalprecondition*
goal_exitcondition : ExitCondition <-> *replace_goalexitcondition*
goal_sourceid : Stakeholder_Id <-> *replace_goalsourceid*
testgoals : TESTGOALS .TestGoals <-> *replace testgoals*

2. MODULE

BBlock ::
bblock_id : Building_Block_Id <-> replace_endproduct_id
bblock_desc: Description <-> replace_description
hasparent : EndProduct_Parent_Id <-> replace_hasparent
endproducts : ENDPRODUCTS_.EndPros <-> replace_endproducts
enablingproducts : ENABLINGPRODUCTS_.EnabPros <-> replace_enablingproducts
techrequirements:TECHREQUIREMENTS_.techrequirements<-> replace_techreqs
assignrequirements:ASSIGNREQUIREMENTS_.assignrequirements<-> replace_assignreqs

3. STAKEHOLDER

Stakeholder ::
stak_id : Stakeholder_Id <-> replace_Stakeholder_id
name : Name <-> replace_name
family : Family <-> replace_family
fonction : Fonction <-> replace_fonction
phone : Phone <-> replace_phone
office : Office <-> replace_office
mail : Mail <-> replace_mail
submit : TechRequirement_Id-set <-> replace_submit
roles : ROLES_.Roles <-> replace_roles

4. PRODUIT FINAL

EndPro ::
endPro_id : EndProduct_Id <-> replace_endproduct_id
endPro_description : Description <-> replace_description
endPro_subsystems : SUBSYSTEMS_.subsystems<->replace_subsystems

ANNEXE D

1. PRINCIPES DE SECURITE

Dans le tableau ci-dessous, on présente les différents principes de sécurité qui doivent être respectés durant le développement des systèmes complexes et qui doivent être présente au sein du plan de sécurité.

Principe	**Définitions**
Principe 1	Si le système fonctionne correctement il ne faut pas qu'il y ait des risques inacceptables ni indésirables
Principe 2	La sécurité du système dans le mode normale ne doit pas être dépendant des actions correctives et des procédures faites par les opérateurs.
Principe 3	Il ne faut pas qu'il y ait l'existence d'une seule action qui transforme l'état du système dans un état critique
Principe 4	Si la combinaison de l'Echec E1 avec E2 transforme le système dans un état critique, le premier échec doit être détectée par le système avant que le deuxième aura lieu.
Principe 5	Les erreurs des logiciels ne doivent pas causes des risques inacceptables ou indésirables pour le système.
Principe 6	Les risques inacceptable et indésirables doivent être éliminé par la conception.
Principe 7	les activités d'entretien exigées pour préserver ou réaliser des niveaux de risque seront exécutées. Les qualifications de personnel exigées pour mettre en application les activités d'entretien devront également identifiées

Tableau D1- Le plan de sécurité doit respecter les 7 principes.

