UNIVERSITÉ SULTAN MOULAY SLIMANE
Faculté des Sciences et Techniques
Béni-Mellal

N° d'ordre : 97/2016

Centre d'Études Doctorales « Sciences et Techniques »
Formation doctorale « Mathématiques et Physique appliquées »

THÈSE

Présentée par

Mr ABDESSAMAD BALOUKI

Pour obtenir le diplôme de

DOCTORAT

Spécialité : Informatique

Option : Systèmes mécatroniques

SPECIFICATION ET VERIFICATION FORMELLES DES SYSTEMES MECATRONIQUES SELON LE MODELE DE REFERENCE DES SYSTEMES REPARTIS OUVERTS RM-ODP

Soutenue publiquement le 18/07/ 2016 devant le Jury

A.MERBOUHA	**Professeur à la FST– USMS – Béni Mellal**	**Président**
A.ABOUNADA	**Professeur à la FST– USMS – Béni Mellal**	**Rapporteur**
M.AIT LAFKIH	**Professeur à la FST– USMS – Béni Mellal**	**Rapporteur**
M.MABROUKI	**Professeur à la FST– USMS – Béni Mellal**	**Examinateur**
M.RAMZI	**Professeur à l'ESTS– UMVR – Salé**	**Rapporteur**
M.SBIHI	**Professeur à l'ESTS– UMVR – Salé**	**Examinateur**
M.FAKIR	**Professeur à la FST– USMS – Béni Mellal**	**Directeur de thèse**

TABLES DES MATIERES

TABLES DES MATIERES 1

INTRODUCTION GENERALE 4

Le besoin d'un standard pour le traitement réparti ouvert 5

Processus de spécification formelle en B 6

Méthodologie de conception pour les systèmes répartis 7

Ce mémoire est structuré en cinq chapitres : 8

CHAPITRE I : ETAT DE L'ART 10

I.1 Introduction : 10

I.2 Systèmes mécatroniques 10
I.2.1 Définition d'un système mécatronique 11
I.2.2 Besoin d'un cadre unificateur 12
I.2.3 Sureté de fonctionnement 13

I.3 Le modèle de référence pour le traitement réparti ouvert 13
I.3.1 Vue générale du modèle RM-ODP 14
I.3.2 Le modèle objet 15
I.3.3 Le modèle architectural 16

I.4 La méthode formelle Event-B 24
I.4.1 Logique et théorie des ensembles 24
I.4.2 Modèles Event-B 26
I.4.4 Raffinements 33
I.4.5 Ingénierie dirigée par les modèles et approche MDA 35

CHAPITRE II : LA SPECIFICATION D'UN SYSTEME MECATRONIQUE AU POINT DE VUE ENTREPRISE 40

II.1 Introduction 40

II.2. Cas d'étude : Système de freinage 41

II.3. Spécification d'un système mécatronique au point de vue Entreprise 42
II.3.1. Point de vue Entreprise 42
II.3.2. Le langage d'entreprise 42

II.4. Utilisation de l'UML pour spécifier le point de vue de l'Entreprise 48
II.4.1. Communauté du système de freinage 49

À ma famille

II.4.2. Modélisation des rôles et leurs relations. 49
II.4.3. Résumé des correspondances 50

II.5. Conclusion 50

CHAPITRE III : SPECIFICATION FORMELLE DES SYSTEMES MECATRONIQUES 51

III.1. Introduction : 51

III.2. Approche classique 52

III.3. Approche formelle 52

III.4. Spécification en Event-B d'un système (ABS) 54
III.4.1. Principe de fonctionnement du système ABS 54
III.4.2. Description informelle du système ABS 55
III.4.3. Modèles de conception 58
III.4.4. Exigences du système ABS 62
III.4.5. Modèle initial : Liaison entre le contrôleur et le frein 64
III.4.6. Premier raffinement : liaison entre la pédale de frein et le contrôleur 66
III.4.7. Deuxième raffinement : liaison entre le contrôleur et le moteur 69
III.4.8. Conclusion 70

CHAPITRE IV : SPECIFICATION DE LA QUALITE DE SERVICE EN MECATRONIQUE 71

IV.1. Introduction 71

IV.2. Point de Vue Ingénierie en mécatronique 72

IV.3. QoS du point de vue Ingénierie 73

IV.4. Formalisation d'un accord de QoS négocié 75
IV.4.1. Négociation bornée 76
IV.4.2. protocole de négociation bornée 77

IV.5. Spécification formelle des protocoles de négociation de QoS 78
IV.5.1. Document des exigences 78
IV.5.2 Stratégie de raffinement 80
IV.5.3. Modèle abstrait de départ 80
IV.5.4. Premier modèle de raffinement 81
IV.5.5. Second modèle de raffinement 83

IV.6. Cas d'étude : Système Mécatronique 85

IV.7. Conclusion 86

REMERCIEMENT

Je tiens à remercier très vivement Monsieur Mohamed FAKIR, Professeur à la Faculté des Sciences et Techniques de Béni Mellal, pour son soutien, sa disponibilité, sa patience, la collaboration étroite dans laquelle nous avons travaillé et son aide qui m'ont permis à mener à bien cette thèse.

J'exprime ma profonde gratitude à Monsieur Abdelkrim MARBOUHA, professeur à la Faculté des Sciences et Techniques de Béni Mellal, pour l'honneur qu'il me fait de présider mon jury de thèse. Qu'il trouve ici l'expression de ma profonde gratitude.

Je souhaite remercier Monsieur M.RAMAZI, professeur à l'École Supérieure de Technologie de Salé, Monsieur M.AIT LAFKIH, professeur à la Faculté des Sciences et Techniques de Béni Mellal et Monsieur A.ABOUNADA, professeur à la Faculté des Sciences et Techniques de Béni Mellal, pour avoir accepté la lourde charge d'être Rapporteur et pour leurs précieuses remarques qui m'ont permis d'améliorer ce manuscrit.

Je tiens à remercier Monsieur M.MABROUKI, professeur à la Faculté des Sciences et Techniques de Béni Mellal et Monsieur M.SBIHI, professeur à l'École Supérieure de Technologie de Salé, pour le temps qu'ils ont bien voulu consacrer à l'examen de ce document.

Mes remerciements vont naturellement à l'ensemble des collègues qui m'ont aidé ou tout simplement rendu le déroulement de cette thèse agréable. En particulier je voudrais remercier mes parents pour leur amour et leur soutien dans les moments difficiles.

Je tiens à remercier ma femme, pour son soutien et sa patience tout au long de la thèse.

Enfin, Je tiens à remercier mes frères et sœurs pour leur amitié et leurs encouragements.

CHAPITRE V : 87

MODELISATION ET TRANSFORMATION DU PROCESSUS COMPORTEMENTAL ODP DANS LE CONTEXTE DE MDA 87

V.1. Introduction 87

V.2. Langage d'exécution des processus métiers 88

V.3. Les concepts de comportement 89

V.4. Profil UML pour des processus automatisés de comportement 92

V.5. Génération d'un processus à partir d'un modèle UML 93

V.6. Conclusion 96

CONCLUSION GENERALE ET PERSPECTIVES 97

REFERENCES BIBLIOGRAPHIQUES 100

AVANT PROPOS

Ce travail a été effectué sous la direction scientifique de Monsieur le Professeur FAKIR Mohamed. L'étude réalisée a donné lieu aux publications (acceptés et soumises):

1. **A.Balouki**, A.Ettabbaa, A.Elhassnaoui, M.Mabrouki, Y.balouki, M.Fakir. *Security Analysis in Mechatronic Systems: Classic and Formal Approach.*soumise.

2. **A.Balouki**, A.Ettabbaa, A.Elhassnaoui, M.Mabrouki, Y.balouki, M.Fakir.*Mechatronic Enterprise Viewpoint Specifications*.Journal of Theoretical and Applied Information Technology,accépté pour publication le 31st July 2016. Vol. 89 No.2

3. **A. Balouki**;M. Sbihi; Y. Balouki. RM-ODP: A framework for Mechatronics systems.16th International Conference on Research and Education in Mechatronics (REM), Pages: 73 – 79, IEEE Conference Publications,November 2015

4. Y.Balouki, **A.Balouki**, M.El Far, A. Kriouile.*Transformation of Models in an MDA Approach for Collaborative Distributed Processes*.Collaborative Systems for Reindustrialization, IFIP Series Advances in Information and Communication Technology, Vol. 408, pp. 201-208, Springer, October 2013.

INTRODUCTION GENERALE

L'apparition des systèmes mécatroniques est une révolution pour le monde industriel, il affecte de plus en plus le monde du transport et en particulier le secteur automobile. L'utilisation des ces systèmes se généralise rapidement et influence maintenant tous les secteurs de l'industrie. Ces systèmes sont des structures réparties dont les composants, aussi bien matériels que logiciels, sont de types différents. Ils agissent indépendamment les uns des autres. L'assemblage de tels composants donne naissance à des systèmes fortement hétérogènes. Les applications que ces systèmes supportent sont elles mêmes constituées de composants répartis. L'interaction entre ces composants applicatifs constitue un des aspects du traitement réparti. Dans un système réparti ouvert idéal, les applications sont capables d'interagir même lorsqu'elles ont été développées dans des environnements différents. Cette capacité ne peut être obtenue que si ces environnements sont conformes à un même modèle conceptuel. Les objectifs du modèle RM-ODP sont la répartition, l'inter- fonctionnement et l'ouverture.

Parallèlement à l'émergence des systèmes mécatroniques, l'industrie de ses systèmes a fortement évolué ces dernières décennies. Cette évolution a eu pour objectif d'augmenter la qualité de leurs produits, mais une question reste essentielle : est-on sûr de ce que fait le système développé ? Et plus généralement, le système qui vient d'être développé répond-il aux exigences de sécurité ?

Pour répondre à ces nouvelles exigences émanant des industries développant des systèmes mécatroniques critiques, dont le dysfonctionnement peut entraîner des pertes humaines ou financières considérables, les techniques et les méthodes de développement ont dû évoluer. La validation traditionnelle consistant à vérifier a posteriori que, sur des ensembles de données passées en entrée du système, l'exécution mène bien au résultat attendu ne répondait plus au niveau d'exigences de sûreté de fonctionnement. L'une des techniques qui a émergé de cette évolution est le développement formel. Ce développement permet de s'assurer par un raisonnement mathématique de la conformité du système vis-à-vis des exigences.

Dans ce contexte, nous nous sommes intéressés à la spécification et la vérification formelle des systèmes mécatroniques basés sur l'utilisation de la méthode formelle Event-B.

RESUME

La mécatronique est la combinaison synergique et systémique de la mécanique, de l'électronique et de l'informatique en temps réel. L'intérêt de ce domaine d'ingénierie interdisciplinaire est de concevoir des systèmes automatiques puissants et de permettre le contrôle de systèmes complexes.

Avec la croissance rapide de la technologie, les fonctionnalités proposées par les constructeurs des systèmes mécatroniques sont devenus de plus en plus complexes, ces derniers sont confrontés à de nouveaux problèmes avec ces fonctions, qui dépendent de plusieurs calculateurs et actionneurs répartis sur des réseaux informatiques internes.

Dans ce travail nous nous concentrons sur les problèmes d'ingénierie et réingénierie des systèmes mécatroniques. Notre contribution repose sur deux axes, le premier consiste à soumettre à la communauté mécatronique un cadre unificateur, le RM-ODP. Ce cadre est défini comme un méta-standard qui vise à établir des normes qui permettent de tirer profit des avantages de la distribution de l'information dans un environnement possédant des unités hétérogènes de traitement de l'information et qui relèvent de différents domaines d'ingénierie. Dans le deuxième axe, nous proposons une méthode formelle afin de vérifier les lois qui gèrent la sûreté de fonctionnement des systèmes mécatroniques, ces lois sont formalisées en B événementiel.

Nous avons construit pas-à-pas une spécification formelle en Event-B d'un système mécatronique. Pour y parvenir, nous avons appliqué une stratégie de raffinement. Une telle stratégie a été appliquée avec succès et a donné naissance à une machine abstraite, des machines raffinées et des contextes. En outre, nous avons validé la spécification formelle obtenue en utilisant avec profit la plateforme RODIN.

Mots clés :

Mécatronique, RM-ODP, Langages de point de vue, Méthodes formelles, politiques ODP, Event-B, Rodin.

Cette méthode tire profit de la puissance des outils supportant les spécifications B tel que la plate forme Rodin. Notre approche consiste à considérer le modèle de plus haut niveau comme étant une abstraction du système envisagé. La modélisation est ensuite raffinée en ajoutant progressivement les propriétés dynamiques du système. De plus, on peut également garantir l'émergence de propriétés non fonctionnelles, telles que les propriétés de sécurité et de qualité. Cette décomposition par raffinements successifs prouvés de spécifications a pour objectif de permettre un développement modulaire, sûr et fiable des systèmes mécatroniques.

Le besoin d'un standard pour le traitement réparti ouvert

Dans un système mécatronique idéal, les composants devraient être capables d'interagir même lorsqu'ils ont été développés dans des environnements différents (développés par des organisations distinctes). Cette capacité ne peut être obtenue que si ces environnements sont conformes à un même modèle conceptuel. Le modèle de référence d'ODP (RM-ODP) fournit ce cadre [1][2].

Le modèle de référence pour le traitement réparti ouvert (RM-ODP) fournit un cadre dans lequel le support de la distribution, l'interfonctionnement et la portabilité peuvent être intégrés. Le RM-ODP est divisé en quatre parties :

- ✓ La partie « Vue générale » introduit le standard et son utilité. elle contient un aperçu général du modèle de référence ODP, en précise les motivations, le domaine d'application et la justification, et propose une explication des concepts clés, ainsi qu'une présentation de l'architecture ODP ;
- ✓ La partie « fondement » contient la définition des concepts et du cadre analytique pour la description normalisée des systèmes de traitement distribué. Ces concepts sont regroupés en plusieurs catégories ;
- ✓ La partie « sémantique architecturale » contient une formalisation des concepts de modélisation définis dans la partie fondement ;
- ✓ La partie « architecture » contient les spécifications des caractéristiques requises pour qualifier un traitement réparti d'ouvert. Il définit un cadre de travail composé de cinq points de vue, des langages de points de vue, des fonctions ODP et des transparences ODP.

Les cinq points de vue, appelés point de vue entreprise, information, traitement, ingénierie et technologie, fournissent une base pour la spécification des systèmes ODP :

LISTE DES FIGURES

Figure 1 : système mecatronique 12
Figure 2 : modele architectural de rm-odp 16
Figure 3 : la specification au point de vue ingenierie 21
Figure 4 : les objets du point de vue ingenierie 22
Figure 5 : la relation entre les points de vue odp 23
Figure 6 : les relations entre les contextes et les machines 28
Figure 7 : structure d'un contexte 29
Figure 8 : structure d'une machine.....30
Figure 9 : structure d'un evenement.....32
Figure 10. Pyramide de modelisation de l'omg 38
Figure 11 : MDA : un processus en y dirige par les modeles 39
Figure 12 : diagramme de cas du systeme de freinage.....43
Figure 13 : les concepts de la communaute du système de frein 44
Figure 14 : le contrat communautaire du système de freinage.....45
Figure 15 : les rôles d'enterprise du système de freinage 28
Figure 16 : la réalisation du rôle.....46
Figure 17 : clauses relatives au comportement de rôle 47
Figure 18 : structure de la politique du freinage 48
Figure 19 : la spécification du point de vue entreprise en mecatronique.....49
Figure 20 : la communaute du système de freinage et son objectif. ...**ERROR! BOOKMARK NOT DEFINED.**
Figure 21 : la structure de la communaute du système de freinage ....**ERROR! BOOKMARK NOT DEFINED.**
Figure 22 : approche formelle pour le developpement des systèmes.....54
Figure 23 : composants du système antiblocage abs dans une voiture 56
Figure 24 : premiere vue schématique du système 57
Figure 25 : deuxieme vue schématique du système 59
Figure 26 : action _ reaction.....60
Figure 27 : faible synchronisation entre l'action et la reaction .**ERROR! BOOKMARK NOT DEFINED.**
Figure 28 : forte synchronisation entre l'action et la reaction.....63
Figure 29 : les connexions du controleur 64
Figure 30 : liaison du controleur à l'etrier & disque.....65
Figure 31 : connexion pedale controleur.....68
Figure 32 : meta-modèle de la qos au point de vue ingenierie.....75
Figure 33 : negociation bornee de la qos.....77
Figure 34 : schéma du protocole de négociation de QOS bornée 78
Figure 35 : meta-modèle du langage d'information-concepts de comportement 91
Figure 36 : une classe uml modélise le processus du comportement BPEL 93
Figure 37 : développement d'un processus 96

- ✓ Le point de vue entreprise : focalise sur les objectifs, le domaine d'application et les stratégies de ce système, à l'aide des concepts suivants: rôle, communauté, processus, étape, objectif, artefact, acteur et ressource ;
- ✓ Le point de vue information : permet la définition des informations traitées par les différentes ressources systèmes, à l'aide des concepts suivants: schéma dynamique, schéma statique et schéma invariant ;
- ✓ Le point de vue traitement : décrit la décomposition fonctionnelle d'un système et les interactions entre les interfaces des différents objets, à l'aide des concepts: signal, opération et flux ;
- ✓ Le point de vue ingénierie : décrit les moyens mis en œuvre pour que les objets du système interagissent, à l'aide des concepts: grappe, capsule, noyau, nœud, canal, souche et éditeur de liens ;
- ✓ Le point de vue technologie : définit les technologies logicielles et matérielles utilisées, à l'aide des concepts standards implantables, implantation et informations supplémentaires sur l'exécution.

A chaque point de vue correspond un langage qui définit les concepts et les règles pour la spécification des systèmes ODP dans ce point de vue.

Le modèle de référence identifie un certain nombre de fonctions fondamentales pour la construction des systèmes ODP. Celles-ci prennent en charge les exigences des points de vue entreprise et ingénierie et sont suffisamment génériques pour être appliquées à un grand nombre d'applications. On cite entre autres : la fonction de courtage, la fonction de sécurité, la fonction de gestion, etc.

Processus de spécification formelle en B

La complexité grandissante des systèmes mécatroniques et la maîtrise des risques inhérents à leur utilisation devient impérative et appelle, de ce fait, une rigueur et une précision accrues lors de leur élaboration. Aussi l'adoption de méthodes et de techniques sûres et fiables, reposant sur des fondements mathématiques, s'impose-t-elle, désormais, en tant que garant de la sécurité. Ces méthodes, dites formelles, ont été élaborées afin d'assurer un niveau aussi élevé que possible en matière de précision et de cohérence. Leur avantage majeur réside, en somme, dans le fait qu'elles sont basées sur les mathématiques, ce qui

LISTE DES TABLEAUX

Tableau 1: correspondance entre langage d'entreprise mecatroniques et uml construction..... 51
Tableau 2 : mapping entre concepts de comportement et constructions uml..... 92
Tableau 3 : mapping entre constructions uml et concepts bpel..... 93

LISTE DES ABREVIATIONS

BPEL4WS	(Business Process Execution Language for Web Services) : le langage de composition de services
CCM	CORBA Component Model
CORBA	Common Object Request Broker Architecture
MDA	Model Driven Architecture (Architecture basée sur les modèles ou Architecture dirigée par les modèles)
OCL	Object Constraint Language
OMG	Object Management Group
QoS	Quality of Service
UML	Unified Modeling Language
XML	eXtensible Markup Language

permet, d'une part, de neutraliser tous les risques d'ambiguïté et d'incertitude, et d'autre part, de parvenir à un produit fini qui répond aux spécifications requises.

La méthode B [3], se présente comme une méthode formelle couvrant toutes les phases d'un cycle de développement formel. Le passage d'une phase à une autre dans un processus de développement en B correspond à un incrément (dit raffinement) de spécifications et est guidé par un ensemble d'obligations de preuves dont l'objectif est de valider les composants en cours de développement. Cette décomposition par raffinements successifs prouvés de spécifications a pour objectif de permettre un développement modulaire, sûr et fiable des systèmes. L'abstraction utilisée pour spécifier les composants B se base sur la théorie des ensembles et la logique du premier ordre, pour spécifier les propriétés statiques, et un langage de substitutions pour spécifier les propriétés dynamiques des machines. Un composant B est la donnée d'une machine abstraite qui spécifie les propriétés du composant, et de ses raffinements.

Les possibilités d'application de la méthode ont été étendues, sans changer la théorie sous-jacente, dans ce qu'il est convenu d'appeler le B-événementiel. On a repensé la notion de fonctionnement et de raffinement d'un projet B. Les systèmes ne sont plus vus comme des composants contrôlés, mais comme des systèmes autonomes où différents événements activent différents comportements au sein du système. Le raffinement est pensé comme une vue de plus en plus précise de ces événements et de ces comportements.

Le déplacement de paradigme de modèles invoqués vers des modèles autonomes, nous a permis de mieux intégrer la preuve dans le processus de conception et de validation des systèmes distribués ouverts ODP. L'idée initiale est de spécifier un projet B normal, et d'introduire graduellement des contraintes dynamiques, compatibles avec les contraintes statiques déjà spécifiées, afin de s'orienter vers une implémentation, tout en préservant la correction des propriétés validées par rapport aux spécifications intermédiaires. Des preuves de correction doivent être effectuées.

Méthodologie de conception pour les systèmes répartis

Le développement des applications ouvertes dans un environnement réparti a atteint aujourd'hui un premier niveau de maturité industrielle qui reste cependant limité : l'apparition de chaînes de développement logiciel permettant la fabrique de lignes de produit, à l'instar d'autres industries, n'est pas encore à l'ordre du jour. Cependant des disciplines émergentes y concourent comme l'ingénierie des modèles, confirmée entre autres par l'OMG (Object Management Group) et son initiative MDA (Model Driven Architecture).

L'Architecture Dirigée par les Modèles est un thème en pleine expansion aussi bien dans le monde académique que dans le monde industriel. Elle apporte un changement important dans la conception des applications, la pérennité des savoir-faire, des gains de productivité et bénéficie des avantages des plates formes existantes. C'est une approche qui préconise de construire des applications réparties interopérables via les modèles. L'idée est d'automatiser la transformation des modèles métier d'une application pour en arriver à une solution technique sur une plate-forme de notre choix. Ainsi, en cas de changement d'architecture technique, il suffirait de regénérer un autre code du même modèle métier. Pour cela, MDA envisage de construire une chaîne automatisée de production de logiciels. Celle-ci doit être suffisamment outillée pour supporter tout le cycle de vie des applications à réaliser et garantir l'industrialisation de leur fabrication. Elle devrait être modulable et générique.

De nombreuses techniques manipulant les modèles permettent la mise en œuvre de cette approche (méta-modélisation, transformation de modèles, …). Toutefois, il n'existe pas de consensus sur une méthodologie pour les appliquer. Parmi celles les plus répandues, la méthodologie ODAC (Open Distributed Applications Construction), orientée MDA, dont l'objectif est d'assurer une industrialisation de la production logicielle. Généralement ces méthodes permettent le développement de système avec la notation UML selon une sémantique ODP. Cependant, cette sémantique est ambigüe, et pour l'essentiel informelle.

Ce mémoire est structuré en cinq chapitres :

Dans le premier chapitre, nous présentons principalement les concepts et principes nécessaires à la compréhension de notre sujet. Dans un premier temps, nous présentons les systèmes mécatroniques. Dans un deuxième temps, nous introduisons le modèle de référence pour le traitement réparti ouvert RM-ODP. Puis, nous présentons la méthode formelle Event-B. Enfin, nous résumons les concepts clé de l'ingénierie dirigée par les modèles et de l'approche MDA.

Dans le deuxième chapitre, nous nous intéressons particulièrement à la spécification des systèmes mécatroniques au point de vue entreprise, qui se concentre sur les rôles, les objectifs, le domaine d'application et les politiques du système. Bien que le modèle de référence ODP fournit des langages pertinents. Nous avons défini un méta-modèle des concepts de base et leurs relations en utilisant le langage de modélisation unifié UML. Ainsi, nous fournissons un guide méthodologique de développement de système mécatronique permettant à un constructeur de spécifier son système, avec la notation UML selon une sémantique ODP.

Dans le troisième chapitre, nous avons présenté la problématique de développement des systèmes complexes sensibles aux erreurs. Pour éviter les erreurs inhérentes au développement de ces systèmes, nous avons proposé un processus de développement formel (Event-B). La méthode formelle Event-B permet d'obtenir une spécification cohérente et digne de confiance du futur système en utilisant les outils associés à Event-B : générateur d'obligations de preuves, prouveur automatique et interactif. Nous avons appliqué le processus de développement proposé sur le Système de freinage ABS (Anti-lock brake systems).

Dans le quatrième chapitre, nous proposons un modèle de négociation des exigences de qualité de service entre les composants d'un système mécatronique en utilisant les techniques de raffinement de la méthode B. Afin de vérifier l'exactitude du modèle conçus, il a été implémenté sur la plateforme Rodin [4]. Cette Plate-forme fournit un support efficace pour le raffinement et la preuve mathématique.

Le cinquième chapitre décrit la méthodologie de modélisation des aspects comportementaux du langage d'information ODP. Notre démarche commence par la création d'un profil UML spécifique au comportement. Ce profil sera transformé en artefacts BPEL en définissant une transformation entre les concepts du comportement du langage d'information et les concepts du langage BPEL. Cette transformation, orientée MDA, va être assurée par l'utilisation de l'outil Rational rose.

Finalement, nous concluons ce travail en récapitulant les résultats obtenus et en évoquant différentes possibilités de continuation et d'extension de travail déjà réalisé.

CHAPITRE I : ETAT DE L'ART

I.1 Introduction :

Le développement des systèmes mécatroniques impliquent différents acteurs avec différentes spécialités qui utilisent différents concepts [5]. Pour avoir une bonne communication entre les différentes parties prenantes et l'intégration permanente des services sophistiqués, nous proposons une approche unifiée basée sur le modèle de référence pour les systèmes distribués ouverts.

L'approche classique de développement des systèmes mécatroniques n'apporte pas des guides méthodologiques favorisant l'écriture des cahiers des charges [6], les risques d'erreurs sont à toutes les phases et ne se détectent qu'aux phases finales et leur détection nécessite des tests très poussés [7]. Aujourd'hui, pour répondre aux enjeux de sécurité imposés par le marché, une nouvelle approche de conception des systèmes est nécessaire pour permettre l'intégration des différentes technologies sûres de fonctionnement dès la première phase de développement. La méthode formelle (Event-B) apparaît comme la méthodologie la plus adaptée au développement des systèmes mécatroniques.

Dans ce chapitre nous présentons principalement les systèmes mécatroniques et la sûreté de fonctionnement de ses systèmes. Ensuite nous présentons le modèle de référence pour le traitement réparti ouvert (RM-ODP), dans le dernier paragraphe nous allons présenter la méthode B et son extension B-événementielle.

I.2 Systèmes mécatroniques

L'apparition des systèmes mécatroniques est une révolution pour le monde industriel, il affecte de plus en plus le monde du transport et en particulier le secteur automobile. L'utilisation de ces systèmes se généralise rapidement et influence maintenant tous les secteurs de l'industrie. La mécatronique a bouleversé la conception et la fabrication des systèmes complexes. En particulier, son introduction dans le secteur automobile a profondément modifié les processus de développement et de fabrication. Ainsi, la voiture n'est plus conçue comme un dispositif mécanique qui porte quelques commandes

électroniques, mais comme un système mécatronique [8], où les composants des différentes technologies sont entièrement intégrés [9].