ANNEXE E

1. LE MODULE : TYPES

```
object TYPES :
 class
  type
   Severity ==
    Catastrophic | Severe | Major | Minor | None,
   Likelihood_Category ==
    High | Moderate | Low | Remote | Negligeable,
   SIL ==
    Level0 | Level1 | Level2 | Level3 | Level4 | Level5,
   Safety == Enviorement | Health | Hardware,
   Requirement_Type ==
    Functional |
    Non_Functional |
    mk_NFR_Safety_Constraint(const_on_saf : Safety) |
    mk_FR_Safety_Critical(
      critical_safety : Safety,
      severity : Severity,
      likelihood : Likelihood_Category,
      level : SIL) |
    mk_FR_Detection_of_Violation(detect_safety : Safety) |
    mk_FR_React_of_Violation(react_safety : Safety) |
    mk_FR_Prevention_of_Safety_Risk(risk : Safety) |
    mk_FR_Prevention_of_Safety_incident(
      incident : Safety),
   Fit_Cretirion = Text,
   Identificator = Nat,
   Description = Text,
   Day = {| x : Nat :- x <= 31 /\ x >= 1 |},
```

```
Month = {| x : Nat :- x <= 12 /\ x >= 1 |},
Year = {| x : Nat :- x >= 2006 /\ x <= 2015 |},
Date = Day >< Month >< Year,
Satisfaction = {| x : Nat :- x <= 5 |},
Dissatisfaction = {| x : Nat :- x <= 5 |},
Dependecy = (Goal_Id >< TechRequirement_Id)-set,
HasTechrequirements =
  (Goal_Id >< TechRequirement_Id)-set,
Rational = Text,
Version = Nat,
Validation = Bool,
RequestChange_Id = Nat,
Number = Nat,
TechRequirement_Id = Text >< Number,
TestCase_Id = Nat,
End_Product_Name = Text,
Building_Block_Name = Text,
Building_Block_Number = Nat,
Building_Block_Id =
  Building_Block_Name >< Building_Block_Number,
EnablingProduct_Id = Text >< Number,
EndProduct_Id = Text,
Enabling_Type ==
  System_Developement | Training | Retirement,
File_Id = Text,
Name = Text,
Family = Text,
Fonction == Manager | Engineer | Technicien,
Phone = Text,
Building = Text,
Level_of_Building = Nat,
Office_Number = Text,
Office =
  Building >< Level_of_Building >< Office_Number,
Mail = Text,
```

```
Role_Id = Role_Type >< Number,
Parent_Id = TechRequirement_Id,
EndProduct_Parent_Id = EndProduct_Id,
Stakeholder_Id = Nat,
Project_Name = Text,
Goal_Id = Text >< Number,
Project_Id = Text >< Number,
PreCondition = Text,
ExitCondition = Text,
TestGoal_Id = Nat,
Request_Date = Date,
Change_Type ==
  mk_Change_TechRequirement(
    pid : Project_Id,
    gid : Goal_Id,
    tr_id : TechRequirement_Id) |
  mk_Change_EndProduct(
    project_id : Project_Id, endp_id : EndProduct_Id) |
  mk_Change_EnablingTechnology(
    enabtech : EnablingTechnology_Id),
Change_Goal :: pid : Project_Id   gid : Goal_Id,
Change_Number = Nat,
Change_Kind == Add | Remove,
Rationale = Text,
Impact == Delay | Cost | Safety | No_Impact,
Message ==
  No_Safety_related_to_System |
  RC_Impact_on_Safety |
  No_Safety_Impact_with_RC |
  Initialization_Process_not_Finish,
Evaluation_Date = Date,
Change_Decision = Bool,
Decision_Date = Date,
Reuqest_Application = Bool,
Application_Date = Date,
```

```
    Change_Confirmation = Bool,
    Validation_Date = Date,
    Enabling_Technology_Name = Text,
    EnablingTechnology_Id = Nat,
    Role_Type ==
      Applicant |
      Evaluator |
      Decision |
      Modificator |
      Verificator,
    Extention = Text,
    Dot = {| c : Text :- c = "." |},
    File_Name = Text >< Dot >< Extention,
    File_Path = Text,
    Action_Message ==
      Adding_is_already_finish |
      Deleting_is_already_finish |
      Project_is_valid_use_a_RC
End
```

ANNEXE F

1. SPECIFICATION FORMELLE

Dans cette annexe, nous allons présenter la spécification formelle de la solution logique présentée au niveau du module « Contrôle ». En fait, cette spécification est une spécification concrète et applicative, chaque fonction sera spécifiée formellement et explicitement. La solution logique du module contrôle, est divisée en deux fichiers : le premier fichier (nommé : TYPES) pour la spécification des types, et un autre fichier (nommé : ROAD) pour pouvoir spécifier les fonctionnalités du contrôle et qui permet à valider la contrainte de sécurité (CS2) spécifiée avec la logique temporelle.