I.2.1 Définition d'un système mécatronique

La définition du mot mécatronique (mechatronics en anglais) a été proposé la première fois par un ingénieur de Yaskawa Electric Co. du Japon en 1969 pour désigner le contrôle des moteurs électriques par ordinateur [10]. Ce terme a par la suite évolué. La mécatronique est « la combinaison synergique de l'ingénierie mécanique de précision, de la commande électronique et du système informatique dans la conception des produits et des processus de fabrication » (définition officielle par le Comité Consultatif de Recherche Industrielle et de Développement de la Communauté Européenne - Industrial Research and Development Advisory Committee of the European Community) [11], [12]. Cette définition établit le caractère multidisciplinaire de la mécatronique qui associe dans la conception et la fabrication d'un produit plusieurs secteurs d'activité de technologies différentes. Le mot-clé ici est « synergique », pris dans le même sens que dans la définition du génie mécanique, qui est perçue comme la combinaison synergique de la dynamique des fluides, de la mécanique des solides, de la thermodynamique et de la science des matériaux [13]. La mécatronique n'est pas intrinsèquement une science ou une technologie, elle doit être considérée comme une attitude, une manière fondamentale de regarder et de faire des choses, et, par sa nature, elle exige une approche unifiée [14].

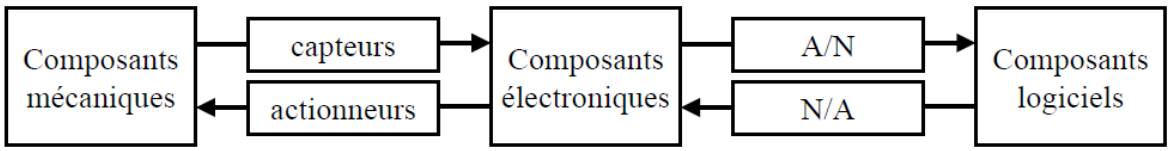


Figure 1 : Système mécatronique [18].

Le système mécatronique de la figure .1 intègre de la mécanique, de l'électronique et du logiciel, mais également des systèmes hydrauliques, pneumatiques et des systèmes thermiques. Cet exemple montre qu'il est important que le système soit conçu comme un ensemble autant que possible [15]. La synergie induite par les systèmes mécatroniques conduit à une combinaison intelligente de technologies. Cette synergie mène alors à des solutions et à des performances supérieures, qui ne pourraient pas être obtenues par des applications séparées.

L'avènement des systèmes mécatroniques dans l'industrie (en particulier l'industrie automobile) a entraîné de nouvelles contraintes, telles que :

– l'assimilation de plusieurs technologies ;

– les interactions entre les différentes entités fonctionnelles ;

– la prise en compte de la dynamique du système (le fonctionnement en temps réel, événementiel et l'intégration des nombreux états possibles) ;

– l'impossibilité de réaliser des tests exhaustifs.

Malgré ces contraintes, la mécatronique apporte des avantages indéniables comme : la baisse des coûts, la satisfaction client par les solutions innovantes proposées, la réponse positive à des exigences sociétales de plus en plus importantes - pollution, consommation, sécurité des passagers et piétons [16]. Par essence, la mécatronique est une conjugaison de technologies différentes, elle requiert pour son développement des équipes pluridisciplinaires avec des langages et des méthodes très différentes entre eux. Cette nouvelle approche génère des risques supplémentaires. La garantie de sûreté de fonctionnement, de fiabilité, devient alors essentielle dans le développement des systèmes mécatroniques [17]. Aujourd'hui, pour répondre aux enjeux qualité/coûts/délais imposés par le marché, une nouvelle approche de conception des systèmes est nécessaire pour permettre l'intégration des différentes technologies sûres de fonctionnement dès la première phase de développement. La méthode formelle apparaît comme la méthodologie la plus adaptée au développement des systèmes mécatroniques.

I.2.2 Besoin d'un cadre unificateur

Le développement de produits mécatroniques implique de multiples parties prenantes qui ont différents points de vue et utilisent donc différents concepts, modèles et outils pour répondre à leurs préoccupations d'intérêt. L'article de Martin Törngren et al [5] persuade que l'accent doit être mis d'avantage sur les relations entre les points de vue au niveau des personnes, des modèles et des outils afin de pouvoir répondre à l'évolution et les exigences des produits mécatroniques. Les points de vue sont utilisés pour définir le vocabulaire, les hypothèses et les contraintes requises pour assurer une bonne communication (niveau personnes). Le modèle de dépendance comprend les relations entre les propriétés des produits appartenant à différents points de vue, et la façon dont ces dépendances se rapportent aux prévisions et décisions (niveau modèle). Les modèles d'intégration d'outils décrivent les relations entre les outils en termes de traçabilité, l'échange de données (niveau outil). cet article a mis l'accent sur le besoin d'un nouvel acteur qui assume la responsabilité de

communication et de coordination. Dans notre travail nous avons proposé une approche unificatrice basée sur le modèle de référence pour les traitements des systèmes distribués ouverts.

I.2.3 Sureté de fonctionnement

La complexité des systèmes mécatroniques les rend sensibles aux erreurs produites durant les phases de spéciation, de conception, de réalisation, de vérification et de validation. Une erreur produite lors des phases initiales a un prix plus élevé que celle produite lors des phases ultérieures. La garantie de sûreté de fonctionnement, de fiabilité, devient alors essentielle dans le développement des systèmes mécatroniques [19]. L'approche classique n'apporte pas des guides méthodologiques favorisant l'écriture des cahiers des charges [6], les risques d'erreurs sont à toutes les phases et ne se détectent qu'aux phases finales et leur détection nécessite des tests très poussés [7].

Aujourd'hui, pour répondre aux enjeux de sécurité imposés par le marché, une nouvelle approche de conception des systèmes est nécessaire pour permettre l'intégration des différentes technologies sûres de fonctionnement dès la première phase de développement. La méthode formelle (Event-B) [45] apparaît comme la méthodologie la plus adaptée au développement systèmes répartis ouverts.

I.3 Le modèle de référence pour le traitement réparti ouvert

Le modèle de référence ODP (ISO/CEI 10746) se compose de:

– La Rec. UIT-T X.901 | ISO/CEI 10746-1[1]: aperçu général: elle contient un aperçu général du modèle de référence ODP, en précise les motivations, le domaine d'application et la justification, et propose une explication des concepts clés, ainsi qu'une présentation de l'architecture ODP;

– La Rec. UIT-T X.902 | ISO/CEI 10746-2 [2]: fondements: elle contient la définition des concepts et le cadre analytique à utiliser pour la description normalisée des systèmes de traitement répartis;

– La Rec. UIT-T X.903 | ISO/CEI 10746-3 [3]: architecture: elle contient la spécification des caractéristiques d'un système réparti ouvert. Ce sont les contraintes auxquelles les normes ODP doivent se soumettre;

– La Rec. UIT-T X.904 | ISO/CEI 10746-4 [4]: sémantique d'architecture: elle contient une formalisation des concepts de modélisation ODP définis dans la Rec. UIT-T X.902 | ISO/CEI 10746-2, articles 8 et 9. La formalisation s'obtient en interprétant chaque concept à partir d'éléments des différentes techniques normalisées de descriptions formelles.

I.3.1 Vue générale du modèle RM-ODP

La norme de traitement réparti ouvert (ODP), développée conjointement par l'ISO et l'ITU-T, fournit un modèle de référence qui définit un cadre architectural pour la construction de systèmes et applications réparties ayant un ensemble de caractéristiques importantes telles que l'ouverture, la facilité d'intégration, la flexibilité, la modularité, la sécurité, la transparence, le contrôle de la qualité de service. Ce modèle de référence ne vise pas à fournir une méthodologie concrète de conception ou de développement. Pourtant, il fournit un cadre de description des architectures selon plusieurs niveaux d'abstraction. Le modèle de référence ODP (RM-ODP) définit deux modèles : un modèle objet et un modèle architectural que nous présentons ci-après.

La vue générale du modèle RM-ODP est présentée dans la figure.2 suivante :

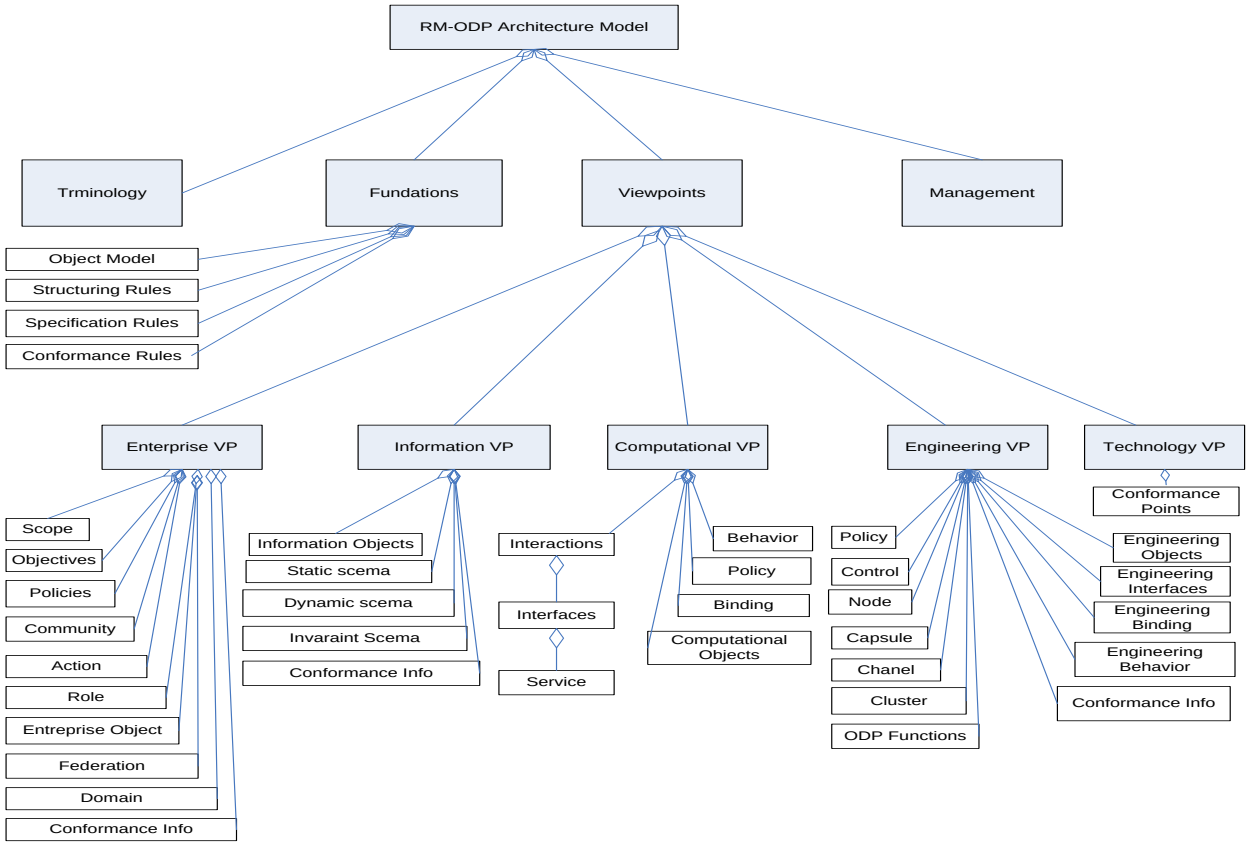


Figure 2 : Modèle architectural de RM-ODP [22]

I.3.2 Le modèle objet

Le modèle de référence ODP a pour but d'offrir un cadre conceptuel qui permet de spécifier rigoureusement l'architecture de systèmes répartis dans un environnement de communication et d'exécution hétérogène. Il définit pour cela un modèle générique d'architecture, et normalise des *points de vue*, qui décrivent chacun un aspect différent du système.

ODP repose sur un modèle orienté objet. Un *objet* est l'entité qui résulte de l'encapsulation d'un ensemble de données et des *opérations* (ou *méthodes*) qui peuvent être réalisées sur ces données. Le service offert par un objet, représenté par son *interface*, est dissocié de son implantation.

L'interface d'un objet est son point d'accès pour les autres objets. Elle caractérise les interactions que celui-ci peut avoir avec son environnement. Un objet ne possède habituellement qu'une seule interface, composée d'un ensemble de méthodes rendues ainsi accessibles aux autres objets. Dans le cadre du modèle RM-ODP, un objet peut avoir plusieurs interfaces. Par exemple, il peut disposer d'une interface de *service*, qui permet de l'invoquer et d'une interface de *contrôle*, destinée à son administration. ODP autorise

également la construction d'objets composites, par l'agrégation de plusieurs objets coopérants. La composition permet de décrire une relation hiérarchique entre objets. Ainsi composer deux objets permet d'en générer un nouveau, appelé objet composite. Réciproquement, la décomposition est un procédé de raffinement permettant de passer d'une application répartie complexe à un ensemble d'objets plus simples qui pourront à leur tour être décomposés. Composition et décomposition sont des termes et des activités de spécification duales, qui s'appliquent non seulement aux objets mais également aux comportements.

Le modèle de référence ODP ne fait pas d'hypothèse sur l'application de ce modèle. Un objet peut être aussi élémentaire qu'un simple entier ou être d'une complexité arbitraire. Un objet peut également afficher n'importe quel comportement interne. Il peut en particulier mettre en œuvre des formes quelconques de parallélisme. Enfin, les interactions entre objets ne sont pas contraintes : elles peuvent être asynchrones, synchrones ou isochrones, sous une forme discrète ou continue.

I.3.3 Le modèle architectural

Le modèle architectural ODP est construit sur la notion **de point de vue**.

Un point de vue est une subdivision d'une spécification d'un système complexe. Il permet pendant la phase de conception de considérer le système sous un angle particulier en ne s'intéressant qu'à la partie de spécification relative à celui-ci. Chaque point de vue représente une abstraction différente du même système. Les points de vue ne forment pas une séquence fixe et ne doivent donc pas être assimilés à une structuration en couches. De même, ils ne sont pas créés dans un ordre fixe selon une méthodologie de conception. Notons d'ailleurs que le modèle de référence ODP ne constitue pas en soi une méthodologie de spécification de systèmes répartis, même si beaucoup l'utilisent comme tel. Il définit une architecture qui utilise pour les besoins de sa spécification une séparation en cinq points de vue ci-après.

I.3.3.1 Les points de vue

❖ Point de vue Entreprise

Le point de vue Entreprise définit une vision conceptuelle de l'architecture étudiée. Il décrit les entités de l'entreprise, c'est-à-dire les communautés et les rôles, les comportements et les interactions entre ces derniers, les acteurs, les processus, les règles de gestion, les droits et les obligations, ce que l'on appelle de façon générale la politique de l'entreprise. Il fournit

une spécification du système dans l'environnement avec lequel il interagit. Cette spécification s'intéresse plus spécialement aux objectifs et à la politique du système. Généralement, ce point de vue exprime les besoins et les exigences définies par l'organisation.

Un concept de structuration majeur d'une spécification d'entreprise est celui de **communauté**. Une communauté est une configuration d'objets d'entreprise formée pour remplir un objectif dit métier. Elle modélise une collection d'entités (êtres humains, système de traitement d'information, ressources variées, etc) sujets à un contrat implicite ou explicite gouvernant leur comportement collectif. Au sein d'une communauté, les objectifs des membres sont contraints pour être conforme à l'objectif de la communauté.
Une spécification d'entreprise inclut au moins la description d'une communauté dans laquelle le système ODP est vu comme un objet unique d'entreprise interagissant avec son environnement. Cette communauté est désignée sous le terme (<S>community).

Une spécification d'entreprise peut inclure la description de plus d'une communauté. Elle peut ainsi être structurée en un ensemble de communautés qui interagissent, chacune de ces communautés étant alors vue comme un objet composite (appelé le c-objet).

Une spécification d'entreprise est ainsi composée des spécifications de différents éléments tels que communautés, rôles, objets d'entreprise etc. Selon le choix du modélisateur et le niveau de détail souhaité, chacun de ces éléments peut être spécifié par les caractéristiques de l'élément, ou par le type de l'élément ou par un gabarit de l'élément. La norme ne fait aucune prescription sur le processus de spécification ou sur le niveau d'abstraction utilisée dans une spécification d'entreprise.

L'objectif de la communauté est donc ce qui motive son existence. Au sein d'une communauté, chaque objet d'entreprise joue un rôle. Un objet peut jouer plusieurs rôles, dans la même communauté ou dans des communautés distinctes. Il peut participer à différents processus. Réciproquement, un rôle peut être rempli par différents objets, mais de façon successive dans le temps et non simultanée. Autrement dit, à un instant donné, un rôle est rempli par un et un seul objet. Le rôle d'acteur dans une communauté est distinct de celui d'artefact. Vis-à-vis d'une action, l'acteur est l'objet qui participe à l'action et l'artefact est l'objet qui est référencé dans l'action. Jouer un rôle d'acteur dans une communauté signifie alors que l'objet est acteur pour au moins une action de ce rôle tandis que jouer un rôle d'artefact dans une communauté signifie que l'objet est artefact pour toutes les actions de ce rôle.

Assigner des objets aux rôles de la communauté permet de peupler la communauté. Le comportement d'un objet auquel on assigne un rôle doit alors être cohérent avec le comportement identifié par ce rôle. Cette assignation peut varier dynamiquement pendant la durée de vie de la communauté. De même, la création et la destruction de rôles sont envisageables. Une spécification d'entreprise doit établir les circonstances de tels changements. L'inclusion de ces changements dans la spécification relève d'un choix de modélisation, i.e., les comportements associés à ces changements peuvent être explicites ou non. La terminaison d'une communauté est également un choix de modélisation. Elle peut se produire car l'objectif a été atteint, ou qu'il y a eu échec dans la recherche de l'objectif. Le comportement de terminaison peut être inclus explicitement ou non dans la spécification.
Les politiques, quant à elles, expriment les contraintes sur le comportement des objets jouant un rôle d'acteurs. Une politique s'applique à la communauté, à un ou plusieurs rôles ou à un type d'actions. Elle peut être exprimée comme une obligation, une autorisation, une permission ou une interdiction, les définitions suivantes devant s'appliquer à ces termes :

- **Obligation** : prescription stipulant la nécessité d'un comportement particulier ;
- **Autorisation** : prescription stipulant qu'un comportement particulier ne doit pas être empêché ;
- **Permission** : prescription autorisant un comportement particulier ;
- **Interdiction** : prescription stipulant qu'un comportement particulier ne doit pas se manifester.

Un type particulier de communauté est la fédération, qui est une communauté de domaines établie à des fins de coopération. Le concept de fédération permet d'établir les principes généraux d'interfonctionnement entre domaines et de décrire les politiques gouvernant cet interfonctionnement.

- **Point de vue Information**

Le point de vue Information est employé pour définir la sémantique de l'information et la sémantique du traitement de l'information nécessaires à une application définie selon le modèle ODP. Cette définition est faite en utilisant trois schémas : statique, invariant et dynamique, qui décrivent l'état et la structure d'un objet.

- le schéma statique capture l'état et la structure d'un objet à un certain moment particulier (par exemple, un état d'initialisation ou un état d'exception) ;

- Le schéma invariant définit la limite de l'état et la structure d'un objet à tout moment ;
- Le schéma dynamique spécifie le changement autorisé de l'état et de la structure d'un objet.

Les schémas peuvent également être employés pour décrire des relations ou des associations entre les objets. Un schéma peut se composer d'autres schémas pour décrire les objets complexes ou composés. Les schémas statique et dynamique doivent respecter les contraintes de tout schéma d'invariant.

Comme RM-ODP n'impose pas une méthode pour décrire les points de vue, les spécifications du point de vue Information d'une application d'ODP peuvent être exprimées par de nombreuses méthodes, par exemple, par les modèles entités-relations, les schémas conceptuels, la méthode formelle Z, ou le langage UML [23].

- **Point de vue Traitement**

Le point de vue Traitement s'intéresse à la décomposition fonctionnelle d'un système en objets qui interagissent grâce à leurs interfaces et ce, en faisant abstraction de la répartition.

La spécification de ce point de vue contient une configuration des objets de traitement, les spécifications des actions de ces objets, la spécification des interactions entre objets, les spécifications des interfaces qui supportent les interactions, la spécification des liaisons pour supporter les interfaces, les contraintes qui s'appliquent sur les interactions entre les objets de traitement, et les contrats d'environnements pour assurer le traitement correct des contraintes pour les objets et leurs environnements [24].

- **Point de vue Ingénierie**

Une spécification d'ingénierie définit les fonctions et les mécanismes requis pour prendre en charge l'exécution et l'interaction répartie entre les objets d'un système ODP. Contrairement au point de vue traitement qui permet de considérer quand et pourquoi les objets interagissent, le point de vue ingénierie s'intéresse à la façon dont ils interagissent (le "comment").

Le point de vue Ingénierie est employé pour décrire des aspects répartition d'un système d'ODP. Il définit un modèle pour l'infrastructure distribuée du système. Le point de vue Ingénierie n'est pas concerné par la sémantique de l'application d'ODP, excepté pour déterminer les conditions et la transparence de la répartition. Par ailleurs, il s'intéresse d'une

part aux moyens pour prendre en charge l'interaction répartie des objets, et d'autre part aux mécanismes de placement des objets sur les ressources d'exécution.

Les entités fondamentales décrites dans le point de vue Ingénierie sont des objets et des canaux. Des objets peuvent être divisés en deux catégories : les objets dits basiques (correspondant aux objets définis dans les spécifications du point de vue Traitement) et les objets d'infrastructure (par exemple, un objet de protocole) qui sont déterminés dans le point de vue Ingénierie. Un canal correspond à une liaison (binding) ou à un objet de liaison (binding) dans les spécifications Traitement.

Le canal : Un canal fournit le mécanisme de communication et contient les fonctions de transparence exigées par les objets basiques de niveau Ingénierie, comme indiqués dans les contrats d'environnement dans les spécifications Traitement. Le schéma ci-dessous illustre le canal entre un objet de client et l'objet de branche de banque. Le secteur ombragé est le canal, composé des talons, des lieurs, et des objets de protocole. Des talons et des lieurs sont employés pour fournir les diverses transparences de distribution.

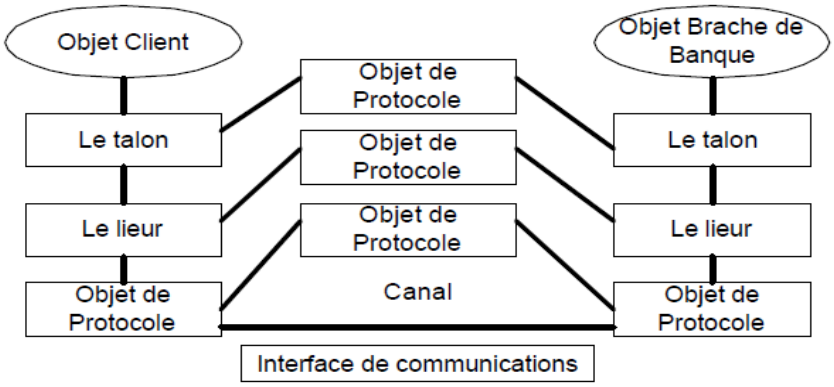


Figure 3 : La spécification au point de vue Ingénierie [1]

Les structures Ingénierie

Le point de vue Ingénierie de RM-ODP préscrit la structure d'un système d'ODP sans définir les composants techniques. Cependant, les objets suivants constituent la base de description des ressources d'exécutions des objets basiques du point de vue Ingénierie (voir Figure 4).

Cluster : une configuration d'objets basiques d'ingénierie formant une unité pour les traitements de type activation, désactivation, réactivation, restauration, migration.

Gérant de cluster : un objet gérant les objets dans un cluster.

Capsule : une configuration d'objets d'ingénierie formant une unité pour l'encapsulation des traitements et du stockage, par exemple une machine virtuelle ou un processus. Une capsule contient un ensemble de clusters.

Gérant de Capsule : un objet gérant les objets dans une capsule.

Nœud : un objet qui regroupe un ensemble de fonctions de type traitement, stockage et communication, par exemple un ordinateur.

Noyau : un objet qui coordonne les traitements, le stockage et la communication pour les objets au sein du nœud auquel il appartient, par exemple un système d'exploitation.

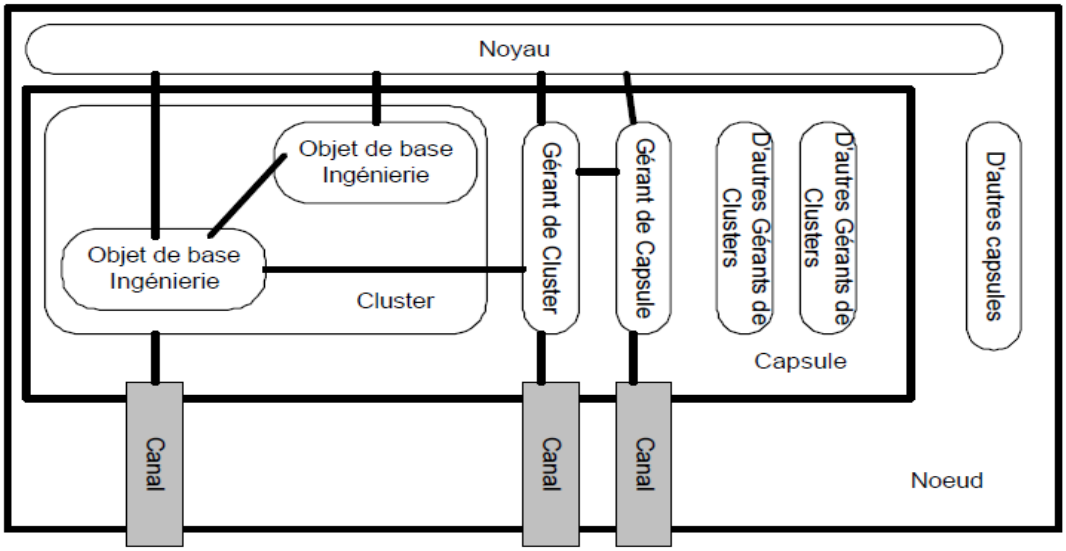


Figure 4 : Les objets du point de vue Ingénierie [1]

Étant donné ces définitions, les règles structurantes suivantes sont définies :

- chaque nœud a un objet de noyau ;
- chaque objet de noyau peut soutenir plusieurs capsules ;
- chaque capsule peut contenir plusieurs clusters ;
- chaque cluster peut contenir des objets basiques Ingénierie ;
- chaque objet de base Ingénierie peut contenir plusieurs activités.

Toute la communication inter-clusters se fait par l'intermédiaire des canaux.

❖ Point de vue Technique

Des spécifications du point de vue Technique d'un système d'ODP décrivent l'exécution de ce système et l'information exigée pour le test. Il s'intéresse aux choix matériels et logiciels

effectués pour implanter les spécifications résultant des autres points de vue. Ces aspects sortent du champ de normalisation.

La Figure 5 montre les relations existantes entre les parties regroupant les concepts ODP. Le fait qu'il y ait des relations entre les points de vue ne consiste pas à créer un système en couches. Chaque point de vue est une abstraction du système spécifié et peut ensuite être présent dans les couches basses ou hautes des réseaux. Les fonctions de transparence et de gestion sont surtout présentes dans le point de vue ingénierie, ces fonctions traitant en fait de la répartition du système ODP.

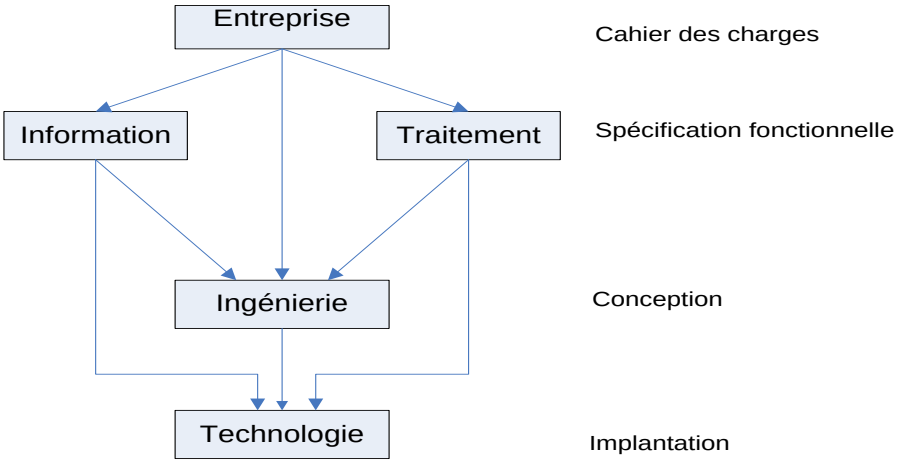


Figure 5 : La relation entre les points de vue ODP

I.3.3.2 Les langages de points de vue

Afin de représenter un système ODP selon un point de vue donné, il est nécessaire de définir un ensemble structuré de concepts qui permettent d'exprimer la partie de spécification relative à ce point de vue. Cet ensemble forme un langage de point de vue. Les termes de chaque langage et les règles de structuration de ces termes sont définis en utilisant les concepts du modèle objet.

❖ Le langage d'entreprise

Il s'utilise pour définir les objectifs, le domaine d'application et les politiques d'un système ODP. Le système ODP et l'environnement dans lequel il opère sont représentés par un ou plusieurs objets d'entreprise regroupés en une communauté d'entreprise liés par un contrat et par les rôles joués par ces objets. La finalité d'une spécification d'entreprise est d'identifier les actions requises par les politiques gouvernant la communauté spécifiée. Ces

politiques fixent le cadre de l'application du contrat de la communauté en définissant les actions permises, obligatoires ou interdites à un objet.

❖ Le langage d'information

Le langage d'information définit la sémantique de l'information et la sémantique de traitement de l'information dans un système ODP. Celui-ci est exprimé en termes d'une configuration d'objets d'information. Les relations entre les objets d'information sont modélisées comme des objets d'information ou comme parties des objets d'information. Une spécification consiste en un ensemble de gabarit d'objet d'information. Un gabarit est décrit en termes de trois schémas qui sont le schéma d'invariant, le schéma statique et le schéma dynamique. Le schéma d'invariant est un ensemble de prédicats sur un ou plusieurs objets d'information qui doivent toujours être vrais. Le schéma statique spécifie l'état d'un ou plusieurs objets à un instant donné. Le schéma dynamique spécifie les changements d'états autorisés par un ou plusieurs objets d'information. Ces changements peuvent inclure la création ou la suppression d'objets d'information. Les schémas statique et dynamique doivent respecter les contraintes de tout schéma d'invariant.

❖ Le langage de traitement

Le langage de traitement est un modèle de programmation réparti abstrait, basé objet pour décrire la structure et l'exécution d'application répartie dans un environnement ODP. Il constitue une spécialisation du modèle objet en considérant l'objet de traitement comme unité de répartition. Une application répartie est alors structurée en un ensemble d'objets interagissant à travers leurs interfaces. Le langage de traitement définit un modèle d'interaction, des interfaces de traitement et un modèle de liaison.

❖ Le langage d'ingénierie

Le langage d'ingénierie permet d'exprimer une spécification d'ingénierie qui définit les fonctions et les mécanismes requis pour prendre en charge l'exécution et l'interaction répartie entre les objets d'un système ODP.

D'un point de vue ingénierie, une application répartie est un ensemble d'objets d'ingénierie de base interagissant via un canal de communication. Les concepts de ce langage sont nœud (ordinateur), noyau (système d'exploitation), capsule (espace d'adressage) et grappe (modules liés pour former une exécutable) et objet d'ingénierie (module d'un programme).

I.3.3.3 Les fonctions ODP

Le modèle de référence identifie un certain nombre de fonctions fondamentales pour la construction des systèmes ODP [8-9]. Celles-ci prennent en charge les exigences des points de vue traitement et ingénierie et sont suffisamment génériques pour être appliquées à un grand nombre d'applications. Ces fonctions sont regroupées en quatre catégories :

1) les fonctions de gestion système qui incluent la gestion des nœuds, d'objets, de grappes et de capsules ;
2) les fonctions de conteneur qui incluent les fonctions de stockage, de gestion de base d'informations, de conteneurs de types et de courtage ;
3) les fonctions de coordination qui incluent la fonction de notification d'événement, de sauvegarde et de reprise, de désactivation, et réactivation, de groupe, de duplication, de migration, de ramasse miettes pour les références d'interfaces d'ingénierie et de transaction ;
4) les fonctions de sécurité qui incluent les fonctions de contrôle d'accès, d'audit, d'authentification, d'intégrité, de confidentialité, de non répudiation et de gestion de clé.

I.4 La méthode formelle Event-B

Event-B [3] est une extension de B qui permet la spécification des systèmes réactifs, des algorithmes séquentiels, concurrents et distribués. Cette méthode utilise des approches mathématiques [25] basées sur la théorie des ensembles et la logique de prédicats [26].

I.4.1 Logique et théorie des ensembles

Le langage logico-ensembliste d'Event-B est basé sur la logique classique du premier ordre et la théorie des ensembles.

Les symboles utilisés pour exprimer des prédicats logiques sont :

$\top$vrai

$\bot$faux

$P \wedge Q$ conjonction

$P \vee Q$ disjonction

$\neg P$ négation

$P \Rightarrow Q$ implication

P⇔Q équivalence

∀ x1, x2, …, xn . P(x1, x2, …, xn) quantification universelle

∃x1, x2, …, xn . P(x1, x2, …, xn) quantification existentielle

Le langage logico-ensembliste d'Event-B supporte les notations ensemblistes usuelles comme :

- l'inclusion (⊆)
- l'inclusion stricte (⊂)
- l'union (∪)
- l'intersection (∩)
- la différence d'ensemble (\)
- l'ensemble vide (∅ ou {})
- l'ensemble des parties non vides (ℙ1)
- l'ensemble des parties (ℙ)
- les ensembles énumérés {x1, x2, …, xn}

En outre, le langage logico-ensembliste d'Event-B supporte les concepts mathématiques couple, relation et fonction. Les concepts usuels sur les relations (s->t) sont définis en Event-B tels que :

- domaine (dom)
- codomaine (ran)
- relation d'identité (id)
- relation réciproque (r^{-1})
- image d'un ensemble par une relation (r[s])
- et la composition (r;q)

De plus, le langage logico-ensembliste d'Event-B propose des opérateurs pour restreindre les relations :

- la restriction sur le domaine (S◁ r)
- l'anti-restriction pour enlever des éléments du domaine (S r)
- la corestriction qui est une restriction sur le codomaine (r▷ T)
- l'anti-corestriction pour enlever des éléments du codomaine (r T)
- et la surcharge permet d'obtenir une relation r p à partir de deux autres relations r et p.

Les fonctions sont considérées comme des relations particulières. On distingue :

- injective partielle (s⇸t)
- surjective partielle (s↠t)
- totale (s→t)

- totale injective (s↣t)
- totale surjective (s↠t)
- et totale bijective (s⤖t)

Enfin, les couples en Event-B sont notés sous la forme x↦y.

Les constructions offertes par le langage logico-ensembliste permettent de typer les ensembles, les constantes et les variables décrivant la partie statique d'un modèle Event-B. En outre, elles sont utilisées pour décrire les propriétés invariantes d'un modèle Event-B sous forme des prédicats logiques. Enfin, elles décrivent les axiomes et théorèmes des modèles Event-B.

I.4.2 Modèles Event-B

Le modèle est le premier concept d'Event-B. Il est composé d'un ensemble de machines et de contextes. Une machine Event-B contient deux parties : statique et dynamique. La partie statique comporte les variables modélisant l'état du système. Ces variables sont typées et peuvent avoir des propriétés invariantes décrites par des prédicats logiques. La partie dynamique comporte des évènements permettant d'agir sur l'état du système. Le nouvel état obtenu suite au déclenchement de l'événement doit préserver l'invariant. Un événement particulier appelé Initialisation doit établir l'invariant. De plus, une machine Event-B peut comporter des théorèmes qui devraient êtres prouvés. Un contexte Event-B comporte les paramètres du système à modéliser : ensembles et constantes. Les propriétés de ces paramètres sont formalisées par des axiomes et des théorèmes à prouver.

Il est à noter qu'un modèle peut contenir uniquement des contextes (c'est un modèle qui représente une structure purement mathématique avec les constantes, les axiomes et les théorèmes), ou bien uniquement des machines (c'est un modèle non paramétré) ou bien les deux ensembles (c'est un modèle paramétré par les contextes). Pour ce troisième type de modèle, il existe un ensemble de relations entre les machines et les contextes, comme représenté dans la figure 6.

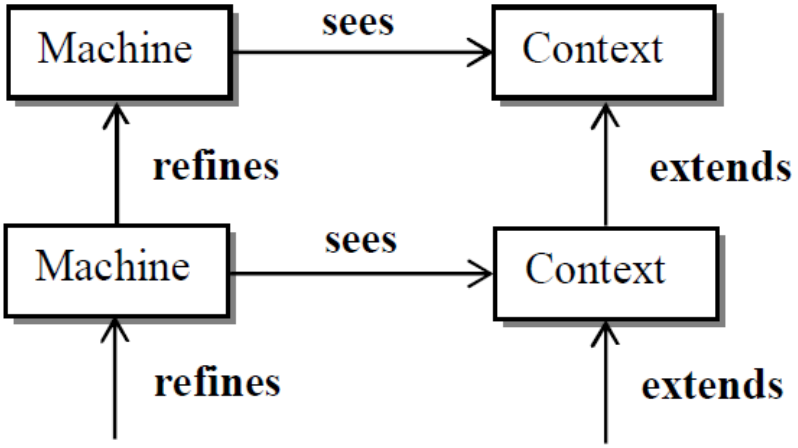


Figure 6 : Les relations entre les contextes et les machines

Les explications liées à la figure 6 couvrent :

-Une machine peut voir explicitement plusieurs contextes (ou pas du tout) ;

-Un contexte peut étendre explicitement plusieurs contextes (ou pas du tout) ;

-La notion d'extension de contexte est transitive: un contexte C1 qui étend explicitement un contexte C2, étend implicitement tous les contextes étendus par C2 ;

-Lorsqu'un contexte C1 étend un contexte C2, alors C1 peut utiliser les ensembles et les constantes de C2;

-Une machine voit implicitement tous les contextes étendus par un contexte qu'elle voit explicitement;

-Quand une machine M voit un contexte C, elle pourra utiliser les ensembles et les constantes de C ;

-Il n y'a pas de cycle dans les relations « refines » et « extend» ;

-Une machine raffine au plus une autre machine ;

-La relation de raffinement « refines » est transitive.

I.4.2.1 Contexte

Un contexte « *Context* » permet de spécifier des données statiques. Il se compose d'ensembles, de constantes avec leurs axiomes et éventuellement des théorèmes. La structure du contexte est constituée d'un ensemble de clauses introduites par des mots clés comme représentée dans la figure 7.

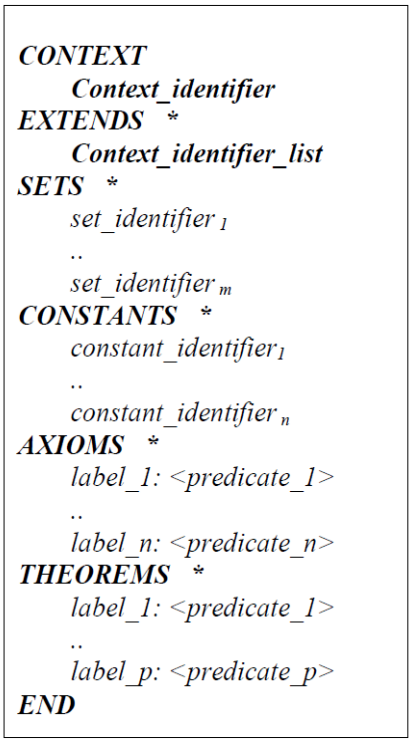


Figure 7 : Structure d'un contexte

-Les champs avec " * " peuvent être vides

Le contenu de chaque clause peut être décrit comme suit :

CONTEXT : permet de définir le nom du contexte. Il doit être distinct de tous les autres noms qui existent dans le modèle.

EXTENDS : contient la liste des contextes hérités.

SETS : regroupe la liste des ensembles qui ne sont pas vides et deux à deux disjoints s'ils sont de même type.

CONSTANTS : définit la liste des constantes introduites dans le contexte.

AXIOMS : définit la liste des prédicats que les constantes doivent respecter.

THEOREMS : définit les théorèmes qui vont être prouvés dans le contexte.

Les axiomes et théorèmes portent des étiquettes (labels). Celles-ci peuvent être personnalisées.

I.4.2.2 Machine abstraite

En Event-B, on distingue deux types de machine : machine *abstraite* introduite dans le modèle le plus abstrait et machine *de raffinement* (ou raffinée). Une machine contient essentiellement des variables et des évènements. Les variables sont données dans la clause *VARIABLES* et initialisées dans la clause *Initialisation*. La clause *Invariant* définit l'espace d'état des variables et les propriétés du système [27].

La structure d'une machine ressemble à la structure du contexte. Elle admet un ensemble de clauses introduites par des mots clés. La figure 8 montre la structure d'une machine.

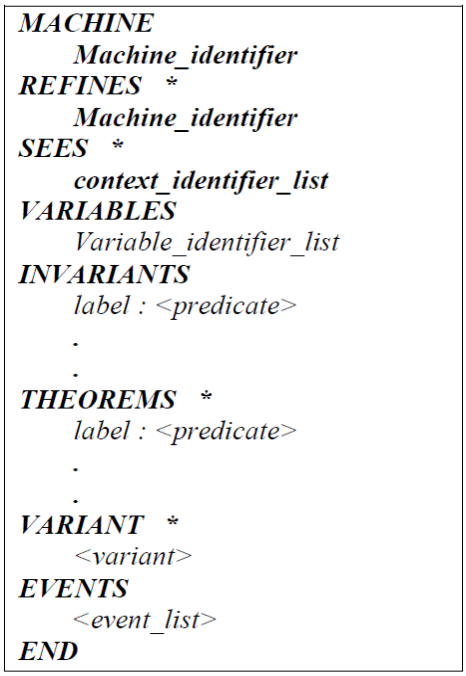


Figure 8 : Structure d'une machine

-Les champs avec " * " peuvent être vides

Voici la description du contenu de chaque clause :

-MACHINE : donne un identifiant à cette machine. Un tel identifiant doit être différent des identifiants de tous les autres composants du même modèle.

-REFINES : représente l'identificateur de la machine (si elle existe) à partir de laquelle raffine la machine actuelle. Une machine raffine au plus une autre machine.

-SEES **:** contient (s'il existe) la liste des contextes vus explicitement par cette machine. Par conséquent, elle peut utiliser les ensembles porteurs (carrier sets) et les constantes vus explicitement ou implicitement par cette machine (par la relation *extends*).
-VARIABLES : définit la liste des variables introduites dans la machine.

-INVARIANTS **:** sert au typage des variables et à décrire les contraintes (sous forme de prédicats) qu'elles doivent satisfaire ou respecter en permanence. Un invariant peut être :

- Invariant *propre* aux variables de la machine courante ;
- Invariant de *collage* reliant les variables de la machine concrète avec celles de la machine abstraite (raffinée).

-THEOREME **:** liste les différents théorèmes qui doivent être prouvés dans cette machine. Pour y parvenir, on utilise comme hypothèses les théorèmes précédents définis avant ce théorème ainsi que les axiomes et les théorèmes des contextes vus explicitement ou implicitement par cette machine.
-VARIANT **:** c'est une clause qui apparaît dans une machine raffinée et qui contient des événements convergents. Cette clause contient une expression de type entier naturel qui décroît par n'importe quel événement convergent ou de type ensemble fini dont sa cardinalité décroît.
-EVENTS **:** détaille les différentes transitions (événements) de la machine.

Dans la suite, nous présentons les constituants d'un événement Event-B.

I.4.2.3 Evénement

Tout événement en Event-B modélise une transition discrète et peut être défini par une relation *"avant-après"* notée *BA (x, x')*, où *x* et *x'* désignent respectivement la valeur des variables avant et après l'exécution des actions associées à l'événement. Cette relation varie d'une forme d'événement à une autre [27].
La forme générale d'un événement est représentée par la figure 9 :

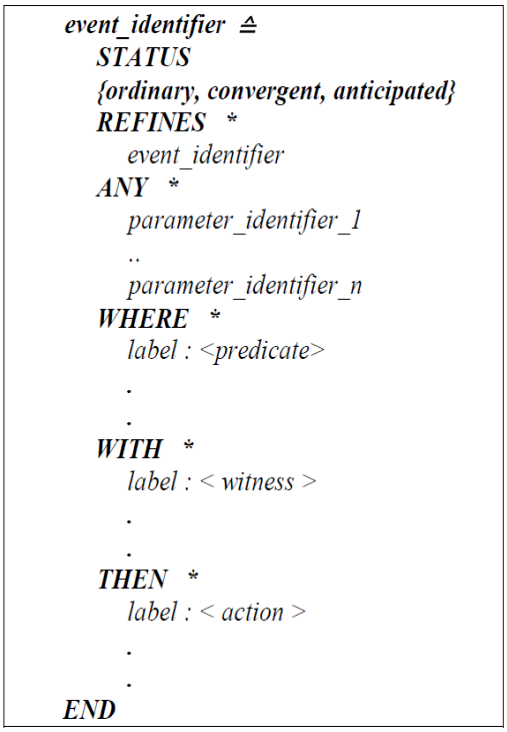


Figure 9 : Structure d'un événement

-Les champs avec " * " peuvent être vides

-STATUS : décrit l'état d'un événement. Un événement peut être :

- ordinary,
- convergent: il doit décrémenter le variant,
- anticipated: va être convergent ultérieurement dans un raffinement.

-REFINES : liste le(s) événement(s) abstraits que l'événement actuel raffine (s'il existe).

-ANY : énumère la liste des paramètres de l'événement.

-WHERE : contient les différents gardes de l'événement. Ces gardes sont des conditions nécessaires pour déclencher l'événement. Il faut noter que si la clause « any » est omise le mot clé « where » est remplacé par « when ».

-WITH : lorsqu'un paramètre dans un événement abstrait disparait dans la version concrète de cet événement, il est indispensable de définir un témoin sur l'existence de ce paramètre : c'est ce qu'on appelle « witness ».

-THEN : décrit la liste des actions de l'événement.

Il est à noter que chaque événement est composé d'une ou plusieurs actions dites encore substitution. Nous expliquons, dans ce qui suit, la notion d'action ainsi que ses différentes formes.

I.4.2.4 Substitution

L'action d'un événement peut être soit *déterministe* soit *non déterministe* [27].

- Une action déterministe est de la forme :

  « liste des identificateurs des variables » := « prédicat avant-après »

En effet, elle se compose d'une liste de variables, suivie du signe « := », suivi d'une liste d'expressions.

- Pour les actions non déterministes, il existe deux formes possibles :

  (1) « liste des identificateurs des variables » :| « prédicat avant-après »

  (2) « identificateur d'une variable » :∈ « expression ensembliste »

La première forme **(1)** correspond à une liste de variables et un prédicat « *avant-après* » séparé par le signe « :| ». La deuxième **(2)** correspond à une variable suivie du signe « :∈ », suivie d'une expression ensembliste [28].

I.4.3 Obligations de preuves

Afin de garantir la correction de notre modèle, il est indispensable de le prouver. Pour y parvenir, un outil de la plateforme RODIN appelé ***générateur d'obligations de preuve*** génère automatiquement des *obligations de preuve* [29]. Une obligation de preuve définit ce que doit être prouvé pour un modèle. Il s'agit d'un prédicat dont on doit fournir une démonstration pour vérifier un critère de correction sur le modèle.

L'outil cité ci-dessus vérifie statiquement les contextes et les machines et génère des ***séquents***. « Un séquent est un nom générique pour quelque chose qu'on veut prouver » . Il est de la forme **H|G** : 'le but G est à démontrer en partant de l'ensemble H des hypothèses'.

Les règles des obligations de preuve sont au nombre de onze, et pour chacune le générateur d'obligations de preuve génère une forme spécifique du séquent.

Etant donné un élément de modélisation; un événement **evt**, un axiome **axm**, un théorème **thm**, un invariant **inv**, une garde **grd**, une action **act**, un variant ou une witness **x**. les règles d'obligation de preuve pouvant être générées pour ces éléments sont les suivantes :

INV : règle de préservation de l'invariant qui assure que chaque invariant dans une machine donnée est préservé par tous les événements. Elle est de la forme :

"evt/inv/INV" .

FIS : se rassurer qu'une action non déterministe est faisable. La forme est: *"evt/act/FIS"* .

GRD : renforcement des gardes abstraits ; les gardes des événements concrets sont plus fortes que celles des abstractions. Elle est de la forme : *"evt/grd/GRD"* .

MRG : la garde d'un événement concret qui fusionne deux événements abstraits est plus forte que la disjonction (ou logique) des gardes de ces deux événements abstraits. Le nom de cette règle est: *"evt/MRG"* .

SIM : chaque action dans un événement abstrait est correctement simulée dans le raffinement correspondant. Autrement dit, l'exécution d'un événement concret n'est pas en contradiction avec son abstraction. Le nom : *"evt/act/SIM"* .

NAT : sous une condition que les gardes d'un événement convergent ou anticipé sont vérifiées, le variant numérique proposé est un *entier naturel*. Son nom est : *"evt/NAT"* .

FIN : dans une condition où les gardes d'un événement convergent ou anticipé sont vérifiées, l'ensemble variant proposé est un ensemble *fini*. Son nom est : *"evt/FIN*

I.4.4 Raffinements

I.4.4.1 Généralités sur la relation de raffinement

Le raffinement [30] est le processus de construction d'un modèle progressivement en le rendant de plus en plus précis. Ce qui aboutit à un modèle très proche de la réalité. Il consiste à construire une séquence ordonnée de modèles où chacun est considéré comme un raffinement d'un autre modèle précédent de la séquence.

Cette technique établit un contrat entre ces deux modèles (partenaires) : un composant dit **abstrait** et un composant dit **concret**. Elle est potentiellement utilisée dans plusieurs niveaux de développement des logiciels :

- *Dans la phase de spécification :* c'est un moyen d'ajout de détails du problèmedans le développement formel. Ceci est traduit en Event-B par l'adjonction de nouvelles variables et de nouveaux événements.

- *Dans la phase de conception :* raffinement du diagramme de classes initial, formépar des classes d'analyse (classes métier), par ajout des classes de conception et puis des classes d'implémentation.

- *Dans la phase d'implémentation :* développement progressif des programmes [30] : raffinement des instructions et des données.

I.4.4.2 Obligations de preuves de raffinement

Dans une machine raffinée, le générateur d'obligations de preuves génère automatiquement des obligations de preuves relatives au raffinement. Parmi ces obligations de preuves, on cite :

-INV : utilisée pour démontrer la préservation de l'invariant du raffinement. Pour unévénement *evt* qui raffine un événement *evt0*, la règle d'obligation de preuves **INV** relative à la préservation de l'invariant pour l'événement evt est de la forme :

Axiomes et théorèmes Invariants et théorèmes abstraits Invariants et théorèmes concrets Gardes de l'événement concret prédicat témoin pour les variables Prédicat concret avant-après ⊦ Invariant spécifique modifié	**evt/inv/INV**

I.4.4.3. Types de raffinement

On peut distinguer deux types de raffinements [31] : ***le raffinement horizontal*** et ***le raffinement vertical***.

Le raffinement horizontal ou superposition: lorsqu'on a un système complexe contenant plusieurs composants et avec des transitions discrètes, on ne peut pas le spécifier d'un seul coup. On a intérêt à le spécifier progressivement pas-à-pas en commençant par un modèle très abstrait et en introduisant à chaque pas des détails issus du cahier des charges réécris conformément à la stratégie de raffinement adoptée.

Le raffinement vertical : on l'adopte lorsque toutes les étapes de raffinementhorizontal sont achevées. Il est guidé par des décisions de conception ou d'implémentation. Le raffinement vertical a pour objectif d'apporter une solution pas-à-pas à la spécification produite par le raffinement horizontal.

I.4.5 Ingénierie dirigée par les modèles et approche MDA

I.4.5.1 Les principes généraux de l'IDM

Suite à l'approche objet des années 80 et de son principe du « tout est objet », l'ingénierie du logiciel s'oriente aujourd'hui vers l'ingénierie dirigée par les modèles (IDM) et le principe du « tout est modèle » [32].

Le concept central de l'IDM est la notion de modèle pour laquelle il n'existe pas à ce jour de définition universelle. A partir des travaux de l'OMG, nous considérerons dans la suite de cette thèse la définition suivante d'un modèle.

Définition (Modèle) Un modèle est une abstraction d'un système, modélisé sous la forme d'un ensemble de faits construits dans une intention particulière. Un modèle doit pouvoir être utilisé pour répondre à des questions sur le système modélisé.

La notion de modèle dans l'IDM fait explicitement référence à la notion de langage bien défini. En effet, pour qu'un modèle soit productif, il doit pouvoir être manipulé par une machine. La définition d'un langage de modélisation a pris la forme d'un modèle, appelé méta modèle.

Définition (Méta-modèle) Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle [36], c.-à-d. le langage de modélisation.

I.4.5.2 La méta-modélisation

Définition: La méta-modélisation est l'activité consistant à définir le méta modèle d'un langage de modélisation. Elle vise donc à bien modéliser un langage, qui joue alors le rôle de système à modéliser [32].

Une méta-modélisation est nécessaire afin de comprendre le lien existant entre différents langages. La méta-modélisation permet la définition de la syntaxe dite « abstraite » d'un langage. Le produit de l'activité de méta-modélisation est habituellement appelé méta-modèle. Des standards tels que XML et MOF peuvent être utilisées.

Actuellement, plusieurs langages et outils de méta-modélisation existent, mais aucun ne cible spécifiquement le domaine de la définition des langages de points de vue ODP. La raison en est que ces langages de méta-modélisation ont la plupart du temps été développés pour concevoir et implémenter des Systèmes d'Informations et des Bases de Connaissance. Un méta-modèle pourrait aussi être utilisé pour définir tout ou une partie de la sémantique du langage [33].

I.4.5.3 L'approche MDA

Après l'acceptation du concept clé de méta-modèle comme langage de description de modèle, de nombreux méta-modèles ont émergés afin d'apporter chacun leurs spécificités dans un domaine particulier (développement logiciel, entrepôt de données, procédé de développement, etc.). Devant le danger de voir émerger indépendamment et de manière incompatible cette grande variété de méta-modèles, il y avait un besoin urgent de donner un cadre général pour leur description. La réponse logique fut donc d'offrir un langage de définition de méta-modèles qui prit lui-même la forme d'un modèle : ce fut le méta-méta-modèle MOF (Meta-Object Facility) [36]. En tant que modèle, il doit également être défini à partir d'un langage de modélisation. Pour limiter le nombre de niveaux d'abstraction, il doit alors avoir la propriété de méta-circularité, c.-à-d. la capacité de se décrire lui-même.

Définition (Méta-méta-modèle) Un méta-méta-modèle est un modèle qui décrit un langage de méta-modélisation, c.-à-d. les éléments de modélisation nécessaires à la définition des langages de modélisation. Il a de plus la capacité de se décrire lui-même.

C'est sur ces principes que se base l'organisation de la modélisation de l'OMG généralement décrite sous une forme pyramidale (cf. figure 10). Le monde réel est représenté

à la base de la pyramide (niveau M0). Les modèles représentant cette réalité constituent le niveau M1. Les méta-modèles permettant la définition de ces modèles (p. ex. UML) constituent le niveau M2. Enfin, le méta-méta-modèle, unique et méta-circulaire, est représenté au sommet de la pyramide (niveau M3).