1.1. Les types

```
scheme TYPES =
  class
    type
      State == state1|state2|state3,
      Jeton={|p:Nat:-p<10|},
      Sensor==sensor1|sensor2,
      Lights==light_R1|light_R2,
      Road_Lights=Lights-m->Bool,
      Road_Sensors=Sensor-m->Bool,
      Petri_net = State -m-> Jeton
  end
```

1.2. Les fonctionnalités

Dans cette partie, nous allons détailler comment on a spécifier les fonctionnalités qui décrivent le comportement du module Contrôle présenter avant l'application de l'évolution. Le fichier ROAD, sera divisé en trois parties : la première partie est pour spécifier les fonctionnalités pour pouvoir rentrer/sortir dans la zone critique; la deuxième partie de la spécification est pour spécifier le système de transition; la dernière partie de la spécification est pour intégrer la contrainte de sécurité : CS2, afin de la valider.

```
        T

scheme ROAD =
  class
    type
      Road_System ::
       petri : T.Petri_net <-> update_state
       lights : T.Road_Lights <-> upadte_lights
       sensors : T.Road_Sensors <-> update_sensor

value
     /*spécification de l'état initial du système*/
     init: Road_System =
       mk_Road_System(
          [T.state1 +> 50, T.state2 +> 0, T.state3 +> 0],
          [T.light_R1 +> false, T.light_R2 +> false],
          [T.sensor1 +> false, T.sensor2 +> false]),

     /*spécification de la fonction : use_road1*/
     use_road1 : Road_System -> Road_System
     use_road1(s) is
       mk_Road_System(
          [T.state1 +> petri(s)(T.state1) - 1,
           T.state2 +> petri(s)(T.state2) + 1,
           T.state3 +> petri(s)(T.state3)],
          [T.light_R1 +> true, T.light_R2 +> false],
          sensors(s))
     pre can_use_road1(s),

     /*spécification de la fonction : use_road2*/
     use_road2 : Road_System -> Road_System
     use_road2(s) is
       mk_Road_System(
          [T.state1 +> petri(s)(T.state1) - 1,
           T.state2 +> petri(s)(T.state2),
           T.state3 +> petri(s)(T.state3) + 1],
          [T.light_R1 +> false, T.light_R2 +> true],
          sensors(s))
     pre can_use_road2(s),

     /*spécification de la fonction : leave_road1*/
     leave_road1 : Road_System -> Road_System
     leave_road1(s) is
       mk_Road_System(
          [T.state1 +> petri(s)(T.state1) + 1,
           T.state2 +> petri(s)(T.state2) - 1,
           T.state3 +> petri(s)(T.state3)],
          [T.light_R1 +> false, T.light_R2 +> false],
          sensors(s))
     pre can_leave_from_road1(s),

     /*spécification de la fonction : use_road2*/
     leave_road2 : Road_System -> Road_System
     leave_road2(s) is
       mk_Road_System(
          [T.state1 +> petri(s)(T.state1) + 1,
```

```
        T.state2 +> petri(s)(T.state2),
        T.state3 +> petri(s)(T.state3) - 1],
       [T.light_R1 +> false, T.light_R2 +> false],
       sensors(s))
    pre can_leave_from_road2(s),


/**************PRE-CONDITIONS**************/

    /*Pré condition pour pouvoir utiliser la Route1*/
    can_use_road1 : Road_System -> Bool
    can_use_road1(s) is
      wait_on_road1(s) /\ petri(s)(T.state1) ~= 0,

    /*Pré condition pour pouvoir utiliser la Route2*/
    can_use_road2 : Road_System -> Bool
    can_use_road2(s) is
      ~ wait_on_road1(s) /\ wait_on_road2(s) /\
      petri(s)(T.state1) ~= 0,

    /*Pré condition pour pouvoir sortir de la Route1*/
    can_leave_from_road1 : Road_System -> Bool
    can_leave_from_road1(s) is
      leave_from_road1(s) /\ petri(s)(T.state2) ~= 0,

    /*Pré condition pour pouvoir sortir de la Route2*/
    can_leave_from_road2 : Road_System -> Bool
    can_leave_from_road2(s) is
      leave_from_road2(s) /\ petri(s)(T.state3) ~= 0,


    /***************OBSERVERS******************/

    /*Voiture existe sur la Route1*/
    wait_on_road1 : Road_System -> Bool
    wait_on_road1(s) is sensors(s)(T.sensor1),

    /*Voiture sort de la Route1*/
    leave_from_road1 : Road_System -> Bool
    leave_from_road1(s) is ~ sensors(s)(T.sensor1),

    /*Voiture existe sur la Route2*/
    wait_on_road2 : Road_System -> Bool
    wait_on_road2(s) is sensors(s)(T.sensor2),

    /*Voiture sort de la Route2*/
    leave_from_road2 : Road_System -> Bool
    leave_from_road2(s) is ~ sensors(s)(T.sensor2)

transition_system
[Road_Cars_Lights]
in

            s:T.Road_Sensors
local
                        sys:Road_System:=init
```