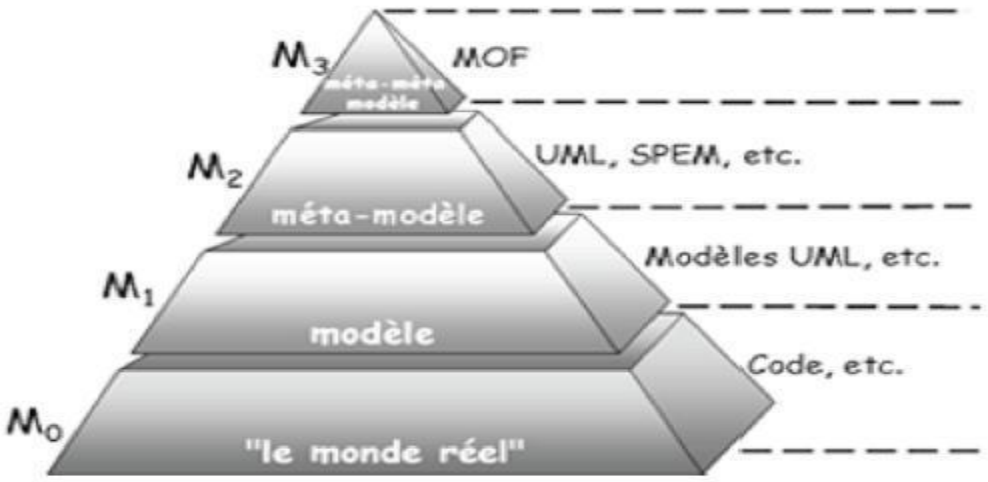


Figure 10. Pyramide de modélisation de l'OMG [32]

Le principe clé du MDA consiste à s'appuyer sur le standard UML pour décrire séparément des modèles pour les différentes phases du cycle de développement d'une application [34].

Plus précisément, le MDA préconise l'élaboration de modèles figure 11 d'exigence(Computation Independent Model – CIM) dans lesquels aucune considération informatique n'apparait, d'analyse et de conception (Platform Independent Model – PIM) et de code (Platform Specific Model – PSM).

Le passage de PIM à PSM fait intervenir des mécanismes de transformation de modèle et un modèle de description de la plateforme (Platform Description Model – PDM). Cette démarche s'organise donc selon un cycle de développement « en Y » propre au MDD (Model Driven Development) (figure 11).

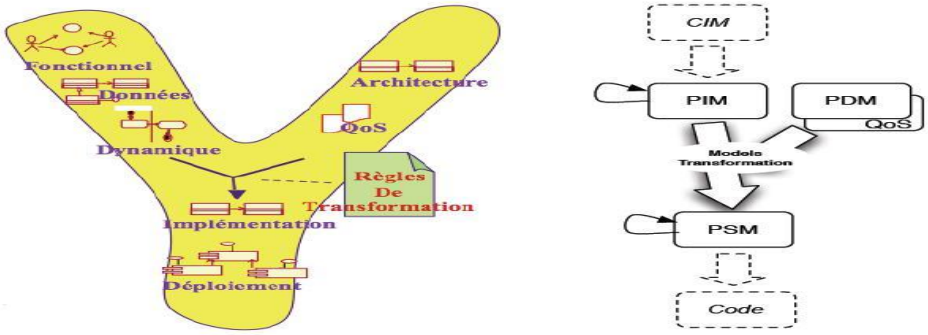


Figure 11 : MDA : Un processus en Y dirigé par les modèles [34]

I.4.5.4 Transformation des modèles

La deuxième problématique clé de l'IDM consiste à pouvoir rendre opérationnels les modèles à l'aide de transformations. Cette notion est au centre de l'approche MDA [32].

D'autre part, l'approche MDA repose sur le principe de la création d'un modèle indépendant de toute plateforme (PIM) pouvant être raffiné en un ou plusieurs modèle(s) spécifique(s) à une plateforme (PSM). Les méthodes de transformation sont là aussi indispensables pour changer de niveau d'abstraction (transformation verticale), dans le cas du passage de PIM à PSM et inversement, ou pour rester au même niveau d'abstraction (transformation horizontale) dans le cas de transformation PIM à PIM ou PSM à PSM [35].

Les règles de transformation sont appliquées au modèle source pour produire un modèle cible. Selon le standard MOF 2.0 Q/V/T, l'application des règles de transformations doit être automatisée par un moteur de transformation [36].

I.5. Conclusion

RM-ODP fournit un modèle de référence pour la spécification architecturale. Il est basé sur la séparation des préoccupations des enjeux en utilisant les cinq points de vue ODP : Entreprise, Information, Traitement, Ingénierie et Technique. Ce modèle traite le problème de la répartition en tenant compte de l'hétérogénéité des systèmes et des organismes.

Cependant, il n'existe pas de méthodologie concrète pour guider les concepteurs des systèmes mécatroniques. Il est donc nécessaire de définir une méthodologie et des outils associés pour pouvoir mener des spécifications dans le cadre du modèle de référence RM-ODP.

La méthode B est une méthode formelle qui couvre toutes les étapes de développement d'un système, de la spécification jusqu'à la réalisation, par une succession d'étapes de raffinement. Elle se présente comme une méthode de développement formelle mettant en valeur aussi bien les aspects statiques que dynamiques d'un système, en particulier un système ODP. D'un côté, les aspects statiques sont définis au moyen de concepts mathématiques de la théorie des ensembles et de la logique des prédicats, et d'un autre côté, les aspects dynamiques sont modélisés en utilisant le langage des substitutions généralisées.

CHAPITRE II : LA SPECIFICATION D'UN SYSTEME MECATRONIQUE AU POINT DE VUE ENTREPRISE

II.1 Introduction

RM-ODP fournit un modèle de référence pour la spécification architecturale. Il est basé sur la séparation des préoccupations des enjeux en utilisant les cinq points de vue ODP : Entreprise, Information, Traitement, Ingénierie, Technique. Ce modèle traite le problème de la répartition en tenant compte de l'hétérogénéité des systèmes distribués ouverts.

Nous nous intéressons particulièrement à la spécification des systèmes mécatroniques de point de vue entreprise, qui se concentre sur les rôles, les objectifs, le domaine d'application et les politiques du système. Bien que le modèle de référence ODP fournit des langages pertinents. Il est cependant très difficile à appréhender non pas à cause de sa longueur et de ses détails mais parce qu'il est très abstrait. Il constitue une méta-norme donc non applicable directement. En effet il n'a défini ni quand ni comment produire les spécifications des différents points de vue.

Dans ce contexte, nous définissons un méta-modèle des concepts de base et de leurs relations en utilisant le langage de modélisation unifié UML. Ainsi, nous fournissons un guide méthodologique de développement de système mécatronique permettant à un constructeur de spécifier son système, avec la notation UML selon une sémantique ODP.

Le présent chapitre est organisé de la façon suivante : Dans la section 2, nous présentons le système de freinage que nous avons utilisé pour illustrer notre approche de modélisation. Dans la section 3, nous avons défini l'ensemble de concepts et notations pour la spécification d'un système mécatronique de point de vue entreprise. Dans la section 4, nous proposons d'utiliser un sous-ensemble de l'UML comme une notation concrète pour modéliser un système mécatronique au point de vue entreprise.

II.2. Cas d’étude : Système de freinage

L'alliage à mémoire de forme magnétique FSMA est un matériau qui possède la capacité de produire des contraintes et des déformations. Lorsque le matériau est exposé à un champ magnétique, il produira des déformations et sera de retour à l'état initial en l’absence de ce champ. La fonction linéaire entre le champ magnétique et les déformations produites rend ce matériau facile à contrôler pour toute application. Ce matériau a le potentiel de remplacer des actionneurs classiques tels que des actionneurs à commande pneumatique, à commande hydraulique et électromagnétique.

Dans ce chapitre, nous avons modélisé un système de freinage destiné au robot, qui utilise l’alliage à mémoire de forme magnétique comme actionneur de frein. L'application a une exigence stricte en termes de poids, la taille et la simplicité de sa conception. En utilisant l'alliage à mémoire de forme magnétique toutes ces exigences strictes peuvent être réalisées.

Le système peut être décomposé en un ensemble de sous composants interconnectés. Le composant contrôleur représente le noyau du système. Il modélise le calculateur électronique où s’exécutent les commandes de freinage. Ce composant est lié à l’encodeur, au moteur, à la génératrice et à l’actionneur du frein. Il effectue des calculs selon le principe de fonctionnement du système et de ses lois spécifiques de commande, et produit des valeurs de contrôle en sortie qui affectent implicitement le frein au niveau de son actionneur. La figure ci-dessous illustre le diagramme des cas d’utilisation qui représente l’ensemble des fonctions de ce système.

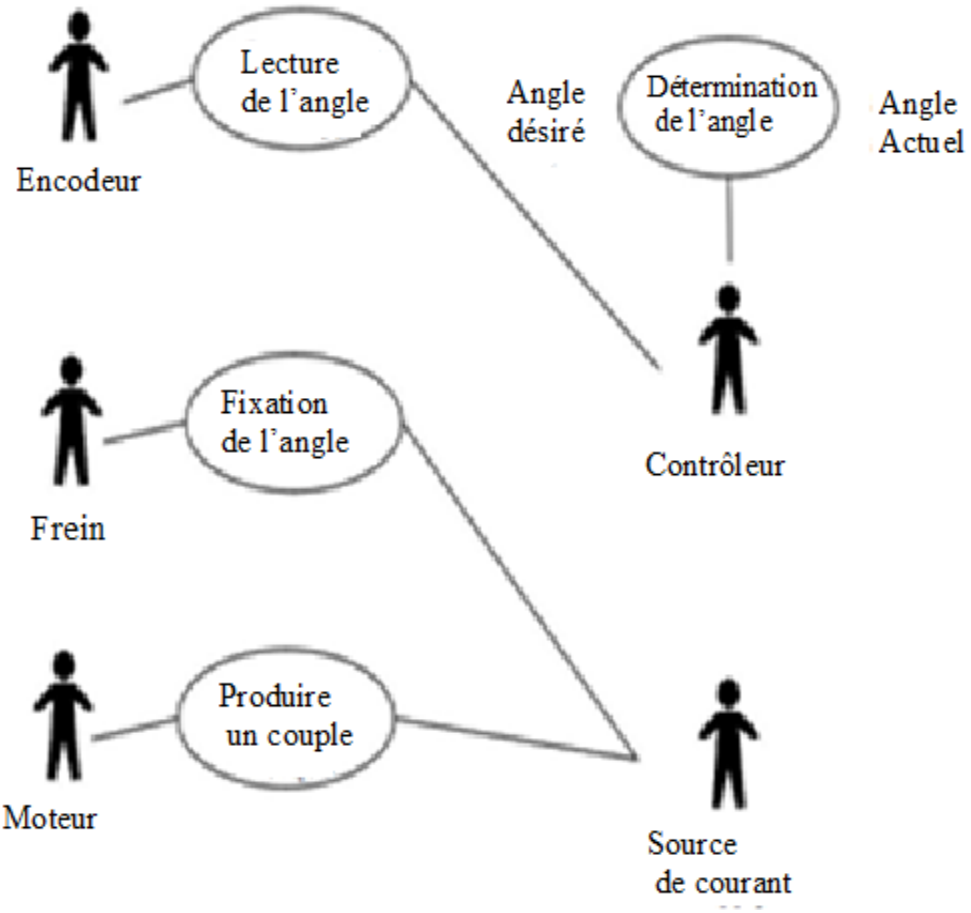


Figure 12 : Diagramme de cas d'utilisation du système de freinage

II.3. Spécification d'un système mécatronique au point de vue Entreprise

II.3.1. Point de vue Entreprise

Le point de vue Entreprise définit une vision conceptuelle de l'architecture étudiée. Il décrit les entités de l'entreprise, c'est-à-dire les communautés et les rôles, les comportements et les interactions entre ces derniers, les acteurs, les processus, les règles de gestion, les droits et les obligations, ce que l'on appelle de façon générale la politique de l'entreprise. Il fournit une spécification du système dans l'environnement avec lequel il interagit. Cette spécification s'intéresse plus spécialement aux objectifs et à la politique du système.

II.3.2. Le langage d'entreprise

Il s'utilise pour définir les objectifs [37], le domaine d'application et les politiques d'un système mécatronique. Le système mécatronique et l'environnement dans lequel il opère sont représentés par un ou plusieurs objets d'entreprise regroupés en une communauté d'entreprise liés par un contrat et par les rôles joués par ces objets. La finalité d'une spécification

d’entreprise est d’identifier les actions requises par les politiques gouvernant la communauté spécifiée. Ces politiques fixent le cadre de l’application du contrat de la communauté en définissant les actions permises, obligatoires ou interdites à un objet.

II.3.2.1. Communautés

Une communauté est une configuration d'objets d'entreprise qui est formée pour atteindre un objectif. Les objets dans un modèle d'entreprise sont des abstractions des personnes, des systèmes et d'autres entités qui influencent l'objectif de la communauté. Un objet d'entreprise peut être affiné en tant que communauté à un niveau plus concret. Dans notre exemple, la communauté est le système de freinage; son objectif est de produire un freinage. Les objets d'entreprise qui travaillent ensemble pour atteindre cet objectif sont les suivants: frein, moteur, Encodeur, actionneur et le contrôleur. Les règles de structuration pour la communauté du système de freinage sont représentées dans le schéma de la figure 13.

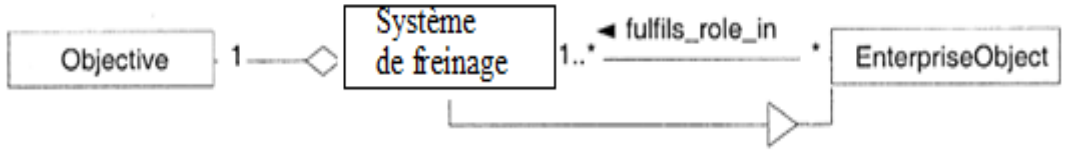


Figure 13 : Les concepts de la communauté du système de freinage

II.3.2.2. Spécification des communautés

Les communautés sont précisées par un contrat ou une communauté modèle [42], qui identifie les différents rôles que les objets jeu dans la communauté et les politiques qui régissent le comportement de ces objets lorsqu'ils remplissent leur rôle dans la communauté. La figure 14 ci-dessous montre comment interpréter les règles de structuration des spécifications communautaires en langage d’entreprise. La Communauté est un modèle de contrat spécifiant la structure et le comportement de la communauté dans la définition des rôles de la communauté et de leurs associations. Un rôle identifie un comportement. La connexion entre ces associations montre les interactions qui peuvent exister entre les rôles.

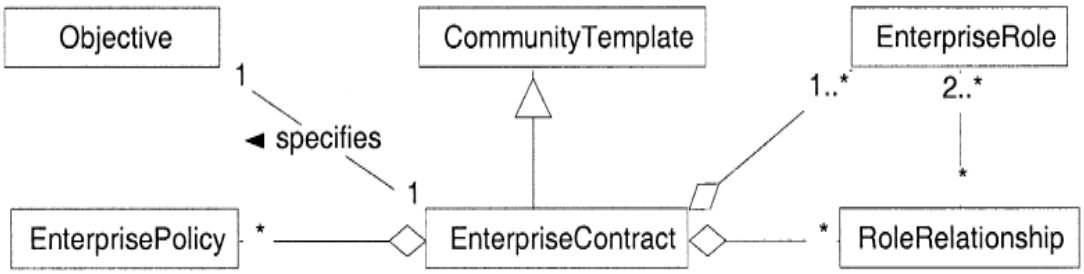


Figure 14 : Le contrat communautaire

II.3.2.3. Les rôles d'entreprise

Il existe quatre sous-catégories de rôles qui sont d'intérêt pour les spécifications de l'entreprise : les rôles acteurs, les rôles outils, les rôles principaux et des rôles non-principaux. Actuellement, il n'est pas explicite que ces classifications sont exclusives et totales. Ainsi, tous les rôles peuvent être classés comme acteur principal, outil principal, acteur non-principal, ou outil non-principal. Cette classification est représentée sur la Figure 15. Un rôle d'acteur est défini comme " un rôle avec un comportement significatif qu'il initie au moins une interaction ". Un rôle outil est défini comme " un rôle pour les objets qui sont référencés par certains comportements de la communauté, et n'initient aucune interaction dans cette communauté ".Un objet d'entreprise remplissant un rôle d'acteur participe aux actions de ce rôle, alors qu'un objet entreprise remplissant un rôle d'outil apparaît uniquement dans une spécification d'entreprise.

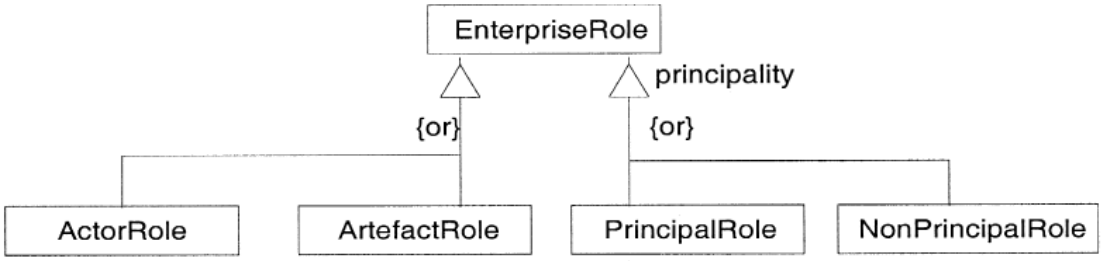


Figure 15 : les rôles d'Enterprise

II.3.2.4. Les relations entre les rôles

Le concept de relation entre rôle n'est pas explicitement défini dans la version actuelle du langage de point de vue entreprise [43]. On représente dans ce qui suit une interprétation possible de ce concept. Il y a au moins deux façons différentes dans lesquelles les rôles

peuvent être liés. Tout d'abord, les rôles peuvent être liés, car ils interagissent. Dans une communauté de freinage, par exemple, le rôle de l'encodeur et le rôle du contrôleur sont liés parce qu'ils partagent l'information sur l'angle. Ces relations peuvent généralement être laissées implicites. Cependant, il existe d'autres relations entre les rôles qui doivent être explicitées à des fins de modélisation.

II.3.2.5. Instanciation

Les Communautés sont créées par instanciation du modèle de contrat correspondant. Instanciation d'un contrat comporte l'attribution d'objets d'entreprise à des rôles. En général, un objet peut remplir plusieurs rôles figure 16, dans un certain nombre de communautés. Par exemple, le contrôleur et le processeur peuvent remplir un rôle dans un système de communauté suspension et également remplir un rôle de déterminer l'angle pour la communauté de freinage.

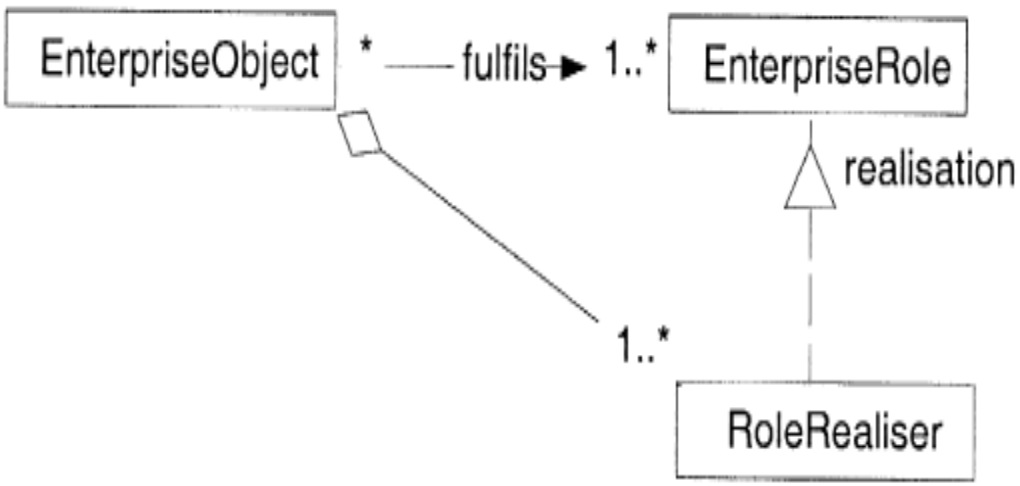


Figure 16 : la réalisation du rôle

II.3.2.6. Le comportement de l'entreprise

Les rôles sont utilisés pour identifier le comportement des objets d'entreprise dans la communauté. À cette fin, les actions sont associées à des rôles (noter que les actions peuvent, en effet, être associées à plus d'un objet). En fait, toutes les actions [entreprise] sont identifiées par au moins un rôle d'acteur (l'initiateur). Puisque le comportement est défini comme étant " une collection d'actions avec un ensemble de contraintes quand ils peuvent se produire, " une définition de rôle devrait également spécifier ces contraintes. Ainsi, un rôle

spécifie le comportement qu'un objet. Notez que la notion de contrainte comportementale n'est pas définie dans la RM-ODP. La raison de ceci est que le type de contraintes qui peut être spécifiée dépend beaucoup de langage de spécification qui est utilisé. Dans un langage de spécification comme LOTOS, par exemple, les contraintes sur le comportement sont spécifiées en termes d'un ordre temporel des actions, qui peuvent donc être composés avec un opérateur de synchronisation. Dans un langage de spécification à base de modèles, tels que Z, le comportement est contraint, entre autres, par le biais des invariants sur un espace d'état. La figue 17 représente une interprétation possible des clauses relatives au comportement de rôle pour le système de freinage.

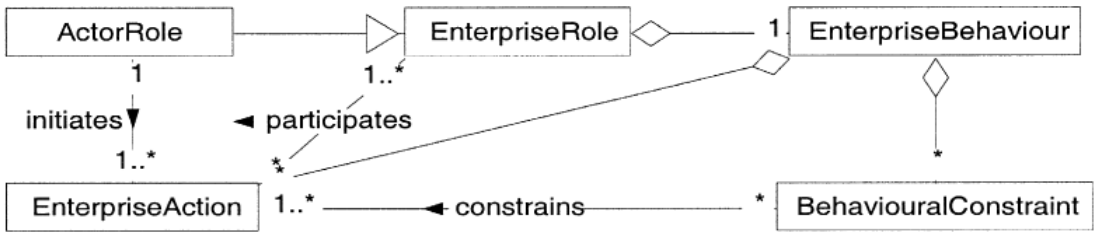


Figure 17 : Clauses relatives au comportement de rôle

Un rôle de l'entreprise définit un certain comportement de l'entreprise, ce qui consiste en un ensemble d'actions de l'entreprise et un ensemble de contraintes sur ces actions. Au moins un rôle d'entreprise participe à chaque action de l'entreprise. En outre, chaque action de l'entreprise est initiée précisément par un rôle de l'acteur. Les actions qui impliquent plus d'un rôle sont appelées interactions.

II.3.2.7. Les stratégies d'entreprise

Les politiques contraignent le comportement des objets d'entreprise qui remplissent les rôles des acteurs dans les communautés [61]. Le concept de la politique est étroitement lié à celle du comportement de l'entreprise : les deux représentent des contraintes sur le comportement des objets. Ils ne sont pas les mêmes. Une spécification du comportement de l'entreprise définit le comportement physiquement possible des rôles dans une communauté. Une spécification de politique peut donc limiter ce comportement plus loin afin d'atteindre les objectifs de la communauté. Ainsi, en général, une spécification d'entreprise inclura une description du comportement de l'entreprise avec une spécification de la politique. Une autre façon de voir la différence est de penser à des politiques comme décrivant le comportement idéal et souhaitable au sein d'une entreprise, alors que le comportement de l'entreprise est un modèle du comportement réel. Ce dernier peut ou non conformer à l'idéal exprimé dans les

politiques. Les spécifications de la politique, par conséquent, contiennent souvent des prescriptions de ce qu'il faut faire dans le cas où une règle est violée.

Les politiques sont conçues pour répondre à l'objectif de la communauté. Les politiques sont précisées par le contrat communautaire. Une politique est " un ensemble de règles relatives à un usage particulier " et que " une règle peut être exprimée comme une obligation, une autorisation ou une interdiction ". La structure des politiques est représentée sur la Figure 18.

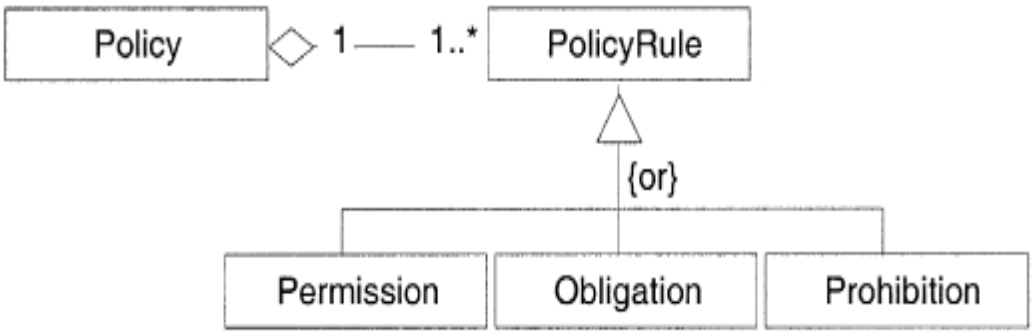


Figure 18 : Structure de la politique

II.3.2.4. Résumé

La figue 19 représente un résumé du langage ODP de point de vue entreprise en mécatronique donné comme un méta-modèle. Nous avons utilisé l'approche méta-modélisation pour définir le passage entre la spécification ODP Enterprise et la spécification mécatronique Enterprise. Une façon de faire est de trouver à la fois la structure fonctionnelle et la structure de mise en œuvre et concernent les objets liés les uns aux autres. Le tableau 1 présente un aperçu sur le passage de la spécification Enterprise ODP à la spécification Mécatronique Enterprise.

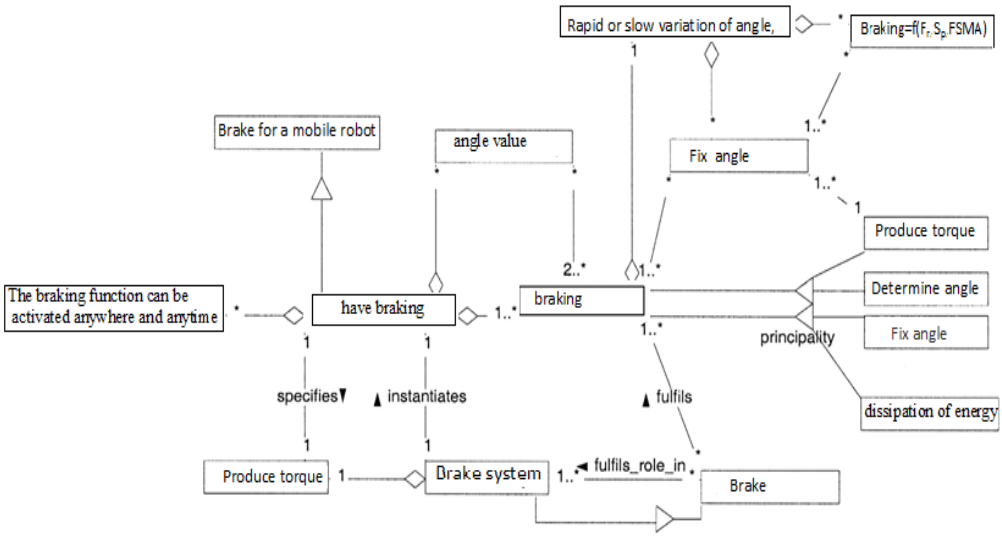


Figure 19 : la spécification du point de vue entreprise en mécatronique

II.4. Utilisation de l'UML pour spécifier le point de vue de l'Entreprise

Le langage du point de vue entreprise est un langage abstrait dans le sens où il ne prescrit pas l'utilisation d'une notation particulière. Il se contente d'identifier un ensemble de concepts et un ensemble de règles de structuration. Ceux-ci ont été capturés dans la méta-modèle présenté ci-dessus, qui définit effectivement une syntaxe abstraite pour le langage. Dans cette section, nous examinons comment l'UML peut être utilisé pour fournir une syntaxe concrète pour le langage de point de vue entreprise [62].

L'UML rassemble un certain nombre d'objets techniques largement utilisés, tels que les diagrammes de classes, le cas d'utilisation des diagrammes, les diagrammes d'états et les diagrammes de séquence dans un seul langage. Le but de cette section est d'étudier comment ces techniques de schématisation peuvent être utilisées pour représenter les spécifications du point de vue entreprise en mécatronique. L'utilisation de l'UML [44] pour la spécification de l'entreprise a l'avantage que beaucoup de gens sont déjà familiers avec, et peuvent lire les spécifications de l'entreprise, sans beaucoup d'instructions supplémentaires. L'inconvénient d'utiliser UML est que le langage est encore en évolution et n'a pas de signification.

Au niveau structurel (par exemple, lors de la définition des communautés), Les avantages l'emportent sur les inconvénients. Cependant, en ce qui concerne la spécification de la politique, où un sens précis est crucial, il est nécessaire d'utiliser une notation avec une sémantique définie.

II.4.1. Communauté du système de freinage

Dans le langage UML, une communauté peut être représentée par un ensemble, Son objectif est spécifier en langage naturel, il est attaché à la communauté dans un champ de l'ensemble de la description.

Une communauté peut être représentée par un paquet voir figure 20.

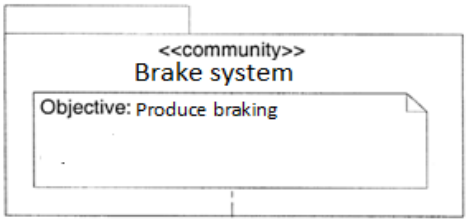


Figure 20 : La communauté du système de freinage et son objectif.

II.4.2. Modélisation des rôles et leurs relations.

La classe du contrôleur représentée sur la figure 21, est un exemple d'une relation conceptuelle entre les rôles du contrôleur et de l'actionneur du frein [52]. A Chaque fois qu'il y'a une fixation d'angle, une instance de cette relation est établie. La relation sera à nouveau dissoute sur le retour à la position initiale. L'avantage de la modélisation explicite de telles relations conceptuelles entre les rôles que nous pouvons leur joindre certains attributs de la relation. Dans le cas d'un freinage, nous nous intéressons à la lecture de l'angle et la force de frottement. La structure de la communauté du système de freinage est illustrée sur la figure 21.

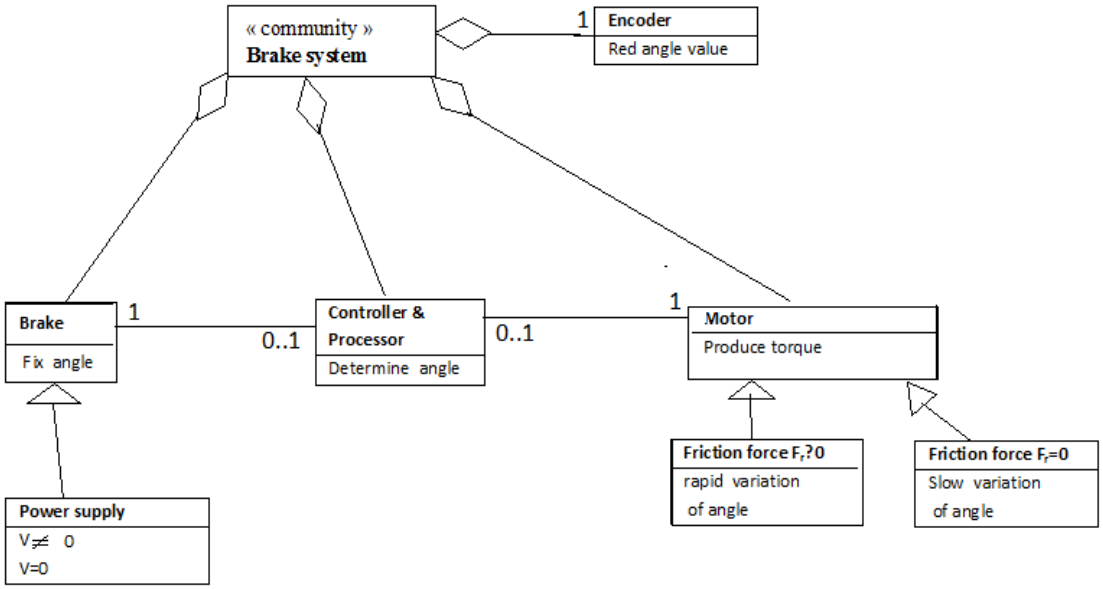


Figure 21 : La structure de la communauté du système de freinage

II.4.3. Résumé des correspondances

Le tableau 1 résume comment l'UML peut être utilisé comme une notation concrète pour le point de vue entreprise en mécatroniques. Il décrit la correspondance des concepts du langage d'entreprise en mécatroniques et l'UML constructions.

TABLEAU 1: Correspondance entre langage d'entreprise mécatroniques et UML construction

langage d'entreprise en mécatroniques	**UML CONSTRUCTIONS**
Community:Brake system	Package
Enterprise object: Brake, processor, motor, power	class
Enterprise role : fix angle, Produce torque	class
Rolerelationship : angle value	Class
Action : convert electrical energy to magnetic energy	Use case

II.5. Conclusion

Nous avons utilisé le standard UML pour construire un méta-modèle pour le langage d'entreprise en mécatronique. Il nous a permis de représenter les concepts de base de point de vue entreprise et leurs relations d'une manière claire, concise et cohérente. Ce qui permet la communication et le partage du savoir faire entre les acteurs ayant des connaissances variés. Nous avons utilisé l'UML pour définir la structure statique des communautés, à savoir, les rôles et leurs relations. Ce meta-modèle du langage d'entreprise pour la mécatronique permet aux concepteurs de formaliser leur exigences sur le comportement des objets d'entreprise qui remplissent des rôles dans les communautés. De toute évidence, il y a beaucoup d'autres aspects, tels que la qualité du service, fiabilité et la sécurité, qui pourrait également être soumis à la spécification de la politique.

CHAPITRE III : SPECIFICATION FORMELLE DES SYSTEMES MECATRONIQUES

III.1. Introduction :

Avec la croissance rapide de la technologie, les fonctionnalités proposées par les constructeurs des systèmes mécatroniques sont devenus de plus en plus complexes, de tels systèmes dépendent de plusieurs calculateurs, capteurs et actionneurs répartis sur des réseaux informatiques internes. La complexité de ces systèmes les rend sensibles aux erreurs produites durant les phases de spéciation, de conception, de réalisation, de vérification et de validation. Une erreur produite lors des phases initiales a un prix plus élevé que celle produite lors des phases ultérieures [45]. La garantie de sûreté de fonctionnement, de fiabilité, devient alors essentielle dans le développement des systèmes mécatroniques [46]. L'approche classique n'apporte pas des guides méthodologiques favorisant l'écriture des cahiers des charges [47], les risques d'erreurs sont à toutes les phases et ne se détectent qu'aux phases finales et leur détection nécessite des tests très poussés [48].

Aujourd'hui, pour répondre aux enjeux de sécurité imposés par le marché, une nouvelle approche de conception des systèmes est nécessaire pour permettre l'intégration des différentes technologies sûres de fonctionnement dès la première phase de développement. La méthode formelle (Event-B) [3] apparaît comme la méthodologie la plus adaptée au développement des systèmes mécatroniques.

Les méthodes formelles, les techniques de test et l'interprétation abstraite favorisent le développement des systèmes corrects [3]. Les processus de développement formels organisés autour des méthodes formelles comme Event-B apportent des réponses pertinentes à la problématique de construction des systèmes complexes. De tels processus permettent de combattre les erreurs très tôt dès les phases initiales. En outre, grâce à la technique de raffinement [31], Event-B permettent l'élaboration incrémentale d'une spécification abstraite cohérente et concrétiser pas à pas cette spécification abstraite jusqu'à la génération de code. La plate forme RODIN est un environnement qui supporte l'Event-B tels que : générateur des obligations de preuves, prouveur automatique et interactif.

Ce chapitre est organisé comme suit : dans la section 2, nous présentons et analysons l'approche classique de développement des systèmes mécatroniques. Dans la section 3, nous présentons et analysons l'approche formelle (Event-B). Dans la section 4, nous avons développé une application sur un système ABS par la méthode formelle (Event-B).

III.2. Approche classique

L'approche classique pour le développement des systèmes mécatroniques comporte plusieurs phases : l'écriture du cahier des charges, spécification, conception, codage et intégration. Cette approche présente plusieurs insuffisances :

- Les cahiers des charges utilisés par l'approche classique sont très souvent difficiles à exploiter. Ils présentent des mécanismes de réalisation au détriment de **l'explicitation des propriétés** du futur système. D'ailleurs, l'approche classique n'apporte pas des guides méthodologiques favorisant l'écriture des cahiers des charges de qualité [6].
- Les risques d'erreurs sont à toutes les phases de l'approche classique. En outre, ces erreurs sont souvent uniquement détectées dans les **phases** finales après la phase de codage et d'intégration.
- La détection des erreurs nécessite des tests très poussés : test unitaire, d'intégration et système [7]. Ces tests sont très coûteux et ne garantissent pas **la correction** du logiciel développé.

III.3. Approche formelle

L'approche formelle est utilisée pour raisonner sur des systèmes complexes, distribués ou réactifs. La sémantique d'un modèle est donnée par les obligations de preuve qui sont générées pour montrer sa cohérence. Cette approche est basée sur des méthodes formelles. Les deux éléments fondamentaux d'une méthode formelle sont : langage formel et processus formel de développement. Un langage formel est une notation mathématique **bien définie** aussi bien sur le plan syntaxique que sémantique. Un processus formel de développement permet le passage pas-à-pas de l'abstrait vers le concret en garantissant une implémentation **correcte par construction** vis-à-vis de sa spécification moyennant une activité de **preuve**.
A titre d'exemple B est une méthode formelle avec preuves englobant un langage logico-ensembliste et de substitutions généralisées et un processus formel de développement basé sur la technique de raffinement. Egalement, Event-B est une méthode formelle avec preuves.

L'approche formelle pour le développement des systèmes comporte les phases suivantes : réécriture du cahier des charges, spécification abstraite, raffinement horizontal, raffinement vertical et génération de code. L'enchaînement de ces phases est illustré par la figure 22.

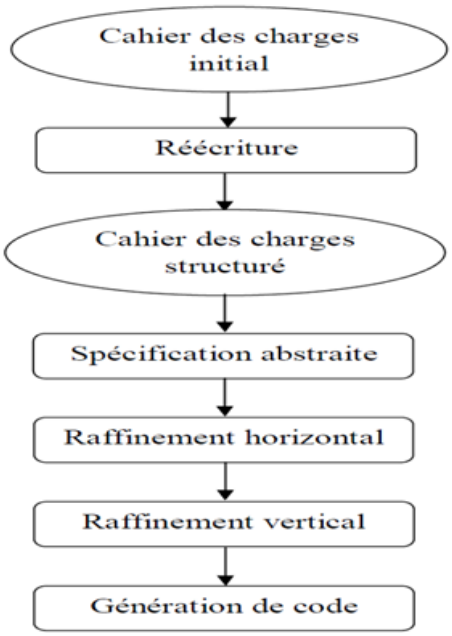


Figure 22 : Approche formelle pour le développement des systèmes

La réécriture du cahier des charges a pour objectif de réécrire le cahier des charges initial de façon à mettre en évidence les propriétés du système à développer. En effet, les cahiers des charges tels qu'ils sont élaborés aujourd'hui ne se prêtent pas bien à l'utilisation des méthodes formelles. La phase de spécification abstraite produit un modèle initial très abstrait. Celui-ci est raffiné pas-à-pas en tenant compte des propriétés explicitées par le cahier des charges de qualité. Le dernier modèle issu de la phase de raffinement horizontal intègre toutes les propriétés et tous les comportements du système à développer. Ceci constitue la spécification du système. La phase de raffinement vertical a pour objectif d'apporter une implémentation pas-à-pas à la spécification fournie par le modèle issu de la phase de raffinement horizontal. Ainsi, une succession de modèles de moins en moins abstraits est produite lors de cette phase. L'ultime modèle issu de la phase de raffinement vertical est transformé en code par la phase de génération de code.

L'approche formelle de développement des systèmes est supportée par une activité de preuve mathématique outillée (générateur d'obligations de preuves, prouveur, animateur et

model-checker) afin de vérifier la cohérence des modèles produits et la correction de chaque étape de raffinement aussi bien pour le raffinement horizontal que vertical.

L'approche classique peine à développer des systèmes corrects. Tandis que, l'approche formelle permet l'obtention des systèmes corrects par construction. En effet, chaque modèle produit par cette approche est soumis à des preuves. Si une preuve échoue, alors le modèle est revu et corrigé. Puis, on crée un modèle plus affiné, auquel on applique des preuves. Et ainsi de suites jusqu'au modèle ultime transformé automatiquement en code grâce au générateur de code. En outre, contrairement à l'approche classique, la phase des tests est très simplifiée dans l'approche formelle. Enfin, les deux approches ont globalement un coût équivalent [49]. Dans l'approche formelle, le temps passé à élaborer les modèles successifs est rattrapé lors de la phase des tests qui prend très peu de temps, contrairement à l'approche classique [3].

III.4. Spécification en Event-B d'un système (ABS)

Dans cette partie, nous comptons établir une spécification formelle en Event-B du Système ABS (Anti-lock Brake System).

III.4.1. Principe de fonctionnement du système ABS

Le rôle du système antiblocage ABS est d'empêcher le blocage des roues en cas de freinage brusque ou freinage sur un revêtement glissant en permettant au conducteur de conserver le contrôle et la stabilité de son véhicule. Il permet aussi de réduire les distances de freinage particulièrement sur une route mouillée. Pour ce faire, le calculateur électronique (voir figure 23) déclenche un cycle de contrôle, dès que le véhicule roule à une vitesse supérieure à 8 km/h, pour vérifier entre autre les capteurs de vitesse des roues et le groupe hydraulique. Ce cycle de contrôle est très court, il s'effectue en une fraction de seconde pour surveiller en permanence tous les composants nécessaires au bon fonctionnement du système ABS. Lorsque le conducteur appuie sur la pédale de frein, le maître cylindre de frein transforme ce mouvement en pression hydraulique. Cette pression se diffuse alors par l'intermédiaire d'un liquide incompressible jusqu'aux roues pour les bloquer dans le but de stopper ou réduire la vitesse du véhicule.

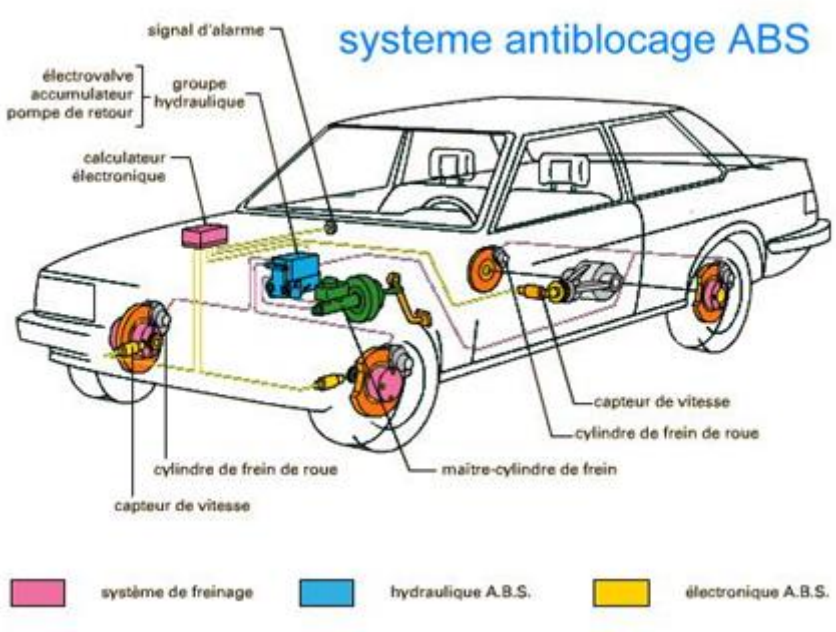


Figure 23 : Composants du système antiblocage ABS dans une voiture

Cependant, dés que les roues ne tournent plus mais glissent sur le sol, le ralentissement n'est plus contrôlé et perd de son efficacité, puisqu'il ne dépend que des frottements entre les roues et le sol. Dès lors, le système ABS entre en action pour empêcher le blocage des roues. Pour ce faire, la vitesse de rotation de chaque roue est mesurée par des capteurs de vitesse. Elle est transmise en permanence au système de contrôle de l'ABS via des impulsions électriques.

Dès qu'une roue amorce un blocage, le système d'antiblocage ABS réduit momentanément la pression hydraulique des étriers de freins. Il diminue ainsi la pression hydraulique des étriers de freins pour les relâchés et les roues sont libérées. Dès que la roue commence à reprendre de la vitesse, le système ABS augmente temporairement la pression hydraulique des étriers de freins. Le système ABS répète ce cycle autant de fois qu'il est nécessaire pour assurer un freinage efficace.

III.4.2. Description informelle du système ABS

On sait depuis longtemps que la modélisation d'un système complexe ne peut s'accomplir en une seule étape. Dans la pratique la démarche suivie lors d'un développement de système est incrémentale et progressive. On part d'une description informelle puis peu à peu, on construit une spécification par adjonction de nouveaux éléments. Il est possible dans cette démarche de réutiliser des éléments existants ou encore d'effectuer des retours arrière afin d'intégrer les erreurs ou les oublis de spécification.

Le système ABS peut être décomposé en un ensemble de sous composants interconnectés. Le composant controleur_abs de catégorie system représente le noyau du système ABS. Il modélise le calculateur électronique où s'exécutent les commandes de freinage. Ce composant est lié en entrée à des capteurs, modélisés par des composants de la catégorie device, pour détecter les états du moteur (capteur_moteur), de la pédale de frein (capteur_pedale), et de la vitesse des roues (capteur_roue). Il effectue des calculs selon le principe de fonctionnement du système ABS et ses lois spécifiques de commande, et produit des valeurs de contrôle en sortie qui affectent le groupe hydraulique au niveau du composant actionneur_frein et l'affichage de l'état du système ABS au niveau du composant temoin_abs.

III.4.2.1. Composants de base du system_ABS :

Les Composants d'un système ABS:

- ✓ Contrôleur abs;
- ✓ capteur moteur ;
- ✓ capteur pédale ;
- ✓ capteur roue ;
- ✓ capteur vitesse ;
- ✓ témoin abs ;
- ✓ actionneur frein.

III.4.2.2. Les commandes de base et boutons :

Les commandes suivantes peuvent être effectuées au moyen de la pédale et du bouton B.

- ➢ Command 1 : Démarrer moteur, par le bouton B
- ➢ Command 2 : Freiner en appuyant sur la pédale de frein

III.4.2.3. Première vue schématique du système

Ce modèle initial formalise la connexion direct du dispositif de commande au système de freinage comme le montre la figure ci-dessous:

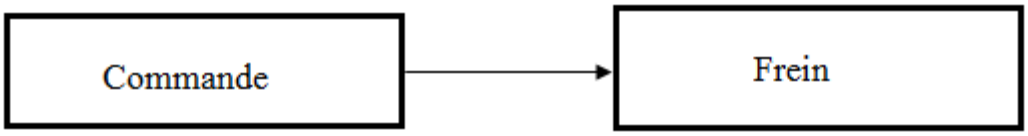


Figure 24 : Première vue schématique du système

III.4.2.4. Session de freinage

Nous supposons qu'initialement le moteur est en arrêt :

1- Démarrer le moteur en appuyant sur le Button B (command 1)

2- Freiner en appuyant sur la pédale du frein (command 2);

3- l'état du moteur est donné par le capteur moteur (capteur_moteur) (prêt ou pas) ;

4- l'état de la pédale est donné par le capteur pédale (capteur_pedal)(engage ou désengage) ;

5- la vitesse des roues est donnée par le capteur_roue (vitesse_r) ;

6- la vitesse linéaire est donnée par le capteur_vitesse (vitesse_v) ;

7- le contrôleur_ABS effectue des calculs suivant les lois spécifiques est envoie des valeurs de contrôles en sortie (prêt ou pas) ;

8-témoin ABS affiche état ABS ;

9- actionner_frein ;

10-Répéter 3 to 9.

III.4.2.5. Nécessité d'un contrôleur

Si on appuie sur la pédale du frein, le maître cylindre du frein transforme ce mouvement en pression hydraulique. Cette pression se diffuse alors par l'intermédiaire d'un liquide incompressible jusqu'aux roues pour les bloquer dans le but de stopper le véhicule. Cependant, dès que les roues ne tournent plus mais glissent sur le sol, le ralentissement n'est plus contrôlé et perd de son efficacité, puisqu'il ne dépend que des frottements entre les roues et le sol. En conséquence, un contrôleur est placé entre les commandes et les équipements afin de faire en sorte que les choses fonctionnent correctement. Afin d'éviter tout dysfonctionnement, les équipements signalent également leur propre statut au contrôleur. Tout cela se traduit par la deuxième vue schématique du système, représenté sur la figure 25.

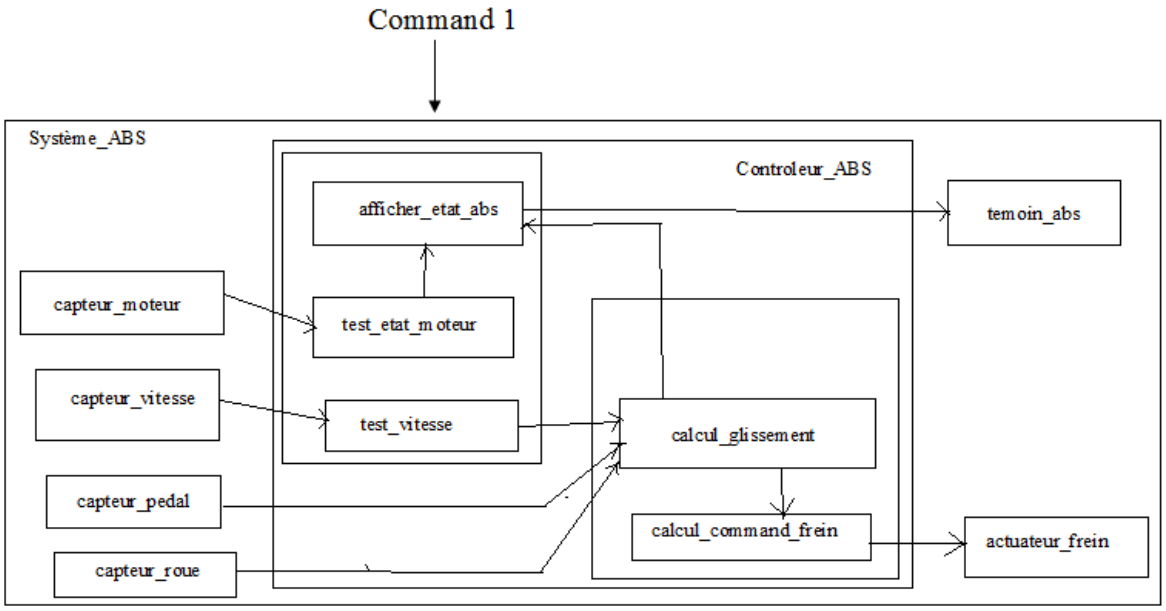


Figure 25 : Deuxième vue schématique du système

III.4.3. Modèles de conception

Un patron de conception est un ensemble de variables, d'événements et d'invariants qui vont permettre de modéliser une notion que l'on veut voir figurer dans le système que l'on construit. Un patron de conception aura aussi recours à des constantes et des propriétés relatives à ces constantes. Ces éléments de modèle seront directement insérés dans le modèle du système en cours de construction, après instanciation des éléments du patron de conception variables,(invariants, constantes propriétés).

Un patron de conception est prouvé lorsque les événements décrits dans ce patron vérifient les propriétés invariantes exprimées sur ces variables. De ce fait, n'importe quelle instanciation correcte du patron de conception va conduire à la production d'éléments de modèle dont la correction est acquise. Dans les faits, la démonstration de correction du modèle produit est la charge des outils de preuve, mais nous savons de fait que le modèle à démontrer est juste, ce qui est un gain significatif.

Cette approche de modélisation par utilisation de patrons de conception prouvés se distingue de l'approche faisant appel au raffinement automatique décrite précédemment. En effet, le raffinement automatique est un processus itératif, majoritairement automatisé, qui utilise un moteur de règles et des règles de raffinement « génériques». Ce moteur de règles ne réalise pas de vérification de correction des règles qu'il tente d'appliquer. Une règle de

raffinement peut aussi être mal adaptée au modèle en cours de traitement et nécessiter une réécriture spécifique, avant toute application. La validité du modèle obtenue sera déterminée lors de la phase de preuve du modèle. Cette validité peut être impossible à obtenir du fait de l'inadéquation/fausseté des règles de raffinement appliquées. Il va falloir alors revenir sur le processus de raffinement, identifier la ou les règles en cause, créer de nouvelles règles de raffinement puis relancer le processus de raffinement automatique. Cette approche est clairement moins efficace que l'approche par patrons de conception prouvés, où l'on sait de fait que les éléments de modèle produits sont corrects et peuvent être démontrés par preuve.

Nous produisons ci-dessous quelques exemples de patrons de conception prouvés qui ont été élaborés [6].

III.4.3.1. Synchronisation faible :

Il s'agit d'un patron simple d'action et réaction simple sans rétroaction qui est présenté sur le figure 26. Nous supposons avoir une action **a**, représentée par un trait continu, suivie d'une réaction notée **r**, représentée par un trait pointillé

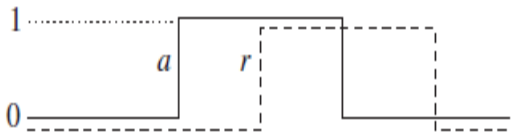


Figure 26 : Action _ Réaction

Nous notons que **r** survient après **a**. En d'autres termes, **r** passe à l'état haut après que **a** y soit passé. De manière similaire, **r** passe à l'état bas une fois que **a** soit passé à l'état bas.

Il est possible qu'une fois que **r** est à l'état haut, **a** passe successivement à l'état bas puis à l'état haut, éventuellement plusieurs fois, alors que **r** n'est pas capable de réagir à ces variations rapides : **r** reste à l'état haut tout le temps. La figure 27 montre une faible synchronisation entre l'action **a** et la réaction **r.**

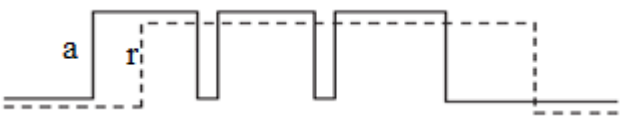
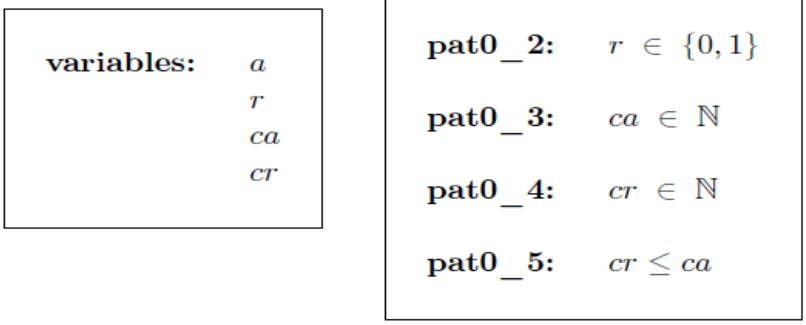


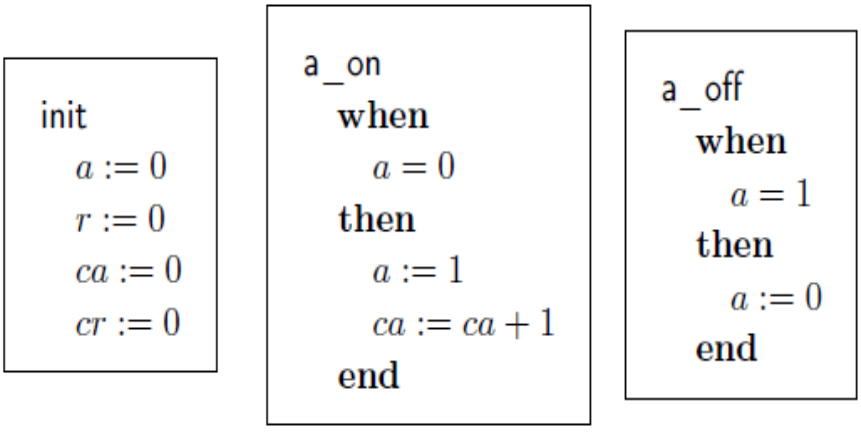
Figure 27 : Faible Synchronisation entre l'action et la réaction

Modélisation

En plus des variables, **a** et **r** désignant l'état de l'action et de la réaction (l'invariant pat0_1 et invariant pat0_2 ci-dessous). On a introduit deux compteurs: le premier est appelé **ca** associée à **a** et le second est nommé **cr** est associé à **r** (l'invariant pat0_3 et l'invariant pat0_4 ci-dessous). Ca et cr désignent respectivement le nombre de fois ou actions et réactions sont passées à l'état haut (le nombre de fronts montants). L'invariant pat 0_5, montre que **cr** est toujours inférieur à **ca**.