in

[T1_use_road1]can_use_road1(sys)==>sys'=use_road1(sys)

[=]

[T2_use_road2]can_use_road2(sys)==>sys'=use_road2(sys)

[=]

[T3_leave_road1]can_leave_from_road1(sys)==>sys'=leave_road1(sys)

[=]

[T4_leave_road2]can_leave_from_road2(sys)==>sys'=leave_road2(sys)

end

ltl_assertion

/*contrainte de sécurité : CS2 */

[light_property]Road_Cars_Lights|-(G(~(lights(sys)(T.light_R1)/\lights(sys)(T.light_R2))))

end

RESUME

Le travail de la thèse porte sur la problématique de l'évolution des exigences et son analyse d'impact sur la sécurité. Au cours du développement des systèmes, les parties prenantes demandent l'application des évolutions, afin d'améliorer leurs fonctionnalités. L'occurrence d'une évolution affecte plusieurs aspects comme: la sécurité, le coût du développement et le délai. Lorsque le développement concerne un système complexe où le nombre des exigences est de l'ordre de dizaines de milliers, alors les demandes des évolutions rendent l'analyse d'impact du changement de plus en plus difficile.

Notre étude est située dans le contexte d'ingénierie système et ingénierie des exigences, en intégrant des modèles formels et outils supports associés pour la vérification des propriétés de sécurité. La méthodologie est basée sur la norme industrielle de l'EIA-632 et sur le format VOLERE d'ingénierie des exigences en intégrant un modèle de traçabilité. De cette démarche ingénierie système, on a développé une méthodologie associée et un système d'information du processus de changement/évolution des exigences. Enfin, ce travail présente la méthodologie de la recherche d'impact lors de la demande de changement, et les outils qui supportent la méthodologie, en se basant sur la méthode formelle (RAISE), qui nous permet d'analyser l'impact de l'évolution sur la sécurité.

Mots clés : évolution des exigences, ingénierie système, méthodes formelles, traçabilité.

ABSTRACT

The work of the thesis concerns the problems of requirements evolution and their impacts on safety. During systems development, stakeholders require the application of evolutions, in order to improve systems' functionalities. The occurrence of an evolution affects several aspects like: safety cost and deadline. The development of complex systems is about tens of thousands of requirements, which make the impact analysis of evolutions more and more difficult.

Our study is located in the context of systems engineering and requirements engineering, with integration for a formal models and associated tools for safety analysis. Our methodology is based on the industrial standard of the EIA-632 and VOLERE format of requirements engineering. From the systems engineering approach, we have developed an associated methodology and tool containing an information system for the process of requirements evolution. Finally, this work presents the methodology of requirements evolution, and the associated tools, by using the formal method (RAISE) which enables us to analyze the impact of evolution on safety.

Key words: requirements evolution, systems engineering, formal methods, traceability.