Les événements **a_on** et **a_off** sont propres à l'action a.

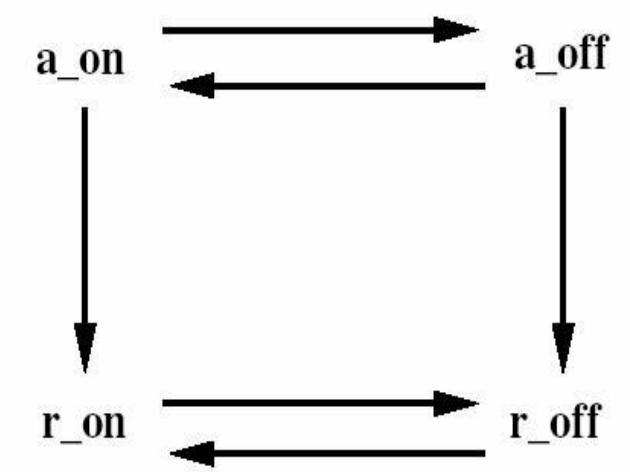


Les événements **r_on** et **r_off** sont propres à la réaction r. Ils sont synchronisés avec a_on et a_off.

```
r_on
  when
    r = 0
    a = 1
  then
    r := 1
    cr := cr + 1
  end
```

```
r_off
  when
    r = 1
    a = 0
  then
    r := 0
  end
```

Nous obtenons le schéma de transition ci-dessous. Les flèches indiquent les relations de précédences entre événements (par exemple, l'événement suivant a_on est soit a_off, soit r_on).



Ce modèle est complètement démontré par preuve mathématique.

III.4.3.2. Synchronisation forte :

Il s'agit d'un patron simple d'action et réaction simple avec rétroaction. Il est comparable au patron précédent, mais nous n'autorisons pas les cas décrits par la figure 27. La figure 28 montre une forte synchronisation entre l'action **a** et la réaction **r**.

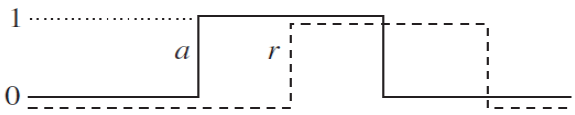
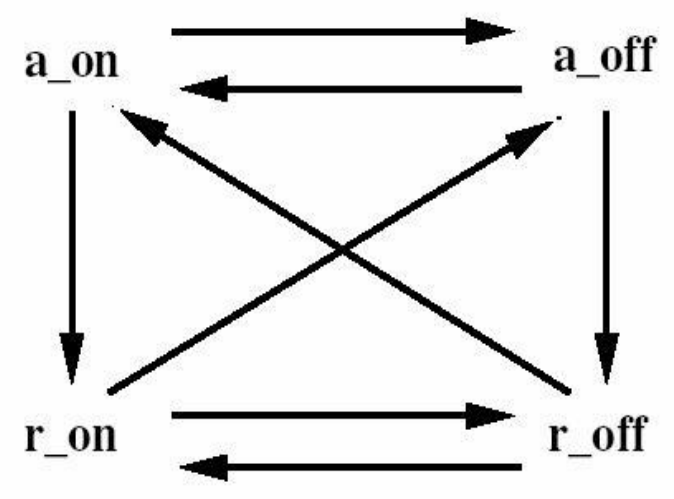


Figure 28 : Forte Synchronisation entre l'action et la réaction

Modélisation

Dans le cas de la forte synchronisation on a les mêmes variables que ceux de la faible synchronisation sauf que, on y ajoute un autre invariant **pat1_1** qui exprime le fait que **ca** ne dépasse pas **cr**.

Nous obtenons le schéma de transition ci-dessous



III.4.4. Exigences du système ABS

Nous avons défini les exigences liées aux équipements.

Les équipements de l'ABS sont : frein, temoin_ abs	EQP_1

Un bouton est utilisé pour démarrer et arrêter le moteur, la pédale frein pour engager et désengager le frein	EQP_2

Un contrôleur est censé gérer ces équipements	EQP_3

Ensuite, nous présentons la façon dont les équipements sont connectés au contrôleur:

Pédale du frein et le contrôleur sont faiblement synchronisés	FUN_1

Contrôleur et les équipements sont fortement synchronisés	FUN_2

Les deux principales exigences de sécurité

Lorsque la pédale du frein est engagée, le moteur doit être en marche	SAF_1

Quand le moteur est en marche, le témoin abs doit afficher l'état de l'ABS	SAF_2

La structure générale du système est présenté sur la figure 29 (frein ={étrier, disque})

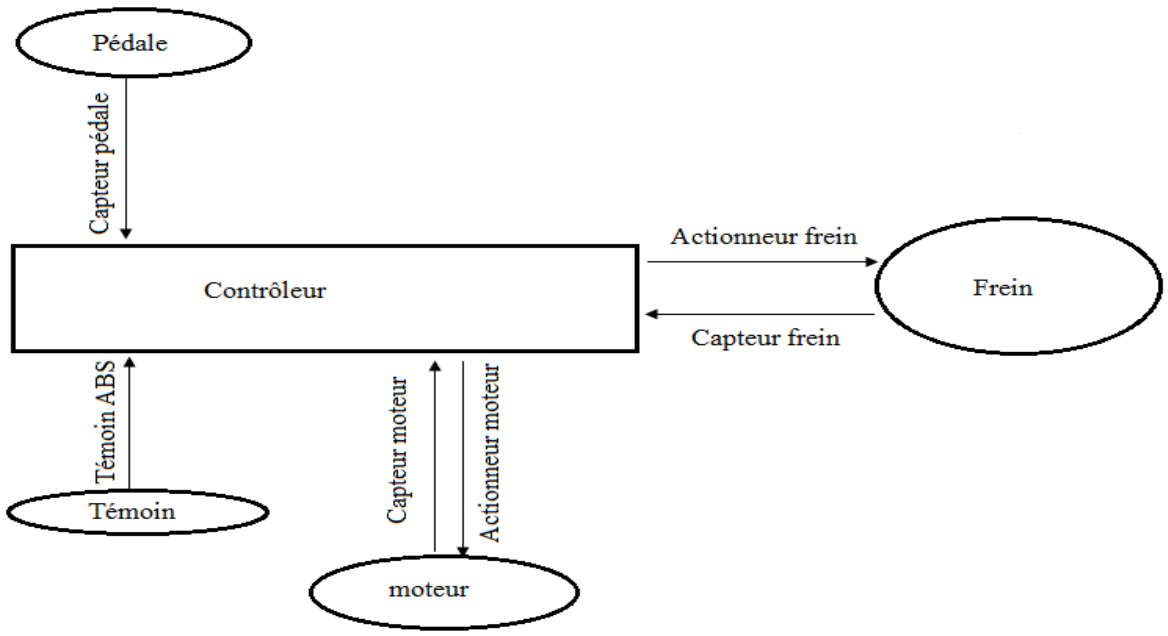


Figure 29 : les connexions du contrôleur

Nous allons développer le système d'ABS selon la stratégie suivante :

- → Modèle initial : lier le contrôleur au frein (étrier et disque)
- → Premier raffinement : lier la pédale de frein au contrôleur
- → Deuxième raffinement : lier le contrôleur au moteur

III.4.5. Modèle initial : Liaison entre le contrôleur et le frein

III.4.5.1. Introduction

Ce modèle initial formalise la connexion du dispositif de commande du frein comme cela est illustré sur la figure 30.

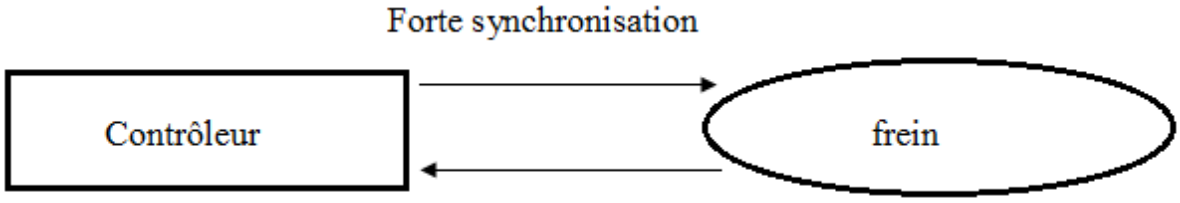


Figure 30 : liaison du contrôleur à l'Etrier & Disque

Contrôleur et les équipements sont fortement synchronisés	FUN_2

III.4.5.2. Modélisation

Les deux états du frein sont: {désactiver, activer}

Set :STATUS	Constants : activate deactivate	Axm0_1 :STATUS={activate,deactivate} Axm0_2 :activate≠deactivate

On a défini deux variables correspondant à la connexion du frein au contrôleur : brake_actuator et brake_sensor

Variables: brake_actuator brake_sensor	**inv0_1**:brake_sensor∈*STATUS* **inv0_2**:brake_actuator∈*STATUS*

La commande agit comme une action et le frein agit comme une réaction

a:	Brake _actuator
r:	brake_sensor
0 :	deactivate
1 :	activate
a_on :	treat_activate_brake
a_off :	treat_deactivate_brake
r_on :	brake_activate
r_off :	brake_deactivate

Les événements qui sont censés représenter l'action du contrôleur

```
a_on
when
        a=0
        r=0
then
a :=1
end
```

```
Treat_activate_brake
     When
Brake_actuator=deactivate
Brake_sensor= deactivate
      Then
Brake_actuator:=activate
end
```

```
a_off
when
   a=1
        r=1
then
a :=0
end
```

```
Treat_deactivate_brake
     When
Brake_actuator=activate
Brake_sensor= activate
      Then
Brake_actuator:=deactivate
end
```

Les événements qui représentent la réaction physique du frein

```
r_on
when
        r=0
        a=0
then
r :=1
end
```

```
brake_activate
     When
Brake_sensor= deactivate
Brake_actuator=activate
      Then
Brake_sensor:=activate
end
```

```
r_off
when
   r=1
        a=0
then
r:=0
end
```

```
brake_deactivate
     When
Brake_sensor= activate
Brake_actuator=deactivate
      Then
Brake_sensor:=deactivate
end
```

III.4.5.3. Résumé des événements

- Environnement

  brake_activate
  brake_deactivate

- Contrôleur

  Treat_activate_brake
  Treat_deactivate_brake

III.4.6. Premier raffinement : liaison entre la pédale de frein et le contrôleur

III.4.6.1. Introduction

Nous étendons maintenant la connexion introduite dans la section précédente en reliant la pédale au contrôleur. Ceci correspond à la figure 31.

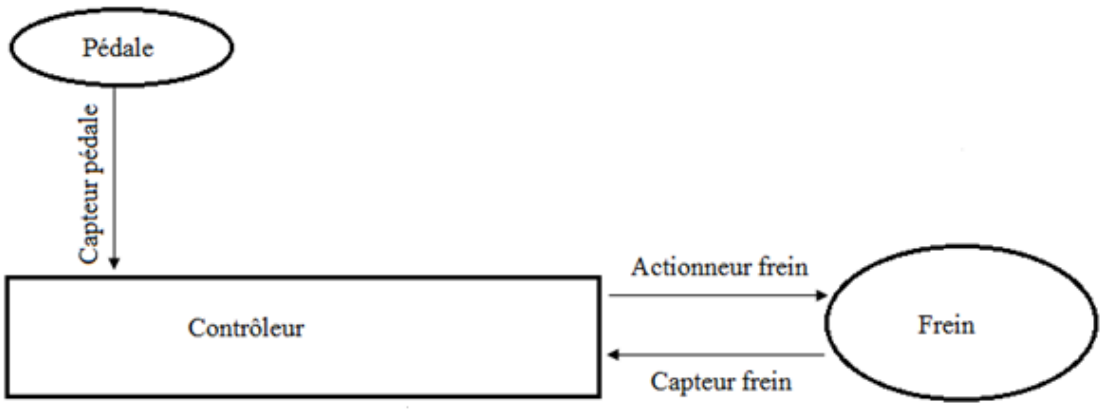


Figure 31: Connexion pédale contrôleur

On a pris en compte partiellement l'exigence FUN_1

Pédale et le contrôleur sont faiblement synchronisés	FUN_1

III.4.6.2. Modélisation

Nous définissons deux variables booléennes correspondant à la connexion de la pédale du frein au contrôleur: *activate_brake_pedal* et *deactivate_brake_pedal*. Ces variables physiques indiquent l'état de la pédale du frein, respectivement: quand il est égal à TRUE, cela signifie que la pédale du frein est physiquement déprimée; quand égale à FALSE, cela signifie qu'il est physiquement libéré.

Nous définissons deux autres variables booléennes: variables de commande du contrôleur: *acivate_brake_impulse* et *deacivate_brake_impulse*. Ces variables indiquent la connaissance par le contrôleur de l'état physique de la pédale du frein. Ils sont clairement distincts des deux variables précédentes, comme le changement de l'état physique d'un bouton se produit avant que le contrôleur peut être au courant:

```
Variables :
activate_brake_pedal
deactivate_brake_pedal
activate_brake _impulse
deacivate_brake _impulse
```

```
Inv1_1 :activate_brake_pedalϵBOOL
Inv1_2:deacivate_brake_pedalϵBOOL
Inv1_3:activate_brake _impulseϵBOOL
Inv1_4:deacivate_brake _impulseϵBOOL
```

Le comportement est clairement une instanciation de faible réaction, Ainsi, nous allons instancier le modèle de faible réaction comme suit:

a_on	push_activate_brake_pedal
a_off	release_activate_brake_pedal
r_on	treat_activate_brake
r_off	treat_release_activate_brake_pedal
a	activate_brake_pedal
r	activate_brake _impulse
0	FALSE
1	TRUE

Les deux premiers événements:

```
a_on
when
        a=0
then
a :=1
end
```

```
push_ activate_brake_pedal

 When
activate_brake_pedal  =FALSE
then
activate_brake_pedal  :=TRUE
end
```

a_off
when
a=1
then
a :=0
end

Release_ activate_brake_pedal
When
activate_brake_pedal =TRUE then
activate_brake_pedal :=FALSE

end

L'événement Treat_activate_brake, est l'instanciation d'une action dans le modèle initial, est maintenant l'instanciation d'une réaction. Il est renommé treat_push_activate_brake_pedal

r_on

when
r=0
a=1

then
r :=1
end

treat_push_activate_brake_pedal
refines
treat_activate_brake
when
activate_brake _impulse=FALSE
activate_brake_pedal =TRUE
brake_actuator=deactivate
brake_sensor=deactivate
then
activate_brake _impulse=TRUE
brake_actuator:=working

r_off

When

r=1
a=0
Then

r:=0

end

Treat_release_ activate_brake_pedal

When

activate_brake _impulse=TRUE
activate_brake_pedal =FALSE
then

activate_brake _impulse:=FALSE

end

L'événement abstrait treat_activate_brake :

Treat_activate_brake
When
Brake_actutor=deactivate
Brake_sensor=deactivate
Then
Brake_actutor=activate
end

Superposition du nouveau model avec le précédent

treat_push_activate_brake_pedal
refines
treat_activate_brake
when
activate_brake _impulse=FALSE
activate_brake_pedal =TRUE
brake_actuator=deactivate
brake_sensor=deactivate
then
activate_brake _impulse=TRUE
brake_actuator:=working

III.4.6.3. Résumé des événements

- Environnement
  - Brake_activate
  - Brake_deactivate
  - Push_activate_brake_pedal
  - Release_activate_brake_pedal
- Contrôleur
  - treat_push_activate_brake_pedal
  - Treat_release_ activate_brake_pedal

III.4.7. Deuxième raffinement : liaison entre le contrôleur et le moteur

Nous connectons maintenant le contrôleur au moteur. On suit exactement la même approche que celle que nous avons déjà utilisée pour la connexion du frein au contrôleur section 4.5

III.4.7.1. Résumé des événements

- Environnement
  - Brake_activate
  - Brake_deactivate
  - Motor_start
  - Motor_stop
  - Push_activate_brake_pedal
  - Release_activate_brake_pedal

- Contrôleur
    - treat_push_activate_brake_pedal
    - Treat_release_ activate_brake_pedal
    - Treat_start_motor
    - Treat_stop_motor

III.4.8. Conclusion

Dans cette partie, nous avons abordé la problématique de développement des systèmes complexes sensibles aux erreurs. Pour éviter les erreurs inhérentes au développement de ces systèmes, nous avons proposé un processus de développement formel (Event-B). La méthode formelle Event-B permet d'obtenir une spécification cohérente et digne de confiance du futur système en utilisant les outils associés à Event-B : générateur d'obligations de preuves, prouveur automatique et interactif, animateur.

Nous avons appliqué le processus de développement proposé sur le système de freinage ABS (Anti-lock brake systems) décrit [17]. Pour y parvenir, dans un premier temps, nous avons structuré le cahier des charges de l'étude de cas ABS en utilisant les recommandations méthodologiques proposées par J-R Abrial [3]. Dans un deuxième temps, nous avons établi une stratégie de raffinement adaptée à l'étude de cas ABS. Dans un troisième temps, nous avons spécifié en Event-B le modèle initial et les modèles raffinés de l'étude de cas ABS. Les propriétés de sûreté liées à ces modèles ont été prouvées. Nous avons dû utiliser l'animateur et le model-checker ProB afin de corriger nos modèles Event-B.

CHAPITRE IV : SPECIFICATION DE LA QUALITE DE SERVICE EN MECATRONIQUE

IV.1. Introduction

La qualité est un élément essentiel dans la caractérisation des produits et services. La définition et la mesure de la qualité, qui est facile dans certains contextes spécifiques, il devient très difficile lorsque nous décidons d'évaluer la qualité de quelque chose qui est «immatériel», tels que les services des systèmes mécatroniques. Par conséquent, il est nécessaire de clarifier et d'identifier les aspects clés de Qualité de Service (QoS) pour toutes les entités impliquées dans le service: le système, ses infrastructures et ses utilisateurs.

Une définition générale de la qualité est la suivante: «toutes les propriétés et caractéristiques d'une entité qui lui confèrent la capacité de répondre aux besoins exprimés ou implicites » [62].

Dans le contexte international, selon des ITU Recommandation E.800, la qualité de service est défini comme étant « l'effet collectif de la performance du service qui détermine le degré de satisfaction d'un utilisateur du service » et, du point de vue du système, en particulier, « la qualité de service est la capacité d'un réseau (ou d'un système) pour assurer un certain niveau de service » [63].

A partir de ces définitions, l'importance de la notion de qualité, comme valeur qui doit être garanti à l'utilisateur final, devient évidente [62]. Pour cette raison, il est essentiel que ce concept devient "commun", même s'il est difficile qu'un utilisateur soit vraiment conscient du niveau de qualité attendu d'un produit (ou plutôt, d'un service), dans un environnement dynamique comme les technologies de l'information.

La Qualité de Service (QoS) des systèmes mécatroniques est l'un des aspects clés de leur succès en raison de la grande compétitivité de l'environnement dans lequel elles sont déployées: quelle que soit la technologie de base, seuls les systèmes garantissant un degré de satisfaction à leurs utilisateurs seront acceptés. Ainsi, bien qu'on puisse trouver plusieurs définitions du terme qualité de service, dans le cadre de cette thèse, la qualité de service sera

comprise comme étant l'ensemble des aspects non fonctionnels d'un système qui déterminent le niveau de satisfaction des utilisateurs de la fonctionnalité qu'ils fournissent [64].

La gestion de la qualité de service désigne l'ensemble des activités consacrées à la surveillance et au contrôle des ressources impliquées dans la fourniture d'un niveau suffisant de qualité de service. C'est une tâche globale dans le sens où elle prend en compte tous les types de ressources supportant le service fourni par un système mécatroniques. Tout doit être synchronisé pour atteindre l'objectif global d'obtenir des niveaux de qualité de service qui répondent aux exigences des utilisateurs. Cette compréhension de la qualité de service est appelé «La QoS de bout en bout» [63]. QoS de bout en bout implique une traduction du haut en bas des exigences de QoS niveau utilisateur vers un niveau inférieur d'exigences de QoS.

Nous développons dans cette partie l'expression des besoins du système ODP en termes de qualité de service. Ces besoins sont en fait des qualités non fonctionnelles qui sont décrites au moment de l'établissement du cahier des charges, c'est-à-dire dans le point de vue Entreprise du modèle ODP et qui sont prises en considération lors du passage du modèle Traitement au modèle Ingénierie. Notre objectif principal est de traduire les spécifications de QoS employées pour négocier dynamiquement des accords de QoS entre les clients et les serveurs sur l'environnement cible à travers le raffinement des spécifications de QoS existantes. Pour bien illustrer cette stratégie de raffinement, nous allons spécifier formellement, dans IV.6, les systèmes mécatroniques.

IV.2. Point de Vue Ingénierie en mécatronique

Nous avons utilisé l'approche de méta-modélisation pour définir le mapping entre langage de point de vue ingénierie et les systèmes mécatroniques. Le tableau 2 présente un aperçu sur les transformations entre les concepts de langage du point de vue ingénierie à des artefacts mécatroniques.

Le point de vue Ingénierie (engineering viewpoint) est employé pour décrire des aspects répartition d'un système d'ODP. Il définit un modèle pour l'infrastructure distribuée du système. Ce modèle s'intéresse d'une part aux moyens pour prendre en charge l'interaction répartie des objets, et d'autre part aux mécanismes de placement des objets sur les ressources d'exécution.

Un système ODP est décrit dans le point de vue ingénierie comme un ensemble de nœuds interconnectés. Un nœud (node) est une configuration d'objet d'ingénierie qui constitue une

abstraction de ressources informatiques et des logiciels associés. Il est sous le contrôle d'un objet d'ingénierie appelé noyau (nucleus) qui fournit la fonction de gestion de nœud. Un nœud contient un ensemble de capsules. Une capsule est une configuration d'objets d'ingénierie constituant une abstraction d'un processus local à un nœud et de son espace d'adressage associé. Elle englobe des grappes, une grappe (cluster) correspondant à une configuration d'objets d'ingénierie de base constituée à des fins de persistance, de sauvegarde et de migration. Un objet de base est caractérisé par son identité unique, ses états, son comportement et ses interfaces et le fait qu'il requiert le support d'une infrastructure répartie.

La communication entre objets d'ingénierie de base est prise en compte par les canaux. Les canaux prennent donc en charge les liaisons dites réparties qui s'établissent entre objets de grappes différentes, voire de la même grappe. Un canal est une configuration d'objets d'ingénierie composée d'objets talons, lieurs, protocoles et intercepteurs.

Tableau 2– les transformations entre le point de vue ingénierie et les concepts mécatronique

Les concepts du langage d'ingénierie	**Les concepts de la mécatronique**
Control	PID connector/sensor/actuator
Node	Calculateur
Capsule	Address space
Cluster	basic objects forming a simple unit
Engineering Objects	Mechatronic objets / Modules d'un programmes
Engineering Interfaces	input capture card// output capture card
nucleus	Operating System
Stubs	connectors
Chanal	Connexion the mechatronic objects
Protocol objet	communications interface/ system bus

IV.3. QoS du point de vue Ingénierie

Les assertions de QoS dans ce point de vue comprennent l'ensemble d'activités exécutées par un système ou un service de communication pour supporter la surveillance, le contrôle (commande) et la gestion de la QoS. Ces activités QoS sont pilotées par les exigences de

l'utilisateur et de l'environnement des systèmes qui sont en vigueur. Les exigences de l'utilisateur sont mesurées et exprimées comme un ensemble de besoins en QoS [51].

Le point de vue ingénierie identifie les rôles que doivent jouer les objets ingénieries pour respecter les demandes en QoS exprimés dans les autres points de vue. Il est possible de qualifier les objets de liaison par des déclarations de qualité de service (bornes mises sur les délais de transmission de bout en bout ou sur la gigue du délai de bout en bout visibles à une interface de réception) qui en restreignent encore davantage le bon comportement.

Pour assurer la consistance lors de la modélisation des diverses qualités de service, nous nous sommes basés sur le cadre architectural de QoS ISO/IEC 13236, afin de définir un méta-modèle de référence dans le contexte UML figure 32. Ceci inclut :

- La catégorisation générale des différents genres de QoS ; y compris les spécifications de QoS définies lors de la conception ;
- L'identification des éléments conceptuels de base impliqués dans la QoS et leurs rapports mutuels. Ceci implique la capacité d'associer des caractéristiques de QoS aux éléments du modèle (spécification). Un modèle générique pour l'approvisionnement de la QoS (modèle de contrôle) est à associer aux cas d'utilisation (modèle d'utilisation).

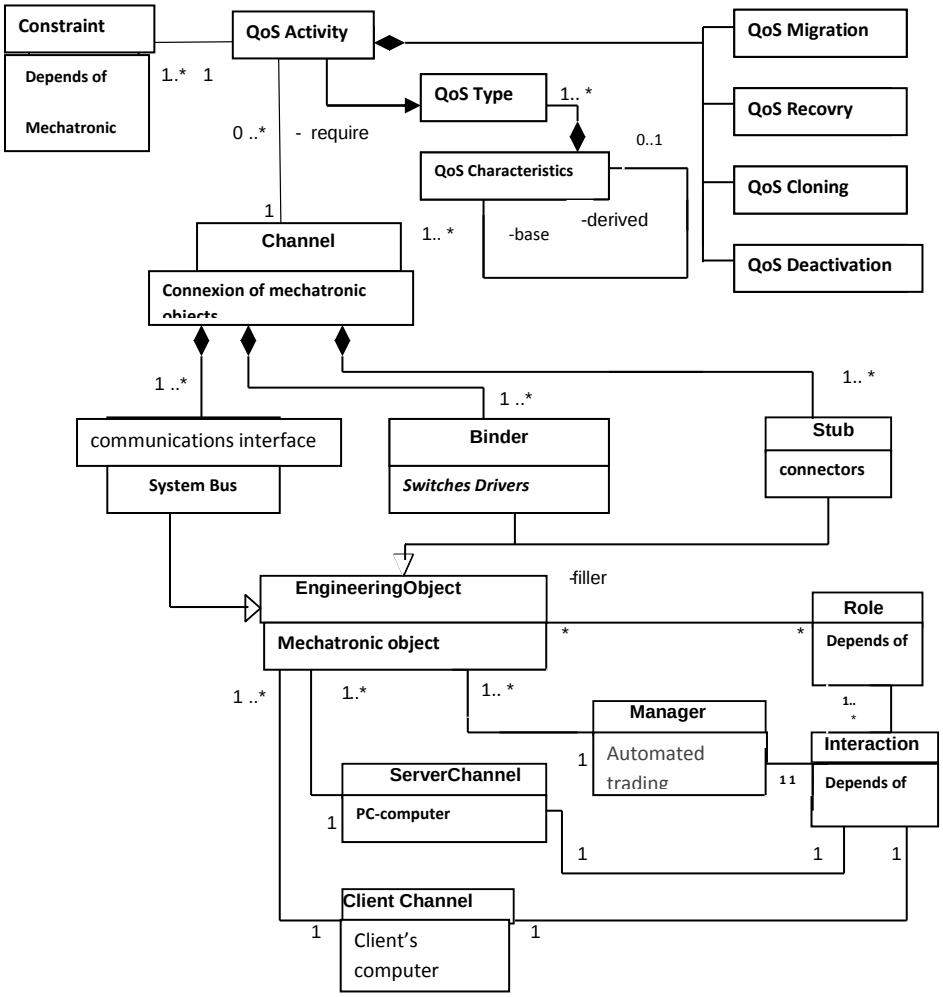


Figure 32 : Méta-modèle de la QoS au point de vue Ingénierie en mécatronique

IV.4. Formalisation d'un accord de QoS négocié

Notre préoccupation est de définir une organisation capable de piloter dynamiquement la QoS de bout en bout à travers ces différents niveaux de visibilité. Cet aspect organisationnel nous conduit à proposer comme rôle clé le Manager-QoS qui offre une autonomie de traitement de la qualité de service à un composant de service pour avoir une meilleure performance de la QoS au cours de la session de l'utilisateur. Il prend en charge le processus de négociation, il assure une coopération au niveau horizontal entre les composants de service quand il s'agit des composants du même niveau de visibilité, et une coordination pour assurer l'agrégation des besoins QoS entre les différents niveaux de visibilité.

IV.4.1. Négociation bornée

1) Le client spécifie un domaine de fonctionnement souhaité en indiquant au Manager-QoS une borne inférieure L et une borne supérieure U, avec $L \leq U$.

2) Le serveur peut rejeter la demande s'il sait que celle-ci ne peut pas être satisfaite, c'est-à-dire s'il ne peut pas prendre en charge au moins la valeur L de la borne inférieure. Si le serveur accepte la demande, mais ne peut pas assurer un fonctionnement dans la totalité du domaine proposé par l'utilisateur initiateur, il peut alors fixer une nouvelle valeur U ' réduite pour la borne supérieure : cette valeur réduite ne peut pas être moins bonne que la borne inférieure. Il en résulte que L <= U´ <= U. (Il est possible que le serveur choisisse de fonctionner de manière interne avec une qualité meilleure, mais ceci n'est pas signalé au trader.)

Le serveur ne peut pas modifier la borne inférieure L. La nouvelle borne supérieure U´ et la borne inférieure L sont indiquées au Gérant.

3) Le gérant-QoS peut rejeter la demande. Il peut, s'il l'accepte, négocier les paramètres QoS à maintenir entre les différents composants de service (Nœuds, Canaux,…) à activer dans les autres niveaux de visibilité pour assurer une solution de bout en bout.

Si ces composants acceptent la demande, mais ne peuvent pas assurer un fonctionnement avec une telle qualité, ils peuvent alors fixer une nouvelle valeur V' réduite : cette valeur ne peut pas être moins bonne que la borne inférieure. Il en résulte que L <= V´<= U´ (Il est possible que le Gérant choisisse de fonctionner avec une qualité inférieure à celle proposée par le réseau V < V´). La valeur choisie est renvoyée serveur.

4) Le serveur ne modifiera pas la valeur V choisie.

5) La valeur V choisie est renvoyée au client. Cette valeur devient alors la valeur "agréée".

Ce protocole est illustré dans la figure 33.

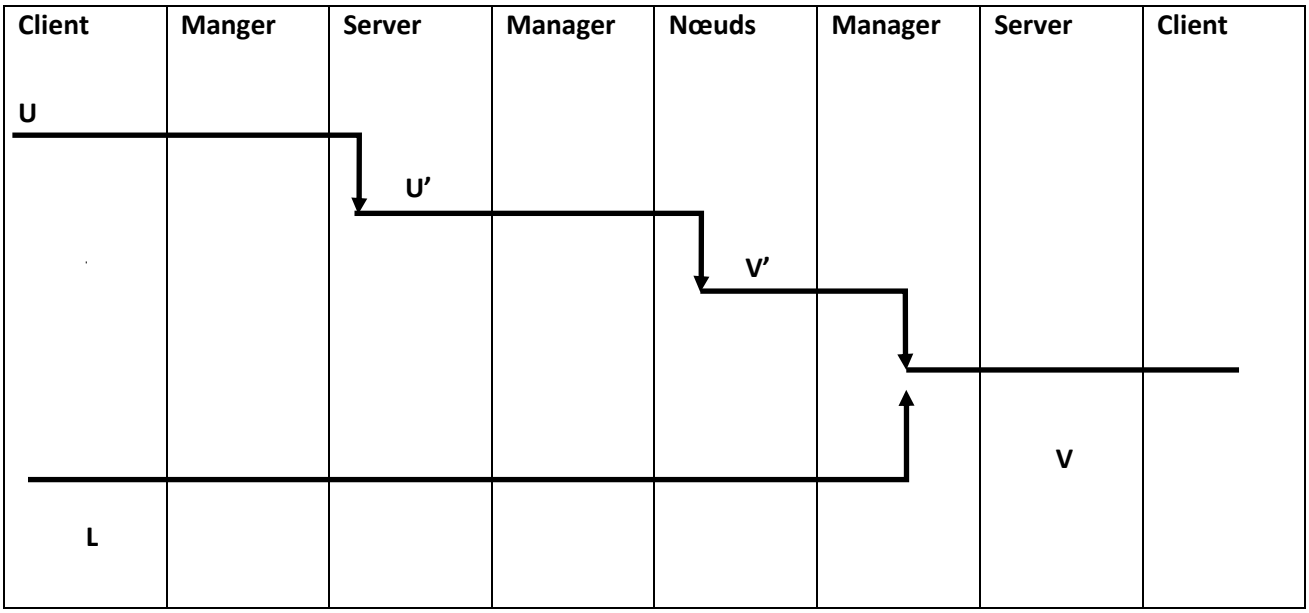


Figure 33 :Négociation bornée de la QoS

IV.4.2. protocole de négociation bornée

Dans la figure 34, nous identifions les différents composants impliqués dans le processus de négociation de la qualité de service de bout en bout. Ce processus permet une déclinaison Top-Down de la négociation QoS du niveau service (QoS du composant de service) jusqu'au niveau des équipements (QoS des équipements) en passant par le niveau réseau (La QoS des réseaux d'accès et de transport). Lorsqu'un utilisateur initie sa session, un processus de négociation de la QoS est déclenché pour identifier les composants de service selon le niveau de QoS qu'ils sont capables de fournir. En fait, le bout en bout suppose que les ressources réseaux devraient être prises en compte en tant que ressources de premier plan lors de la session de l'utilisateur. Sur la couche réseau, de nombreux protocoles sont utilisés pour déclencher le processus de renégociation de la QoS si une dégradation de la QoS réseau se produit.

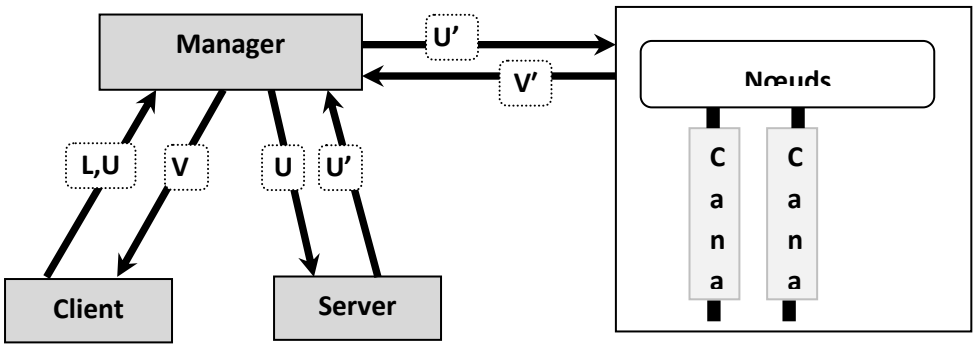


Figure 34 : Schéma du protocole de négociation de QoS bornée

IV.5. Spécification formelle des protocoles de négociation de QoS

Nous avons choisi la méthode formelle B événementiel (Event-B) pour spécifier et prouver la correction du protocole de négociation. En effet, la méthode B événementiel est directement liée à l'approche de conception : "correct-par-construction". Cette méthode garantit qu'au moment où le développement formel du système est terminé que celui-ci soit déjà vérifié. Cette approche propose à débuter le développement par un modèle abstrait du système et rajouter progressivement des détails pour générer à la fin un modèle très proche de l'implémentation.

IV.5.1. Document des exigences

Le document des exigences que nous proposons offre des explications informelles que nous avons données précédemment. Il ne propose pas une implémentation, mais consiste essentiellement à expliquer quelle intention que chaque objet d'ingénierie peut avoir à la fin du protocole :

Le système pilote la négociation de la QoS entre les objets basiques d'ingénierie	**FUN-1**

Puis nous expliquons ce que signifie une négociation achevée :

L'initiateur spécifie un domaine de fonctionnement souhaité en indiquant au serveur une borne inférieure L et une borne supérieure U, avec L < U. (Lorsqu'une limite de qualité LQA est négociée, la borne L	**FUN-2**

correspond à la valeur proposée pour la qualité LQA; lorsqu'une cible de fonctionnement est négociée, la borne U correspond à la valeur CHQ cible proposée.)	

La valeur finale de QoS est publiée par le gérant-QoS	**FUN-3**

Nous décrivons par l'intention **FUN-4**, ce que le client et le serveur peuvent croire à la fin du processus de négociation :

Les objets client et serveur doivent avoir la même intention de la réussite ou de l'échec de la négociation de la valeur de QoS	**FUN-4**

Par la suite, nous lions les intentions du client et du serveur :

Lorsque la valeur de QoS est publiée par le gérant-Qos, le client, les nœuds, les canaux et le serveur sont conscients que la négociation a réussi. Autrement, ils savent tous que la négociation a échoué.	**FUN-5**

A la fin, Nous présentons le cas où le client refuse la valeur proposée :

Cependant, si le client refuse la valeur de QoS proposée par le serveur, la négociation est abandonnée.	**FUN-6**

IV.5.2 Stratégie de raffinement

Nous présentons ci-dessous les principales étapes de notre démarche :

- le modèle initial met en place le protocole tenant compte des exigences FUN-1, FUN-2 et FUN-3. Il le présente comme si le protocole se fait d'un seul coup ;
- Dans le premier raffinement, nous prenons en considération les obligations restantes de FUN-4 à FUN-6 d'une manière abstraite. La QoS est approuvée étape par étape, mais nous sommes encore abstraits dans le sens où le client et le serveur peuvent avoir un accès direct aux états l'un de l'autre ;
- Dans le second raffinement, nous introduisons le Gérant-QoS entre le client et le serveur pour que chacun d'entre eux ne réagisse qu'avec les messages du Gérant: chacun ne peut accéder directement à la valeur de l'autre comme c'est le cas dans les raffinements précédents. Nous considérons aussi les entités réseau dessystèmes intermédiaires qui peuvent choisir un itinéraire parmi les itinérairesdisponibles.

- dans le troisième raffinement, nous considérons le temps de transmission de bout en bout entre les différents participants.

IV.5.3. Modèle abstrait de départ

Notre modèle initial spécifie l'objectif global du protocole de négociation de la valeur QoS bornée sans entrer dans le détail du traitement distribué. Il traite les exigences FUN-1, FUN-2 et FUN-3. Le protocole est exécuté en un seul coup.

IV.5.3.1. État

L'état de notre modèle est constitué de deux parties : une partie statique et une partie dynamique. La partie statique contient la définition et les axiomes associée à certaines constantes, la partie dynamique est représentée par les variables d'état de l'évolution du système. Les valeurs de QoS sont limitées par une borne inférieure **`Uchannel_min`** et une borne supérieure **`Uchannel_max`**, avec **`Uchannel_min`**<**`Uchannel_max`**, définis de façon simple dans l'axiome axm0_1, axm0_2.

Sets: **IR**	**Constants:** Uchannel_max Lchannel_min	**Axm0_1** : Uchannel_max ∈ IR **Axm0_2** : Lchannel_min ∈ IR

La valeur de QoS négociée est une variable typée dans les invariants inv0_1,inv0_2 et inv0_3.Ces deux invariants sont des prédicats qui demeurent valides tout au long de l'exécution du système. Ils servent à typer la variable **Uneg** et à spécifier les propriétés du système. Ils sont déclarés dans la clause INVARIANT.

variables**: Uneg**	**Inv0_1** : **Uneg**∈ IR Inv0_2**: Uneg <= Uchannel_max** Inv0_3**: Uneg > Lchannel_min**

IV.5.3.2. Événements

Les événements permettent de spécifier le comportement dynamique du système et d'initialiser ses variables. L'initialisation de la variable **Uneg** par la valeur **U**, est accomplie par le moyen d'un événement spécial appelé événement d'Initialisation. Sa valeur finale est définie de manière non-déterministe par l'événement QoSbrn, qui indique que la valeur Ufinal doit être inférieure ou égal à U et supérieure à L.

Init : Uneg := U Uchannel_max > 0 Uchannel_min > 0	**Brn :** Ufinal <= U Ufinal > L

IV.5.4. Premier modèle de raffinement

Un premier raffinement du modèle initial consiste à introduire des nouvelles variables et de nouveaux événements qui codifient les nouveaux comportements qui peuvent être aperçus à ce niveau, sans contredire le modèle initial. Nous introduisons la notion du canal ainsi que la valeur de la QoS requis par le canal afin d'assurer une liaison fiable entre les objets d'ingénierie (Exigences FUN-4 à FUN-6).

IV.5.4.1. État

Dans ce premier raffinement, nous introduisons la notion de statut. Pour ce faire, nous introduisons un ensemble nommé **STATUS**. Il est composé de trois éléments distincts: working, success and failure, comme illustré ci-dessous:
Nous remplaçons la variable abstraite Vnegoc par V_channel indiqué dans l'invariant **inv1_1**. Puis, nous introduisons respectivement les statuts v_channelet v_sever des objets canal et serveur. Ces statuts sont indiqués par les trois invariants inv1_2, inv1_3 et inv1_4.

Variables:	**Inv1_1** : 0 < Vnegoc <= Vchannel_max
v_server	**Inv1_2** : v_server = success ⇔ v_channel >= V_server
v_channel	**Inv1_3** : v_client = success ==> v_client <= v_channel and v_server >= v_client
v_client	**Inv1_4**: v_server = failure ⇔ v_channel < v_client or v_server < v_client

L'exigence FUN-6 est formalisée dans inv1_4 et stipule que le client accepte lorsque le serveur accepte.

IV.5.4.2. Événements

Les nouveaux événements **Client_snd, Server_failure, Server_accept et Channel_refuse** permettent de spécifier une négociation répartie entre le client et le serveur tenant compte du réseau. Ces événements raffine clairement 'SKIP' (leurs actions désignées par de nouvelles variables), et aussi de préserver les invariants inv1_2 et inv1_3 concernant le statut du canal et le serveur. Formellement, les quatre événements sont décrits comme suit :

Client_snd	Client_ failure
When	**When**
V_client = working	v_channel = propose
v_channel = accept	v_server = failure
v_server = accept	**Then**
Then	V_client := failure
V_client := accept	**End**
End	

Server_accept	Server_failure
When	**When**
v_channel = accept	V_client = working
	v_channel = failure
Then	**Then**
v_server := accept	v_server := failure
End	**End**

L'événement BRP défini ci-dessous, raffine également SKIP. Il représente une abstraction mais pas une implémentation.

Init :	**brp**
Vnegoc := P	**When**
Vserver_max > 0	V_Client ≠ working
Vchannel_max > 0	v_server ≠ working
V_Client:=required	v_channel ≠ working
v_channel := prescribed	**Then**
v_server := proposed	skip
	End

Le raffinement de l'événement brp doit affiner son abstraction, qui est un événement non-déterministe.

IV.5.5. Second modèle de raffinement

Dans ce raffinement, le Gérant-QoS va entrer en scène en coopérant avec les objets client, serveur et nœuds pour négocier la valeur de QoS. En fait, le client ne pourra plus accéder directement à la valeur du serveur comme c'était le cas dans le raffinement précédent, cela sera fait par l'intermédiaire du gérant. Nous introduisons alors l'objet gérant qui se situe entre le client et le serveur en se servant des différents éléments du réseau.

IV.5.5.1. État

L'état est d'abord élargi avec une variable de qualité de service du trader: Vbinder désignée et typée par les axiomes Inv2_1, Inv2_3 et Inv2_4. Nous introduisons une autre variable booléenne Publish indiquée implicitement par inv2_1 et inv2_2.Suite à l'ajout de cette variable, nous serons capables d'observer plus d'événements, à savoir les actions du gérant.

variables: required Vbinder	**Inv2_1:** required=true ==> Vbinder <= Vchannel_max **Inv2_2** : Vbinder > 0 **Inv2_3** : Vbinder <= Vserver **Inv2_3** : Vbinder <= Vclient

IV.5.5.2. Événements

L'événement d'initialisation est étendu de façon simple comme il est indiqué ci-dessous. La valeur booléenne publie la valeur False au début afin que les deux seuls événements qui peuvent être tirées sont celles décrites ci-après.

Init : Vnegoc := P Vserver_max > 0 V_server := proposed V_channel:= prescribed required := FALSE	Client_accept **When** V_client = required Publish = TRUE **Then** V_server := accept **End**	Server_accept **When** V_server := proposed Publish = TRUE **Then** V_client := accept **End**
Client_refuse **When** V_client = required V_client>> v_channel Publish = FALSE **Then** V_client := refuse **End**	Server_refuse **When** V_server = proposed V_server>>v_channel Publish = FALSE **Then** V_server := refuse **End**	Binder_accept **When** V_server~ = v_channel V_client ~ = V_server Publish = FALSE **Then** V_binder := accept **End**

IV.6. Cas d'étude : Système Mécatronique

La mécatronique [52] est la combinaison synergique et systémique de la mécanique, de l'électronique et de l'informatique en temps réel. L'intérêt de ce domaine d'ingénierie interdisciplinaire est de concevoir des systèmes automatiques puissants et de permettre le contrôle de systèmes complexes.

Avec la croissance rapide de la technologie, les fonctionnalités proposées par les constructeurs des systèmes mécatroniques sont devenus de plus en plus complexes, ces derniers sont confrontés à de nouveau problèmes avec ces fonctions, qui dépendent de plusieurs calculateurs et actionneurs répartis sur des réseaux informatiques internes.
Les dispositifs mécatroniques sont utilisés pour piloter des systèmes et rétroagir pour s'adapter aux conditions variables de fonctionnement, pour surveiller leur état (sollicitation, fatigue…), réaliser leur maintenance, … Ce domaine très vaste reste complexe (accès aux informations, compétences en électronique et mécanique, répartition des fonctions, fiabilité et sécurité…). Il nécessite des moyens importants et entraîne des coûts élevés.

Dans ce travail nous nous concentrons sur les problèmes d'ingénierie et réingénierie des systèmes mécatroniques. Notre contribution repose sur deux parties la première consiste à soumettre à la communauté mécatronique un cadre unificateur, le RM-ODP. Ce cadre est défini comme un méta-standard qui vise à établir des normes qui permettent de tirer profit des avantages de la distribution de l'information dans un environnement possédant des unités hétérogènes de traitement de l'information et qui relèvent de différents domaines d'ingénierie [53]. Dans la deuxième partie, nous proposons une méthode formelle afin de vérifier les lois qui gèrent la sûreté de fonctionnement des systèmes mécatroniques, ces lois sont formalisées en B événementiel.

Nous avons construit pas-à-pas une spécification formelle en Event-B d'un système mécatronique. Pour y parvenir, nous avons appliqué notre stratégie de raffinement proposée dans le chapitre 1. Une telle stratégie a été appliquée avec succès et a donné naissance à une machine abstraite, trois machines raffinées et trois contextes. En outre, nous avons validé la spécification formelle obtenue en utilisant avec profit la plateforme RODIN [54].

IV.7. Conclusion

Pour établir une comparaison avec les étapes classiques du développement logiciel, on peut considérer que les activités relevant des points de vue Entreprise, Information et Traitement à celles d'analyse, celles du point de vue Ingénierie à celle de la conception et celles du point de vue Technologie à celle d'implantation.

D'un point de vue ingénierie, une application répartie est un ensemble d'objets d'ingénierie de base interagissant. Un objet d'ingénierie de base constitue alors la représentation ingénierie d'un objet de traitement qui nécessite le support d'une infrastructure répartie.

Nous avons fourni par ailleurs la spécification formelle d'ingénierie permettant au constructeur de décrire l'environnement considéré pour implanter le protocole de négociation bornée de la qualité de service par l'intermédiaire du gérant-QOS en utilisant la méthode event-B. Cette description permet alors de paramétrer la spécification comportementale pour la transformer en spécification opérationnelle, modèle de l'application spécifique à un environnement d'exécution, à partir duquel le code peut être généré et déployé dans l'environnement. Pour bien illustrer notre stratégie de formalisation et de raffinement, nous avons spécifié formellement le système mécatronique. Ses concepts s'appuient sur le traitement réparti ouvert. ODP est une norme pour la construction de tout système distribué et n'est pas spécifique au monde des mécatroniques. Le but de nos travaux est de raffiner et d'adapter la norme ODP afin qu'elle soit applicable aux services de mécatronique en particulier. En effet, un service de mécatronique est une application distribuée qui s'exécute sur les différents nœuds d'un environnement possédant des unités hétérogènes de traitement de l'information et qui relèvent de différents domaines d'ingénierie.

CHAPITRE V : MODELISATION ET TRANSFORMATION DU PROCESSUS COMPORTEMENTAL ODP DANS LE CONTEXTE DE MDA

V.1. Introduction

La technologie des services Web repose essentiellement sur une représentation standard des données (interfaces, messageries) au moyen du langage XML. Cette technologie est devenue la base de l'informatique distribuée sur Internet et offre beaucoup d'opportunités au développeur We. Un des avantages majeurs des services Web par rapport à ses prédécesseurs tels que DCOM et XML-RPC est l'apport de l'interopérabilité et la portabilité sur les systèmes distribués ouverts.

Dans ce contexte, nous présentons notre démarche pour le développement des systèmes répartis ODP sur les plates-formes services Web. Cette dernière consiste à séparer les aspects indépendants et dépendants de la plate-forme par une description séparée de leurs modèles. Cette tendance est mise en avant par l'approche de l'architecture dirigée par les modèles (MDA - Model Driven Architecture), définie par l'OMG (Object Management Group). Nous avons choisi UML(Unified Modeling Language) comme formalisme pour décrire les aspects indépendants de la plate-forme (modèle métier), et les méta modèles pour décrire les aspects dépendants de la plate-forme des services Web (modèle deplate-forme). Les apports de ce travail sont : la séparation explicite entre spécification de correspondances et définition de transformations ; l'étude de la plate-forme des Services Web dans le contexte de MDA; la proposition des spécifications de correspondances entre les méta modèles de UML et des services Web; les définitions de transformations générées à partir de ces spécifications.
Ce chapitre est structuré de la manière suivante : la section 2 présente, le langage BPEL. La section 3 décrit la modélisation du comportement du système par un profil UML spécifique. Dans la section 4, nous définissons un mapping entre les concepts du comportement du langage d'information ODP et les concepts de BPEL tout en présentant la structure et la syntaxe d'un processus du comportement BPEL. Ainsi, la transformation des modèles UML

du comportement vers les BPEL correspondants va être assurée par l'utilisation de l'outil Rational rose.

V.2. Langage d'exécution des processus métiers

- Management des processus métiers

L'objectif de BPM (Business Process Management) est de permettre aux décideurs, analystes métiers, équipes fonctionnelles et équipes techniques de collaborer pour la définition et la modélisation des processus métiers via un seul outil fédérateur. Un des objectifs clés du BPM est de capitaliser sur les applications du système d'information : le mot d'ordre est la réutilisabilité. La réutilisabilité est rendue effectivement possible, tant au niveau technique (connecteurs, règles techniques, transformation de données) que fonctionnel (réutilisation d'un processus).

Un processus métier est défini généralement selon trois niveaux :

- Le niveau métier : le niveau haut de la vue métier du processus, définissant ses principales étapes et l'impact sur l'organisation de l'entreprise. Ce niveau est défini par les décideurs, et les équipes méthodes ;
- Le niveau fonctionnel : formalisation des interactions entre les participants fonctionnels du processus où sont formalisées les règles métiers conditionnant son déroulement. Ce niveau est modélisé par les équipes fonctionnelles ;
- Le niveau technique : établit le lien entre les activités (modélisés dans le niveau fonctionnel) et les applications du SI. Ce niveau est réalisé par les architectes et les équipes techniques de l'entreprise.

Pour la prise en compte de l'analyse et de la modélisation des processus métiers, l'OMG a développé son approche qui s'inscrit dans l'architecture générale de modélisation sous le nom de MDA **(**Model Driven Architecture**)**.

La MDA est un cadre de définition et de transformation des méta-modèles, de spécification des notations associées ainsi que d'échange des modèles et de leurs diagrammes sous un format normalisé(XMI).L'OMG a lancé l'initiative de spécification des processus métiers– BPDM qui s'appuie sur les modèles d'activités d'UML 2.0 en prenant en charge les différents niveaux d'analyse des processus : implémentation, organisation et stratégie. Ainsi la BPMI a publié la spécification BPMN **(**Business Process Modeling Notation**)** 1.0, qui présente une avancée indéniable dans la formulation graphique des processus.

Cependant, le véritable défi se situe dans la capacité des processus, dont la description a été formalisée par les acteurs métier, de déboucher directement sur la génération de langages exécutables par des applicatifs informatiques et des services web. Le langage BPEL4WS est une spécification de standards pour l'orchestration des web services annoncée en août 2002 par BEA, IBM et Microsoft.

- Langage de spécification

Le BPEL est la représentation XML d'un processus exécutable, qui peut être déployée sur n'importe quel moteur de processus métier. Ce langage est construit en se basant sur le WSFL (Web Services Flow Language) d'IBM et le XLANG (Web Services for Business Process Design) de Microsoft. Il fournit un langage de spécification des processus métiers et des protocoles de leur interaction [55]. Il combine les dispositifs d'un langage de processus structuré par bloc (XLANG) avec ceux d'un langage de processus graphique (WSFL). BPEL est prévu pour décrire un processus de métiers de deux manières différentes: abstraite et exécutable. Le processus abstrait est un protocole de métiers spécifiant le comportement d'échange de message entre différentes parties, sans préciser leur comportement interne. Le processus exécutable spécifie l'ordre d'exécution entre un certain nombre d'activités, les partenaires impliqués, le message échangé entre ces partenaires et les mécanismes de gestion d'erreur et d'exception.

La composition d'un service au niveau BPEL est décrite en termes de processus. Chaque élément du processus est appelé une activité. BPEL fournit deux genres d'activités [56] : Les activités primitives et les activités structurées.

V.3. Les concepts de comportement

Pour que le système réparti se comporte comme prévu, tous ses composants doivent partager une même compréhension de l'information qu'ils échangent au cours de leurs communications. Certains de ces éléments d'information sont manipulés d'une manière ou d'une autre par un grand nombre d'objets du système. Afin d'assurer une interprétation cohérente de ces éléments d'information, le langage d'information définit la sémantique de l'information et la sémantique de traitement de l'information dans un système ODP. Celui-ci est exprimé en termes d'une configuration d'objets d'information. Les relations entre les objets d'information sont modélisées comme des objets d'information ou comme parties des objets d'information. Une spécification consiste en un ensemble de gabarits d'objet d'information. Un gabarit est décrit en termes de trois schémas qui sont le schéma d'invariant, le schéma statique et le schéma dynamique. Le schéma d'invariant est un ensemble de

prédicats sur un ou plusieurs objets d'information qui doivent toujours être vrais. Le schéma statique spécifie l'état d'un ou plusieurs objets à un instant donné. Le schéma dynamique spécifie les changements d'états autorisés par un ou plusieurs objets d'information. Ces changements peuvent inclure la création ou la suppression d'objets d'information. Les schémas statique et dynamique doivent respecter les contraintes de tout schéma d'invariant.

Nous allons considérer les concepts de modélisation de base nécessaires à la spécification du comportement des objets d'information. Pour spécifier le comportement des systèmes répartis, nous représentons un système concurrent par le triplet : actions, l'ensemble de comportements, processus [57]. Chaque comportement est modélisé comme est une collection d'actions auxquelles peut prendre part cet objet, associée au jeu de contraintes qui portent sur les circonstances dans lesquelles ces actions peuvent se produire. Le modèle par objets n'impose pas de contraintes quant à la forme ou à la nature du comportement de l'objet. Les actions peuvent être soit des interactions entre l'objet et son environnement, soit des actions internes. Une action et une séquence finie ou infinie des états interchangeables.

Les concepts d'état et de comportement sont liés. L'état d'un objet est la condition de cet objet à un instant donné. Il détermine les séquences d'actions dans lesquelles il est possible que l'objet soit impliqué. En même temps, les actions induisent des transitions d'état. Par conséquent, l'état d'un objet est déterminé en partie par son comportement passé. Les actions qu'un objet va effectivement entreprendre ne sont naturellement pas dictées par son seul état présent, mais vont dépendre aussi des actions auxquelles son environnement est en mesure de participer. Dans la figure 35, nous définissons le méta-modèle d'UML, pour les concepts de comportement et d'information. Ce modèle représente l'ensemble des objets d'information, leur relation et leur comportement.

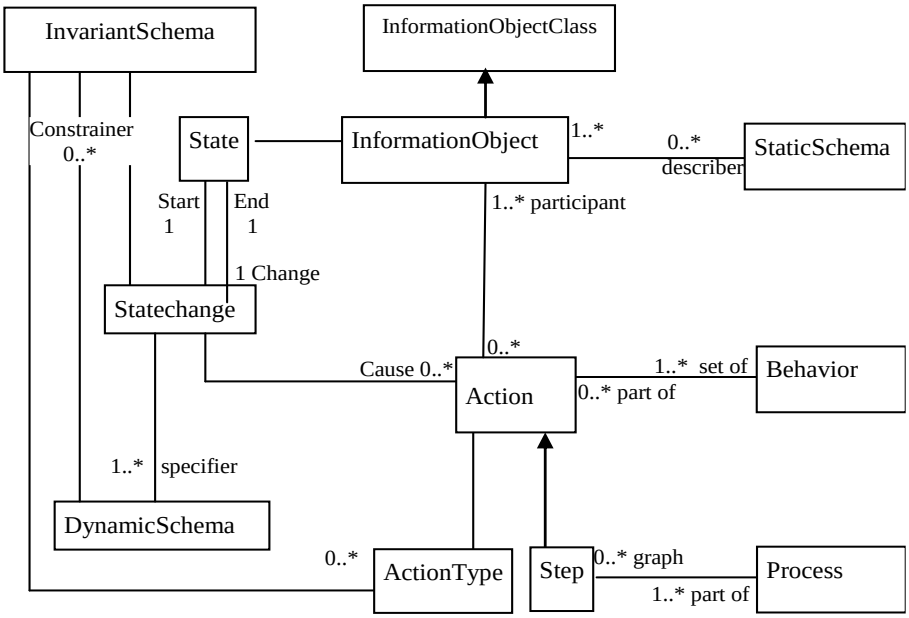


Figure 35 : Méta-Modèle du langage d'information-Concepts de comportement

Pour spécifier l'action en UML/OCL, nous avons considéré deux autres concepts de modélisation : le temps et l'état. La liaison entre ces deux concepts et la notion d'action est définie dans la clause 9 da la partie 2 du modèle RM-ODP.

On a défini comme suit la sémantique précondition et postcondition de chaque action en utilisant le langage de contraintes OCL [58]:

context Time inv :

forall(o:InformationObject ,t:Time | t.instant ->notEmpty implies o.state ->notEmpty)

context Precondition inv :

Forall (prec: Dynamicschema.Precondition , o : InformationObject|exists(s : State) | o.mappedTo = prec and o.state_start = s)

context Postcondition inv :

forall (postc: dynamicschema.Postcondition , o : InformationObject | exists(s : State) | o.mappedTo = postc and a.state_end = s)

V.4. Profil UML pour des processus automatisés de comportement

L'existence de modèles de composants technologiques dont les plus connus sont CCM (CORBA Component Model) de l'OMG, EJB de Sun ainsi que COM/DCOM et .Net de Microsoft a donné naissance à l'extension du méta-modèle d'UML 2.X. et à la création de profil.

Pour l'OMG, un profil est un ensemble de stéréotypes, de valeurs marquées (tagged-value) et de contraintes. Ces éléments permettent d'adapter dynamiquement un méta-modèle à un domaine d'intérêt particulier. C'est dans ce contexte que nous avons développé un nouveau profil UML pour la modélisation du comportement dans le langage d'information des systèmes ODP. Le profil proposé est basé essentiellement sur la notion de stéréotype. En effet, le but ultime des stéréotypes est de combler les manques d'UML par l'extension de la notation à des points spécifiques. Le développeur peut créer des stéréotypes et les personnaliser en leur donnant un sens clair et précis à l'aide du langage OCL[59].

Nous présentons ci-dessous un profil UML pratique, qui supporte la modélisation d'un ensemble de constructions sémantiques du comportement dans le langage d'information, tout en donnant leurs correspondants en langage d'exécution de processus des métiers BPEL Tableau 3 [58].

Tableau 3 : Mapping entre concepts de comportement et constructions UML

Behavior Concepts	Profile Construct
Process	<< process>> class
Behavior	Activity graph on a <<process>> class
Action	<<metaclass>> signal
Role	<<partner>> class
static Schema	<< metaclass >> statemachine
Invariant Schema	<< metaclass >> constraint
dynamic Schema	<< metaclass >> package

Dans le profil UML, un processus est représenté comme classe avec le stéréotype <<Process>>. Le stéréotype <<IV_Process>> étend le métaclass Activity. Il est destiné à capturer la sémantique d'un processus dans la langue de l'information RM-ODP.Les attributs de la classe correspondent à l'état du processus (variables dans BPEL 1.1). La classe UML représentant le processus de comportement est illustrée dans la Figure 36.

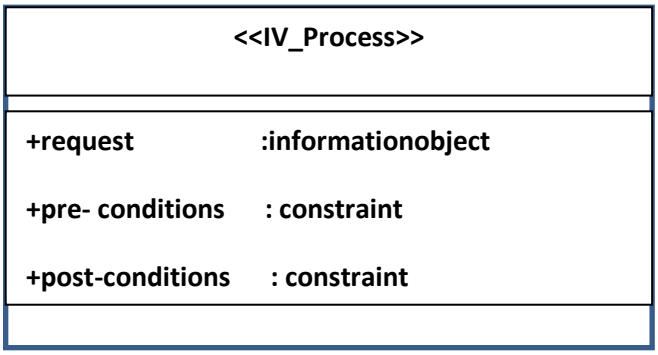


Figure 36 : Une classe UML modélise le processus du comportement BPEL

V.5. Génération d'un processus à partir d'un modèle UML

- Mapping entre UML et BPEL

Le profil UML pour les processus de comportement automatisés montre comment on peut générer des artéfacts BPEL exécutables à partir des modèles UML [55]. Le mapping entre profil UML développé et concepts BPEL est illustré dans le Tableau 4 :

Tableau 4 : Mapping entre constructions UML et concepts BPEL

Profile Construct	**BPEL Concept**
<< process>> class	BPEL process definition
Activity graph on a <<process>> class	BPEL activity hierarchy
<<process>> class attributes	BPEL variables
Hierarchical structure and control flow	BPEL sequence and flow activities
<<receive>>, <<reply>>, <<invoke>>activities	BPEL activities

BPEL est une représentation XML d'un processus exécutable qui peut être déployé sur n'importe quel moteur de processus. L'élément atomique d'un processus BPEL est l'« activité », qui peut être l'envoi d'un message, la réception d'un message, l'appel d'une opération (envoi d'un message, tentative d'une réponse), ou une transformation des données.

Selon le BPEL, les activités réalisées dans le cadre de l'exécution du processus de comportement sont définies en XML. Nous présentons ci-dessous la structure et la syntaxe de processus de comportement proposé (**IV_behavior**) :

<**IV_behavior**>

< roles />→definition of the actors

<containers/> →definition of the containers of the data

<invariant schema /> →Aset of predicates which must always be true.

<static schema /> → A configuration of information objects.

<transitioncondition>

<dynamic schema /> →Astate changes of one or more information objects.

</transitioncondition>

</**IV_behavior**>

<IV_**process**>

< partners /> →definition of the partners (actions)

<containers/> →definition of the containers of the data

<sequence />

<receive /> →reception of a request

<assign /> →transformation of the data

<invoke /> →call of an action

<reply /> →sending of an answer

</sequence>

</IV_**process**>

<**schema**> name = "nameschema"

<process name ="process"/>

< action name = "action"/>

<constraint type ="pre-conditions"/>

<constraint type ="post-conditions"/>

</**schema**>

- Transformation de la spécification de processus au BPEL

Nous présentons alors la démarche technique de transformation de modèles, entre les langages UML et BPEL, pour l'automatisation du processus de comportement. Cette technique est basée sur l'idée qu'un modèle de processus, développé dans un outil UML tel que IBM Rational's XDE or Rose, peut être convertit en fichiers BPEL et WSDL nécessaires pour l'implémentation de ce processus. La transformation est alors conduite selon les éléments constituant le profil. Elle préconise l'utilisation de deux types de fichiers qui sont les modèles UML qui peuvent être ouverts et modifiés avec les outils **Rose** ou **XDE**, et les fichiers XML contenant la version XMI des modèles UML, qui sont exportés par **Rose** ou **XDE** [60].

La Figure 37 fait appel à un diagramme d'activité UML pour présenter le processus global de transformation des fichiers; les rectangles représentent les artefacts (généralement des fichiers), tandis que les ellipses représentent une action ou une activité.

Notre démarche repose sur les étapes suivantes :

1. Spécification du modèle UML (CIM)
2. Exportation des diagrammes UML en XMI (PIM)
3. Génération du BPEL, des fichiers comportement, et des fichiers XSD (PIM)
4. Création de la base de données d'objets d'information ;
5. Déploiement des artefacts produits en BPEL moteur (PSM).

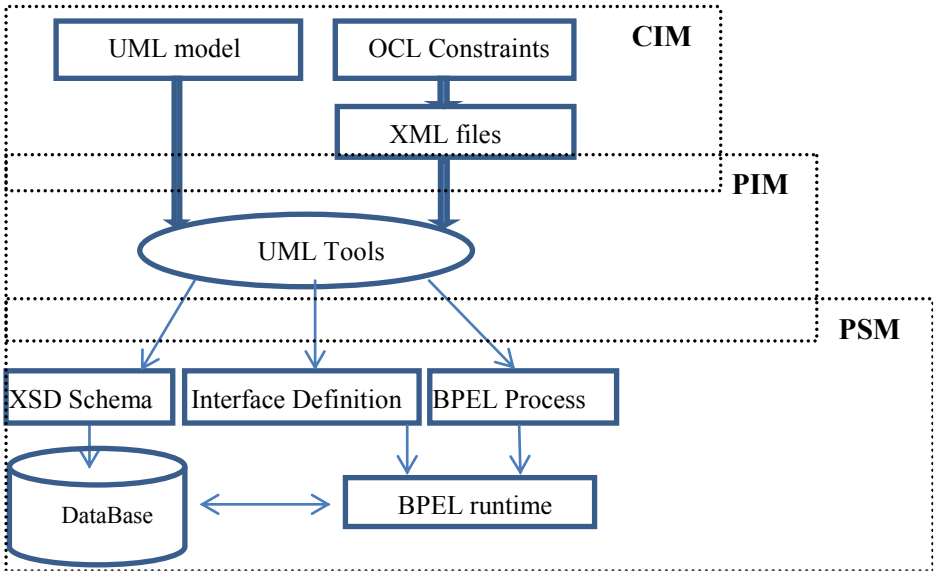


Figure 37 : Développement d'un processus

V.6. Conclusion

Dans le cadre d'automatisation de production des logiciels, qui offre une meilleure qualité, et une transparence vis-à-vis des changements possibles de la technologie, nous avons proposé une méthodologie de modélisation du comportement d'un système distribué, orientée MDA. Cette approche met l'accent sur l'élaboration des modèles de plus haut niveau d'abstraction et favorise l'approche de transformation d'un modèle à l'autre. A cet effet, nous avons modélisé le comportement du système ODP par un profil UML spécifique, ce profil permet aux développeurs d' utiliser la notation UML et les outils appropriés, puis nous avons proposé les méta-modèles des deux langages d' information et BPEL tout en présentant la structure et la syntaxe d'un processus du comportement BPEL. Ensuite, nous avons proposé des spécifications de correspondances et nous avons défini des règles de transformation assurant le passage automatique entre les différents niveaux d'abstraction (conceptuel à logique, logique à physique). Ce qui nous a permis de générer automatiquement les processus exécutables BPEL à partir des modèles UML. Cette transformation est de type transformation de méta-modèle décrite par l'approche MDA.

CONCLUSION GENERALE ET PERSPECTIVES

La modélisation et le développement des systèmes complexes et interopérables conformes à un cahier des charges incluant des politiques de sécurité et des exigences de qualité, requiert des méthodes, des techniques et des outils permettant de s'assurer de l'adéquation du système développé aux exigences du cahier des charges. Nous avons choisi d'utiliser la norme de traitement réparti ouvert (ODP) comme un cadre architectural pour la construction des systèmes mécatroniques et applications réparties et les techniques formelles pour les concevoir. Ce choix est basé sur les tendances actuelles de l'ingénierie, à savoir la définition d'architecture du système, la spécification formelle et l'architecture dirigée par les modèles (MDA). La définition d'architecture système est une étape intermédiaire entre l'analyse des besoins et la réalisation du système. Les méthodes formelles associées à sa définition permettent de vérifier les propriétés structurelles, comportementales et de qualité. Quant à l'architecture dirigée par les modèles elle fournit des mécanismes permettant d'enrichir des modèles par transformation.

Dans le second chapitre du présent travail, Nous avons utilisé le standard UML pour construire un méta-modèle pour le langage d'entreprise en mécatronique. Il nous a permis de représenter les concepts de base de point de vue entreprise et leurs relations d'une manière claire, concise et cohérente. Ce qui permet la communication et le partage du savoir faire entre les acteurs ayant des connaissances variés. Nous avons utilisé l'UML pour définir la structure statique des communautés, à savoir, les rôles et leurs relations. Ce meta-modèle du langage d'entreprise pour la mécatronique permet aux concepteurs de formaliser leur exigences sur le comportement des objets d'entreprise qui remplissent des rôles dans les communautés. De toute évidence, il y a beaucoup d'autres aspects, tels que la qualité du service, fiabilité et la sécurité, qui pourrait également être soumis à la spécification de la politique.

Dans le troisième chapitre, nous avons abordé la problématique de développement des systèmes complexes sensibles aux erreurs. Pour éviter les erreurs inhérentes au développement de ces systèmes, nous avons proposé un processus de développement formel (Event-B). La méthode formelle Event-B permet d'obtenir une spécification cohérente et

digne de confiance du futur système en utilisant les outils associés à Event-B : générateur d'obligations de preuves, prouveur automatique et interactif, animateur. Nous avons appliqué le processus de développement précité sur le Système de freinage ABS (Anti-lock brake systems) décrit dans [17]. Pour y parvenir, dans un premier temps, nous avons structuré le cahier des charges de l'étude de cas ABS en utilisant les recommandations méthodologiques proposées par J-R Abrial [3]. Dans un deuxième temps, nous avons établi une stratégie de raffinement adaptée à l'étude de cas ABS. Dans un troisième temps, nous avons spécifié en Event-B le modèle initial et les modèles raffinés de l'étude de cas ABS. Les propriétés de sûreté liées à ces modèles ont été prouvées. Nous avons dû utiliser l'animateur et le model-checker ProB afin de corriger nos modèles Event-B.

Dans le quatrième chapitre nous avons développé le processus de négociation de la QoS entre les objets d'un système ODP, en utilisant la fonction de courtage. Nous avons proposé d'introduire un assistant de négociation dynamique dans le terminal de l'utilisateur afin de l'aider, d'une part, à choisir le meilleur fournisseur de services et, d'autre part, à négocier dynamiquement les paramètres de qualité de service (QoS) répondant aux exigences de l'utilisateur et de l'application. Le modèle que nous avons élaboré se base sur la méthode formelle Event-B. L'intérêt de Event-B dans notre étude réside dans sa modélisation permettant d'exprimer formellement des propriétés validées par preuves pendant la conception des modèles du système, mais également dans son principe de raffinement permettant de maîtriser la complexité du système par un développement progressif et sûr. Notre démarche de prise en compte de la qualité de service comprend la définition d'un cadre pour la spécification et l'évaluation de la qualité qui couvre tous les aspects du développement (Points de vue entreprise ODP) et le développement d'un environnement dédié à la qualité qui met en œuvre l'approche qualité proposée et permet l'automatisation de la mesure de la qualité (Point de vue ingénierie ODP).

Dans le dernier chapitre, nous avons décrit la méthodologie de modélisation des aspects comportementaux du langage d'information ODP. La démarche commence par la création d'un profil UML spécifique au comportement. Ce profil sera transformé en artefacts BPEL en définissant un mapping entre les concepts du comportement du langage d'information et les concepts du langage BPEL. Cette transformation, orientée MDA, va être assurée par l'utilisation de l'outil Rational rose.

Les travaux réalisés ouvrent de nombreuses perspectives, à court terme comme à long terme, afin d'aider le concepteur à définir l'ensemble des aspects d'un système ODP.
Pour les travaux futurs, nous allons généraliser notre approche à d'autres concepts des systèmes ODP. Ce sera notre base pour une investigation plus approfondie de l'utilisation de la méthode event-B dans le processus de conception des systèmes ODP. En outre, des études de cas devraient être développées en utilisant ces modèles.

Nous projetons, également de généraliser notre approche de négociation de la QoS à d'autres méthodes de négociation de qualité de service dans les systèmes mécatroniques en l'occurrence aux négociations multi-parties 1 x N et M x N de QoS.

REFERENCES BIBLIOGRAPHIQUES

1. ISO/IEC, ‘’Basic Reference Model of Open Distributed Processing-Part1: Overview and Guide to Use, ‘’ISO/IEC CD 10746-1, 1994.
2. ISO/IEC, ‘’RM-ODP-Part4: Architectural Semantics, ‘’ ISO/IEC DIS 10746-4, July 1994.
3. J. Abrial. The B-book: assigning programs to meaning, Cambridge university press, 1996.
4. RODIN. Development Environment for Complex Systems (Rodin). 2009. http://rodin.cs.ncl.ac.uk/.
5. Martin Törngren, Ahsan Qamar, Matthias Biehl, Frederic Loiret, Jad El-khoury- Integrating viewpoints in the development of mechatronic products- Mechatronics Volume 24, Issue 7, October 2014, Pages 745–762.
6. Abrial J-R., Le Cahier des Charges : Contenu, Forme et Analyse (en vue de la Formalisation), http://www.liafa.jussieu.fr/~sighirea/cours/genielog/Cdc.pdf, Juin 1987.
7. Xanthakis S., Regnier P., Karapolios C., *Le test des logiciels*, Hermès-Lavoisier, 1999. deploy-eprints.ecs.soton.ac.uk/186/1/multi_refine.pdf
8. B ertram, T., Bekes, F., Greul, R., Hanke, O., Hab, C., Hilgert, J.,Hiller, M., Ottgen, O., Torlo, M., et Ward, D. (2003). Modelling andsimulation for mechatronic design in automotive systems. Control Engineering Practice,11 :179–190.
9. DesJardin, L. (1996). A day in the life of mechatronic engineers 10 years from now. In SAE International Congress and Exposition, number SAE96C038, Detroit/Michigan, USA.
10. Yaskawa-Electric, C. (1969). http ://www.yaskawa.co.jp/encompany/rekisi.htm.
11. Comerford, R. (1994). Mecha...what ? IEEE Spectrum, pages 46–49.
12. Grimheden, M. et Hanson, M. (2001). What is mechatronics? proposing a didactical approach to mechatronics. In 1st Baltic Sea Workshop on Education in Mechatronics, Kiel, Germany.
13. Hewit, J. (1996). Mechatronics design - the key to performance enhancement. Robotics and Autonomous Systems, 19 :135–142.
14. Millbank, J. (1993). Mecha-what ! Mechatronics Forum Newsletter, 6.
15. Breedveld, P. C. (2004). Port-based modeling of mechatronic systems. Mathematics and Computers in Simulation, 66 :99–127.
16. Thesame (2003). La mécatronique à l’épreuve du marché. Jitec.
17. Rieuneau, F. (1993). Sûreté de fonctionnement en phase de developpement des systèmes embarqués automobiles. In Integrated Logistics & Concurrent Engineering, Montpellier.

18. Shetty, D. et Kolk, R. (1997). Mechatronics System Design. PWS Publishing Company, USA.
19. Siemers, C., Falsett, R., Seyer, R., et Ecker, K. (2005). Reliable event-triggered systems for mechatronic applications. Journal of Systems and Software, 77(1) :17–26.
20. ISO/IEC, ‘’RM-ODP-Part2: Descriptive Model, ‘’ ISO/IEC DIS 10746-2, 1994.
21. ISO/IEC, ‘’RM-ODP-Part3: Prescriptive Model, ‘’ ISO/IEC DIS 10746-3, 1994.
22. J-R. Putman. Architecting with RM-ODP, Prentice-Hal, 2001.
23. M.-P. Gervais, Méthodologie ODAC: Le guide de spécification comportemental, Rapport de recherche LIP6 2001.
24. J-R. Putman, Architecting with RM-ODP, Prentice-Hal, 2001.
25. Summary of the Event-B Modeling Notation: http://www.cse.unsw.edu.au/~cs2111/PDF/sld_evtb.pdf
26. Chill R., Logique et théorie des ensembles, Laboratoire de Mathématiques et Applications de Metz, Université de Metz, 2007/08.www.math.univ-metz.fr/~chill/logique.pdf
27. Troudi I., Développement formel des algorithmes séquentiels en Event-B, Master, FSEGS, 2011.
28. Robinson K., A Concise Summary of the Event B mathematical toolkit, Version April 21, 2009 . http://wiki.event-b.org/images/EventB-Summary.pdf
29. Summary of Event-B Proof Obligations http://www.computing.dcu.ie/~hamilton/teaching/CA648/sld_po.pdf
30. Wirth N., Program Development by Stepwise Refinement, P. Wegner Editor, Volume 14 Number 4, 221-227, ETH Zurich, April 1971.
31. Abrial J-R., Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.
32. F. Jean-Marie, E. Jacky and B. Mireille : L’Ingénierie Dirigée par les Modèles : au-delà du MDA. Informatique et Systèmes d’Information. Hermès Science, lavoisier édition, février 2006.
33. Hervé Panetto, Meta-modèles et modèles pour l’intégration et l’interopérabilité des applications d’entreprises de production, rapport de thèse ,2006.
34. J. Miller and J. Mukerji: Model Driven Architecture (MDA) 1.0.1 Guide. Object Management Group, Inc., juin 2003.
35. B. Combemale (2008). Approche de métamodélisation pour la simulation et la vérification de Modèle Application à l’ingénierie des procédés, Thèse de doctorat, spécialitéinformatique,l’Université de Toulouse, 198 p .
36. Object Management Group, Inc. Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification, version 1.0, avril 2008.

37. ISOrIEC, Open Distributed Processing — Reference Model — Part 2: Foundations. International Standard 10746-2, ITU-T Recommendation X.902, January 1995.
38. A. Balouki;M. Sbihi; Y. Balouki. RM-ODP: A framework for Mechatronics systems.16th International Conference on Research and Education in Mechatronics (REM), Pages: 73 – 79, IEEE Conference Publications,November 2015
39. A. Johar and R. Stetter INTERNATIONAL DESIGN CONFERENCE - DESIGN 2008 Dubrovnik –(Croatia, May 19 - 22, 2008.
40. ISO/IEC, ‘’Basic Reference Model of Open Distributed Processing-Part1: Overview and Guide to Use, ‘’ISO/IEC CD 10746-1, 1994.
41. ODP enterprise viewpoint specification M.W.A. Steen, J. Derrick) Computing Laboratory, UniÍersity of Kent at Canterbury, Canterbury CT2 7NF,UK Accepted 13 January 2000
42. ISOrIEC, Open Distributed Processing — Reference Model — Part 2: Foundations. International Standard 10746-2, ITU-T Recommendation X.902, January 1995.
43. ISOrIEC. Open Distributed Processing — Reference Model— Enterprise Viewpoint. WorkingDraft 15414, ITU-T Rec-ommendationX.911, January 1999.
44. S. Kent, S. Gaito, N. Ross, A meta-model semantics for structural constraints in UML, in: H. Kilov, B. Rumpe, I. Simmonds Eds. ., Behavioral Specifications for Businesses and Systems, Kluwer Academic Publishers, Norwell, MA, 1999, pp. 121–139, September,Chap. 9
45. Imen Sayar, Mohamed Tahar Bhiri: From an abstract specification in event-b toward an UML/OCL model. FormaliSE 2014
46. User manuel of the rodin platform: http://wwww.cs.man.a.uk/rodinmanuel.pdf
47. Siemers, C., Falsett, R., Seyer, R., et Ecker, K. (2005). Reliable event-triggered systems for mechatronic applications.Journal of Systems and Software 77(1) :17–26.
48. Xanthakis S., Regnier P., Karapolios C., Le test des logiciels, Hermès-Lavoisier, 1999. deploy-eprints.ecs.soton.ac.uk/186/1/multi_refine.pdf
49. Have we learned from the Vasa Disaster? : http://videotorium.hu/en/recordings/details/1674,Have_we_learned_from_the_Wasa_disaster
50. ISO/IEC, ‘’RM-ODP-Part2: Descriptive Model, ‘’ ISO/IEC DIS 10746-2, 1994.
51. H.Soulimani (2012). Pilotage Dynamique de la Qualité de Service de Bout en Bout pour une Session «User Centric», Thèse de doctorat, spécialité informatique, l’Institut ParisTech.
52. A.Balouki, A.Ettabbaa, A.Elhassnaoui, M.Mabrouki, M.Fakir.Mechatronic Enterprise Viewpoint Specifications.Journal of Theoretical and Applied Information Technology,accépté pour publication le 31st July 2016. Vol. 89 No.2.
53. ISO/IEC, ‘’Basic RM-ODP-Part1: Overview and Guide to Use, ‘’ ISO/IEC CD 10746-1, 1994
54. G. Myers, The art of Software Testing , John Wiley &Sons, 1979

55. keith_mantell.:From UML to BPEL Model Driven Architecture in a Web services world Report IT Architect, IBM 2003

56. B. Rumpe.: Agile Modeling with UML, '' LNCS vol. 2941, Springer, 2004, pp. 297-309.

57. ISO/IEC, "Use of UML for ODP system specifications", ISO/IEC 19793, may 2006.

58. A.Balouki, et al, Transformation of Models in an MDA Approach for Collaborative Distributed Processes. Collaborative Systems for Reindustrialization, IFIP Series Advances in Information and Communication Technology, Vol. 408, pp. 201-208, Springer, October 2013.

59. Y. Balouki and Mohamed Bouhdadi :Using BPEL for Behavioural Concepts in ODP

Enterprise Language, Virtual Enterprises and Collaborative Networks, IFIP, Vol. 283, pp. 221-232, Springer, 2008.

60. L. Briand , ''A UML-based Approach to System testing, '' LNCS vol. 2185. Springer, 2001, pp. 194-208,

61. M.W.A. Steen, J. Derrick ODP enterprise viewpoint specification, Computer Standards & Interfaces 22 2000. 165–189

62. M. Di Micheli, G. Cotta. Qualité de service dans les réseaux de valeur des TIC et de la sécurité pour l'utilisateur, ISCOM (Institut supérieur de communication et de technologie de l'information) March 2005-2006).

63. ITU-T Recommendation E.800 – Term and definition related to Quality of Service and network performance including dependability (August 1994).

64. ISO/IEC TR 13243 – Information technology – Quality of service –Guide to methods and mechanism (November 1999).